

Библиотека профессионала

PYTHON

СОЗДАНИЕ ПРИЛОЖЕНИЙ

ТРЕТЬЕ ИЗДАНИЕ

Вы уже знаете язык Python, но хотите узнать больше? Намного больше? Погрузитесь в разнообразие тем, связанных с реальными приложениями

Книга охватывает регулярные выражения, сетевое программирование, графические пользовательские интерфейсы, SQL/базы данных/ORM, потоки и веб-программирование

Узнайте больше о современных трендах программирования, таких как Google+, Twitter, MongoDB, OAuth, Python 3 и Java/Jython

В книге представлен новый материал о каркасе Django, платформе Google App Engine, форматах CSV/JSON/XML и приложениях Microsoft Office

Книга содержит примеры программ на Python 2 и Python 3, готовых к использованию!

В книге много фрагментов кода, интерактивных примеров и практических упражнений



УЭСЛИ ДЖ. ЧАН

Core

PYTHON

APPLICATION PROGRAMMING

THIRD EDITION

WESLEY J. CHUN



PRENTICE
HALL

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Библиотека профессионала

PYTHON

СОЗДАНИЕ ПРИЛОЖЕНИЙ

ТРЕТЬЕ ИЗДАНИЕ

УЭСЛИ ДЖ. ЧАН



Москва • Санкт-Петербург • Киев
2015

Издательский дом "Вильямс"

Зав. редакцией С.Н. Тригуб

Перевод с английского О.Л. Пелявского, К.А. Птицына

Под редакцией докт. физ.-мат. наук Д.А. Ключина

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Чан, Уэсли.

Python: создание приложений. Библиотека профессионала, 3-е изд. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2015. — 816 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1793-5 (рус.)

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Prentice Hall, Inc.

Authorized translation from the English language edition published by Prentice Hall, Inc., Copyright © 2012 Pearson Education, Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2015

Научно-популярное издание

Уэсли Чан

Python: создание приложений Библиотека профессионала, 3-е издание

Литературный редактор *И.А. Попова*

Верстка *Л.В. Чернокозинская*

Художественный редактор *В.Г. Павлютин*

Корректор *Л.А. Гордиенко*

Подписано в печать 23.01.2015. Формат 70×100/16.

Гарнитура Times.

Усл. печ. л. 65,79. Уч.-изд. л. 50.

Тираж 300 экз. Заказ № 374.

Отпечатано способом ролевой струйной печати

в ОАО «Первая Образцовая типография»

Филиал «Чеховский Печатный Двор»

142300, Московская область, г. Чехов, ул. Полиграфистов, д.1

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1793-5 (рус.)

ISBN 978-0-13-267820-9 (англ.)

© Издательский дом "Вильямс", 2015

© Pearson Education, Inc., 2012

Оглавление

Предисловие	
Об авторе	
Часть I. Общие прикладные темы	33
Глава 1. Регулярные выражения	35
Глава 2. Сетевое программирование	83
Глава 3. Программирование интернет-клиентов	121
Глава 4. Многопоточное программирование	181
Глава 5. Программирование графического пользовательского интерфейса	235
Глава 6. Программирование баз данных	273
Глава 7. Программирование приложений для работы с Microsoft Office	339
Глава 8. Создание расширений для языка Python	379
Часть II. Разработка веб-приложений	403
Глава 9. Веб-клиенты и веб-серверы	405
Глава 10. Веб-программирование: интерфейсы CGI и WSGI	451
Глава 11. Веб-платформы: Django	499
Глава 12. Облачные вычисления: Google App Engine	605
Глава 13. Веб-службы	679
Часть III. Дополнительная и экспериментальная	705
Глава 14. Обработка текста	707
Глава 15. Разное	735
Приложение А. Ответы на некоторые упражнения	755
Приложение Б. Справочные таблицы	759
Приложение В. Версия Python 3: эволюция языка программирования	777
Приложение Г. Переход к версии Python 3 на основе выпуска Python 2.6+	785
Предметный указатель	801

Содержание

Предисловие

Об авторе

Часть I. Общие прикладные темы 33

Глава 1. Регулярные выражения 35

1.1. Общее назначение	36
1.1.1. Первое знакомство с регулярными выражениями	38
1.2. Специальные знаки и символы	39
1.2.1. Сопоставление нескольких шаблонов регулярных выражений с помощью оператора чередования ()	40
1.2.2. Сопоставление с любым отдельным символом (.)	41
1.2.3. Сопоставление с началом или концом строки или с границей слова (^, \$, \b, \B)	41
1.2.4. Создание классов символов ([...])	42
1.2.5. Формирование диапазонов (-) и отрицаний диапазонов (^)	43
1.2.6. Использование операторов замыкания (*, +, ?, { }) для представления нескольких вхождений/повторений	44
1.2.7. Специальные символы, обозначающие наборы символов	45
1.2.8. Обозначение групп с применением круглых скобок (())	46
1.2.9. Расширенный синтаксис регулярных выражений	47
1.3. Регулярные выражения и язык Python	48
1.3.1. Модуль re. Основные функции и методы	48
1.3.2. Компиляция регулярных выражений с применением функции compile()	50
1.3.3. Объекты сопоставления и методы group() и groups()	51
1.3.4. Согласование со строками с применением функции match()	51
1.3.5. Поиск шаблона в строке с помощью функции search() (поиск вместо сопоставления)	53
1.3.6. Сопоставление с несколькими строками ()	54
1.3.7. Сопоставление с любым отдельным символом (.)	54
1.3.8. Создание классов символов ([...])	55
1.3.9. Повторение, специальные символы и группирование	55
1.3.10. Сопоставление с началом и концом строк и с границами слов	58
1.3.11. Поиск каждого вхождения с помощью функций findall() и finditer()	58
1.3.12. Поиск и замена с помощью функций sub() и subn()	60
1.3.13. Разбиение (по шаблону разграничения) с помощью метода split()	61
1.3.14. Расширенный синтаксис (?...)	62
1.3.15. Разное	65
1.4. Некоторые примеры регулярных выражений	66
1.5. Более сложный пример регулярного выражения	72
1.5.1. Сопоставление со строкой	74
1.5.2. Сравнение поиска и сопоставления с учетом жадных выражений	76
1.6. Упражнения	79

Глава 2. Сетевое программирование 83

2.1. Введение	84
2.2. Что такое архитектура “клиент–сервер”	84
2.2.1. Аппаратная архитектура “клиент–сервер”	85
2.2.2. Программная архитектура “клиент–сервер”	85
2.2.3. Кассир банка как пример сервера	86
2.2.4. Сетевое программирование по принципу “клиент–сервер”	87

2.3. Сокеты: конечные точки связи	88
2.3.1. Общее определение понятия сокета	88
2.3.2. дреса сокетов: пара "хост-порт"	89
2.3.3. Сокеты с установлением и без установления соединения	90
2.4. Сетевое программирование на языке Python	91
2.4.1. Функция модуля socket ()	91
2.4.2. Методы объекта сокета (встроенные)	92
2.4.3. Создание сервера TCP	93
2.4.4. Создание клиента TCP	97
2.4.5. Эксплуатация сервера и клиентов TCP	101
2.4.6. Создание сервера UDP	102
2.4.7. Создание клиента UDP	103
2.4.8. Эксплуатация сервера и клиентов UDP	105
2.4.9. Атрибуты модуля socket	105
2.5. *Модуль SocketServer	107
2.5.1. Создание сервера TCP с применением модуля SocketServer	108
2.5.2. Создание клиента TCP на основе модуля SocketServer	110
2.5.3. Эксплуатация сервера и клиентов TCP	111
2.6. *Введение в концепцию Twisted	111
2.6.1. Создание сервера TCP на основе классов reactor инфраструктуры Twisted	112
2.6.2. Создание клиента TCP на основе классов reactor инфраструктуры Twisted	113
2.6.3. Эксплуатация сервера и клиентов TCP	115
2.7. Связанные модули	115
2.8. Упражнения	117
Глава 3. Программирование интернет-клиентов	121
3.1. Что такое интернет-клиенты	122
3.2. Передача файлов	123
3.2.1. Протоколы Интернета для передачи файлов	123
3.2.2. Протокол передачи файлов	123
3.2.3. Язык Python и протокол FTP	125
3.2.4. Методы класса ftplib.FTP	126
3.2.5. Пример программы для работы с протоколом FTP в интерактивном режиме	127
3.2.6. Пример клиентской программы FTP	127
3.2.7. Другие особенности протокола FTP	130
3.3. Сетевые новости	131
3.3.1. Usenet и группы новостей	131
3.3.2. Протокол передачи сетевых новостей	132
3.3.3. Язык Python и протокол NNTP	132
3.3.4. Методы класса nntplib.NNTP	134
3.3.5. Пример использования протокола NNTP в интерактивном режиме	135
3.3.6. Пример клиентской программы NNTP	135
3.3.7. Дополнительные сведения о протоколе NNTP	141
3.4. Электронная почта	141
3.4.1. Компоненты и протоколы почтовой системы	141
3.4.2. Отправка электронной почты	142
3.4.3. Язык Python и протокол SMTP	144
3.4.4. Методы класса smtplib.SMTP	145
3.4.5. Пример сеанса интерактивного взаимодействия с использованием протокола SMTP	146
3.4.6. Дополнительные сведения о протоколе SMTP	147
3.4.7. Получение электронной почты	147
3.4.8. Протоколы POP и IMAP	147
3.4.9. Язык Python и протокол POP3	149
3.4.10. Интерактивный пример работы по протоколу POP3	149
3.4.11. Методы класса poplib.POP3	150
3.4.12. Пример применения протоколов SMTP и POP3	151
3.4.13. Язык Python и протокол IMAP4	153
3.4.14. Интерактивный пример применения протокола IMAP4	154

3.4.15. Общие методы класса <code>imaplib.IMAP4</code>	154
3.4.16. Практический пример	156
3.5. Связанные модули	171
3.5.1. Электронная почта	171
3.5.2. Другие клиентские средства поддержки протоколов Интернета	172
3.6. Упражнения	172
Глава 4. Многопоточное программирование	181
4.1. Введение/общее назначение	182
4.2. Поток и процессы	184
4.2.1. Общее определение понятия процесса	184
4.2.2. Общее определение понятия потока	184
4.3. Поддержка потоков в языке Python	185
4.3.1. Глобальная блокировка интерпретатора	185
4.3.2. Выход из потока	186
4.3.3. Доступ к потокам из программы Python	186
4.3.4. Организация программы без применения потоков	187
4.3.5. Многопоточные модули Python	188
4.4. Модуль <code>thread</code>	189
4.5. Модуль <code>threading</code>	194
4.5.1. Класс <code>Thread</code>	195
4.5.2. Другие функции модуля <code>Threading</code>	202
4.6. Сравнение однопоточного и многопоточного выполнения	203
4.7. Практическое применение многопоточной обработки	205
4.7.1. Пример ранжирования книг	205
4.7.2. Примитивы синхронизации	213
4.7.3. Пример применения блокировки	213
4.7.4. Пример семафора	220
4.8. Проблема “производитель–потребитель” и модуль <code>Queue/queue</code>	225
4.9. Дополнительные сведения об использовании потоков	229
4.9.1. Модуль <code>subprocess</code>	229
4.9.2. Модуль <code>multiprocessing</code>	229
4.9.3. Модуль <code>concurrent.futures</code>	230
4.10. Связанные модули	232
4.11. Упражнения	232
Глава 5. Программирование графического пользовательского интерфейса	235
5.1. Введение	236
5.1.1. Что такое <code>Tcl</code> , <code>Tk</code> и <code>Tkinter</code>	236
5.1.2. Установка и ввод в действие интерфейса <code>Tkinter</code>	237
5.1.3. Архитектура “клиент–сервер” — два компонента	238
5.2. Библиотека <code>Tkinter</code> и программирование на языке Python	238
5.2.1. Модуль <code>Tkinter</code> , обеспечивающий реализацию интерфейса <code>Tk</code> в приложениях	238
5.2.2. Введение в программирование графического пользовательского интерфейса	239
5.2.3. Окно верхнего уровня: <code>Tkinter.Tk()</code>	241
5.2.4. Графические элементы <code>Tk</code>	242
5.3. Примеры <code>Tkinter</code>	243
5.3.1. Графический элемент <code>Label</code>	243
5.3.2. Графический элемент <code>Button</code>	244
5.3.3. Графические элементы <code>Label</code> и <code>Button</code>	245
5.3.4. Графические элементы <code>Label</code> , <code>Button</code> и <code>Scale</code>	246
5.3.5. Более реальный пример	247
5.3.6. Пример использования интерфейса <code>Tkinter</code> в более сложном приложении	251
5.4. Краткий обзор других графических пользовательских интерфейсов	257
5.4.1. Среда <code>Tk Interface eXtensions (Tix)</code>	259
5.4.2. Объекты Python <code>MegaWidgets (PMW)</code>	260
5.4.3. Модули <code>wxWidgets</code> и <code>wxPython</code>	261

5.4.4. Интерфейсы GTK+ и PyGTK	263
5.4.5. Модуль Tile/Ttk	265
5.5. Связанные модули и другие графические пользовательские интерфейсы	268
5.6. Упражнения	271
Глава 6. Программирование баз данных	273
6.1. Введение	274
6.1.1. Система постоянного хранения	274
6.1.2. Основные операции с базами данных и язык SQL	274
6.1.3. Базы данных и язык Python	277
6.2. Спецификация DB-API Python	279
6.2.1. Атрибуты модуля	280
6.2.2. Объекты класса Connection	282
6.2.3. Объекты класса Cursor	283
6.2.4. Объекты и конструкторы типов	285
6.2.5. Реляционные базы данных	286
6.2.6. Базы данных и Python: адаптеры	287
6.2.7. Примеры применения адаптеров баз данных	288
6.2.8. Пример приложения на основе адаптера базы данных	293
6.3. Объектно-реляционные преобразователи	306
6.3.1. Применение объектов вместо запросов SQL	306
6.3.2. Язык Python и объектно-реляционные преобразователи	306
6.3.3. Пример базы данных с описанием должностей сотрудников	308
6.3.4. SQLAlchemy	308
6.4. Нереляционные базы данных	325
6.4.1. Введение в NoSQL	326
6.4.2. База данных MongoDB	326
6.4.3. Адаптер PyMongo: MongoDB и Python	327
6.4.4. Резюме	331
6.5. Справочная информация	332
6.6. Упражнения	334
Глава 7. Программирование приложений для работы с Microsoft Office	339
7.1. Введение	340
7.2. Программирование клиентов COM на языке Python	341
7.2.1. Программирование клиентов COM	341
7.2.2. Вводные сведения	342
7.3. Вступительные примеры	343
7.3.1. Программа Excel	343
7.3.2. Программа Word	346
7.3.3. Программа PowerPoint	347
7.3.4. Программа Outlook	349
7.4. Промежуточные примеры	352
7.4.1. Програма Excel	353
7.4.2. Программа Outlook	355
7.4.3. Программа PowerPoint	362
7.4.4. Резюме	371
7.5. Соответствующие модули/пакеты	372
7.6. Упражнения	372
Глава 8. Создание расширений для языка Python	379
8.1. Введение/обоснование	380
8.1.1. Что такое расширение	380
8.1.2. Причины, по которым может потребоваться создание расширения Python	381
8.1.3. Причины, по которым целесообразно отказаться от создания расширения Python	382
8.2. Модули расширения	383
8.2.1. Создание прикладного кода	383

8.2.2. Оформление кода с применением стандартных шаблонов	385
8.2.3. Компиляция	391
8.2.4. Импорт и проверка	393
8.2.5. Подсчет ссылок	396
8.2.6. Многопоточная организация и GIL	398
8.3. Другие темы	398
8.3.1. Упрощенный генератор оболочек и интерфейса	398
8.3.2. Язык Pyrex	399
8.3.3. Язык Cython	399
8.3.4. Язык Psyco	399
8.3.5. PyPy	400
8.3.6. Внедрение	401
8.4. Упражнения	401
Часть II. Разработка веб-приложений	403
Глава 9. Веб-клиенты и веб-серверы	405
9.1. Введение	406
9.1.1. Навигация по веб-страницам с помощью средств архитектуры клиент-сервер	406
9.1.2. Интернет	407
9.2. Инструменты веб-клиентов Python	410
9.2.1. Унифицированный указатель информационного ресурса	411
9.2.2. Модуль urlparse	412
9.2.3. Модуль/пакет urllib	413
9.2.4. Пример аутентификации с помощью модуля urllib2 при установлении соединения по протоколу HTTP	418
9.2.5. Перенос примера аутентификации HTTP в Python 3	421
9.3. Веб-клиенты	423
9.3.1. Простой поисковый робот, спайдер, бот	423
9.3.2. Синтаксический анализ содержимого веб-страниц	430
9.3.3. Программирование инструментов для просмотра веб-страниц	436
Резюме	440
9.4. Веб-серверы (HTTP)	440
9.4.1. Простые веб-серверы Python	440
9.5. Связанные модули	444
9.6. Упражнения	446
Глава 10. Веб-программирование: интерфейсы CGI и WSGI	451
10.1. Введение	452
10.2. Средства обработки клиентских данных на веб-сервере	452
10.2.1. Введение в интерфейс CGI	452
10.2.2. Приложения CGI	454
10.2.3. Модуль cgi	454
10.2.4. Модуль cgitb	455
10.3. Создание приложений CGI	456
10.3.1. Установка веб-сервера	456
10.3.2. Создание страницы формы	458
10.3.3. Формирование страницы с результатами	459
10.3.4. Формирование страниц с формой и результатами	461
10.3.5. Полностью интерактивные веб-сайты	465
10.4. Использование стандарта кодирования Юникод с интерфейсом CGI	472
10.5. Расширения CGI	473
10.5.1. Передача многокомпонентной формы и передача файла	474
10.5.2. Многозначные поля	474
10.5.3. Cookie-файлы	475
10.5.4. Cookie-файлы и передача файлов	476

10.6. Введение в WSGI	485
10.6.1. Стимулы к дальнейшему развитию (альтернативы технологии CGI)	485
10.6.2. Интеграция сервера	485
10.6.3. Внешние процессы	486
10.6.4. Основные сведения о стандарте WSGI	487
10.6.5. Серверы WSGI	488
10.6.6. Эталонный сервер	489
10.6.7. Примеры приложений WSGI	490
10.6.8. Промежуточное программное обеспечение и создание оболочек для приложений WSGI	491
10.6.9. Изменения в интерфейсе WSGI в версии Python 3	492
10.7. Практическая разработка для веб	493
10.8. Связанные модули	494
10.9. Упражнения	495
Глава 11. Веб-платформы: Django	499
11.1. Введение	500
11.2. Веб-платформы	500
11.3. Введение в Django	502
11.3.1. Установка	503
11.4. Проекты и приложения	507
11.4.1. Создание проекта в Django	507
11.4.2. Работа с сервером для разработки	510
11.5. Первое приложение Hello World (блог)	512
11.6. Создание модели для добавления службы базы данных	513
11.6.1. Подготовка базы данных	514
11.6.2. Создание таблиц	517
11.7. Командный интерпретатор для приложений Python	518
11.7.1. Использование командного интерпретатора Python в Django	519
11.7.2. Эксперименты с моделью данных	520
11.8. Приложение администрирования Django	522
11.8.1. Настройка приложения admin	522
11.8.2. Проверка работы с приложением admin	523
11.9. Создание пользовательского интерфейса для блога	531
11.9.1. Создание шаблона	532
11.9.2. Создание шаблона URL	533
11.9.3. Создание функции представления	537
11.10. Усовершенствование вывода	540
11.10.1. Изменение запроса	541
11.11. Работа с данными, введенными пользователем	545
11.11.1. Шаблон. Добавление формы HTML	546
11.11.2. Добавление записи URLconf	546
11.11.3. Представление. Обработка данных, введенных пользователем	547
11.11.4. Проверка возможности передачи данных с одного сайта на другой	548
11.12. Простые и модельные формы	550
11.12.1. Вводные сведения о формах Django	550
11.12.2. Вариант с применением форм модели	551
11.12.3. Использование объекта класса ModelForm для создания формы HTML	552
11.12.4. Обработка данных класса ModelForm	553
11.13. Дополнительные сведения о представлениях	554
11.13.1. Полууниверсальные представления	555
11.14. *Усовершенствования внешнего интерфейса	557
11.15. *Проверка компонентов	558
11.15.1. Описание кода приложения блога	560
11.15.2. Общие сведения о приложении блога	566
11.16. *Промежуточное приложение Django: TweetApprover	567
11.16.1. Создание структуры файла проекта	568

11.16.2. Установка библиотеки Twython	573
11.16.3. Структура URL	575
11.16.4. Модель данных	579
11.16.5. Передача новых твитов для рецензирования	584
11.16.6. Проверка твитов	589
11.17. Ресурсы	598
11.18. Заключение	599
11.19. Упражнения	599
Глава 12. Облачные вычисления: Google App Engine	605
12.1. Введение	606
12.2. Что такое облачные вычисления	606
12.2.1. Уровни службы облачных вычислений	608
12.2.2. Что такое App Engine	610
12.3. “Песочница” и набор SDK App Engine	613
12.3.1. Службы и API	615
12.4. Выбор платформы для системы App Engine	617
12.4.1. Платформы: webapp, затем Django	619
12.5. Поддержка версии Python 2.7	625
12.5.1. Различия общего характера	626
12.5.2. Различия в коде	626
12.6. Сравнение с платформой Django	627
12.6.1. Запуск приложения Hello World	627
12.6.2. Создание приложения Hello World вручную	628
12.7. Преобразование приложения Hello World в простой блог	629
12.7.1. Быстрый просмотр изменений: преобразование простого текста в HTML за 30 секунд	630
12.7.2. Добавление формы	631
12.7.3. Добавление службы Datastore	633
12.7.4. Постепенные усовершенствования	637
12.7.5. Консоль Разработка/SKD	638
12.8. Добавление службы Memcache	644
12.9. Статические файлы	648
12.10. Добавление службы Users	648
12.10.1. Идентификация с помощью службы Google Accounts	649
12.10.2. Объединенная идентификация	650
12.11. Оболочка удаленного API	650
12.11.1. Функция Datastore Admin	651
12.12. Быстрый обзор (с использованием кода на языке Python)	652
12.12.1. Отправка электронной почты	653
12.12.2. Получение электронной почты	654
12.13. Мгновенная отправка сообщений с помощью службы XMPP	656
12.13.1. Прием при мгновенном обмене сообщениями	657
12.14. Обработка изображений	658
12.15. Очереди задач (незапланированные задачи)	658
12.15.1. Создание задач	659
12.16. Профилирование с помощью Appstats	666
12.16.1. Добавление стандартной программы обработки в файл app.yaml	666
12.16.2. Добавление специализированной страницы Admin Console	667
12.16.3. Инициализация этого интерфейса как встроенной функции	667
12.17. Служба URLfetch	667
12.18. Быстрый обзор (без использования кода Python)	668
12.18.1. Служба Cron (планируемые задачи/задания)	668
12.18.2. Запросы разогрева	669
12.18.3. Защита от хакерских атак типа “отказ в обслуживании”	670
12.19. Обеспечение замкнутости поставщика	670

12.20. Ресурсы	671
12.21. Резюме	673
12.22. Упражнения	674
Глава 13. Веб-службы	679
13.1. Введение	680
13.2. Сервер биржевых котировок Yahoo! Finance	680
13.3. Создание микроблогов в сети Twitter	684
13.3.1. Социальные сети	684
13.3.2. Сеть Twitter и язык Python	685
13.3.3. Пример API с более длинной комбинацией	687
13.3.4. Резюме	699
13.3.5. Дополнительные интернет-ресурсы	699
13.4. Упражнения	699
Часть III. Дополнительная и экспериментальная	705
Глава 14. Обработка текста	707
14.1. Значения, разделяемые запятыми	708
14.1.1. Введение в значения, разделяемые запятыми	708
14.1.2. Повторение примера с портфелем акций	710
14.2. Нотация объектов JavaScript	712
14.3. Расширяемый язык разметки гипертекста	716
14.3.1. Введение в язык XML	716
14.3.2. Языки Python и XML	717
14.3.3. Применение формата XML на практике	721
14.3.4. *Клиент-серверные службы с использованием протокола XML-RPC	725
14.4. Литература	729
14.4.1. Дополнительные ресурсы	729
14.5. Модули, связанные с обработкой текстов	730
14.6. Упражнения	731
Глава 15. Разное	735
15.1. Интерпретатор Jython	736
15.1.1. Введение в интерпретатор Jython	736
15.1.2. Пример графического пользовательского интерфейса с использованием библиотеки Swing	737
15.2. Служба Google+	740
15.2.1. Введение в платформу Google+	740
15.2.2. Среда Python и интерфейс API Google+	741
15.2.3. Простой инструмент анализа социальных средств коммуникации	741
15.3. Упражнения	750
Приложение А. Ответы на некоторые упражнения	755
Приложение Б. Справочные таблицы	759
Приложение В. Версия Python 3: эволюция языка программирования	777
В.1. Почему изменяется язык Python	778
В.2. Что изменилось	778
В.2.1. Оператор print становится функцией print()	778
В.2.2. Строки: Юникод по умолчанию	779
В.2.3. Единственный тип класса	780
В.2.4. Обновленный синтаксис для исключений	780
В.2.5. Обновления, касающиеся целых чисел	781
В.2.6. Повсеместное использование итераторов	782

В.3. Инструменты миграции	783
В.3.1. Инструмент 2to3	783
В.3.1. Версия Python 2.6+	783
В.4. Выводы	784
В.5. Список Литературы	784
Приложение Г. Переход к версии Python 3 на основе выпуска Python 2.6+	785
Г.1. Язык Python 3: следующее поколение	785
Г.1.1. Гибридная версия 2.6+ как инструмент перехода	786
Г.2. Целые числа	787
Г.2.1. Единый целочисленный тип	787
Г.2.2. Новые двоичные и модифицированные восьмеричные литералы	787
Г.2.3. Классическое или истинное деление	788
Г.3. Встроенные функции	790
Г.3.1. Оператор print или функция print()	790
Г.3.2. Функция reduce() перенесена в модуль functools	790
Г.3.3. Прочие обновления	791
Г.4. Объектно-ориентированное программирование: два разных объекта классов	791
Г.5. Строки	792
Г.5.1. Литералы bytes	793
Г.6. Исключения	793
Г.6.1. Обработка исключений (использование оператора as)	793
Г.6.2. Генерация исключений	794
Г.7. Другие инструменты перехода и рекомендации	794
Г.8. Написание кода, совместимого как с версией 2.x, так и с версией 3.x	795
Г.8.1. Операция print или функция print()?	796
Г.8.2. Импортируйте в решение свой способ	797
Г.8.3. Составление полной картины из отдельных фрагментов	798
Г.9. Резюме	799
Предметный указатель	801

“Упрощенное, но в то же время глубокое детальное изложение, всестороннее освещение материала и информативные исторические ссылки делают эту книгу идеально подходящей для преподавания... Легкое чтение со сложными примерами, представленными достаточно просто, и обширные исторические ссылки, которые редко удается найти в таких книгах. Удивительно!”

Глория У. (Gloria W.)
Из отзыва на предыдущее издание

“Долгожданное второе издание книги Core Python Programming оправдало ожидания — его глубокое и широкое изложение и полезные упражнения помогут читателям учиться и получить опыт программирования на языке Python”.

Алекс Мартелли (Alex Martelli),
автор книги Python in a Nutshell,
редактор книги Python Cookbook

“Книга Core Python Programming вызвала большой шум. Оказывается, этот шум вполне оправдан. Я думаю, что на данный момент это лучшая книга для изучения языка Python. Я бы рекомендовал книгу Чана в дополнение к книгам Learning Python (O'Reilly), Programming Python (O'Reilly) или The Quick Python Book (Manning)”.

Дэвид Мерц (David Mertz),
Ph.D., IBM DeveloperWorks

“В прошлом году я интенсивно изучал язык Python и видел много положительных обзоров Вашей книги. Они подтвердили мнение, что книга Core Python Programming теперь считается стандартным вводным учебником”.

Ричард Озаки (Richard Ozaki),
Lockheed Martin

“В заключение отмечу, что книга является одним из лучших учебников и справочников по языку Python среди существующих изданий”.

Майкл Бакстер,
Linux Journal

“Очень хорошо написано. Это самая четкая, самая дружелюбная книга о языке Python, описывающая его в широком контексте. Она не требует от читателя большой подготовки. Некоторые важные темы языка Python излагаются тщательно и подробно. В отличие от авторов многих книг, предназначенных для новичков, Чан никогда не снижает уровень изложения или не мучает читателя детскими играми. Он твердо придерживается последовательного изложения синтаксиса языка Python и его структуры”.

“Если бы я мог выбрать только одну книгу по языку Python, то остановился бы на книге Core Python Programming. В ней затрагиваются более глубокие темы по сравнению с книгой Learning Python, но при этом она охватывает все основные вопросы. Если Вы хотите купить единственную книгу о языке Python, я рекомендую эту. Вам понравится ее содержание и иронический стиль. Заодно вы научитесь программировать на языке Python. Эта книга облегчит Вашу ежедневную работу. Хорошо сделано, г-н Чан!”

Рон Стивенс (Ron Stephens),
Python Learning Foundation

“Я думаю, что наилучшим языком программирования для новичков несомненно является Python. Моя любимая книга — Core Python Programming”.

s003ap,
Форумы MP3Car.com

“Лично мне действительно нравится язык Python, понятный, абсолютно интуитивный, удивительно гибкий и довольно выразительный. Язык Python совсем недавно стал завоевывать свое место в мире Windows, но уже получил широкое признание. Для изучения язык Python я начал бы с книги Core Python Programming”.

Билл Босуэлл (Bill Boswell),
MCSE, Microsoft Certified Professional Magazine Online

“Если Вы легко обучаетесь по книгам, я рекомендую Core Python Programming. Это лучшее, что я видел. Начиная с нуля, я за три месяца научился писать работоспособные проекты на языке Python (автоматизация MSOffice, SQL DB и т.д.)”.

ptonman,
Dev Shed Forums

“Python — просто превосходный язык. Его легко изучить, он работает на нескольких платформах и очень удобен. Он обеспечивает достижение многих технических целей, для которых создавался язык Java. Одним словом, все остальные языки являются результатом эволюции, и только язык Python был спроектирован, причем хорошо. О языке Python написано множество книг. Core Python Programming — лучшая из них”.

Крис Тиммонс (Chris Timmons),
C. R. Timmons Consulting

“Если Вам нравится серия Core издательства Prentice Hall, обратите внимание на книгу Core Python Programming. В ней изложены тщательно продуманные конкретные детали многих практических тем, которые слабо освещены в других книгах”.

Митчелл Л. Модел (Mitchell L. Model),
MLM Consulting

ПРЕДИСЛОВИЕ

Третье издание книги *Python: создание приложений!*

Мы рады, что вы обратились к нам за помощью в максимально быстром и глубоком изучении языка Python. Цель серии *Core Python* не сводится к простому преподаванию языка Python; мы хотим, чтобы вы получили такой уровень знаний, чтобы разрабатывать программное обеспечение в любой прикладной области.

В других книгах серии *Core Python* — *Core Python Programming* и *Core Python Language Fundamentals* — мы не только описываем синтаксис этого языка, но и стремимся всесторонне изложить его структуру. Мы полагаем, что вооружившись этими знаниями, вы напишете более эффективные приложения, независимо от уровня вашей подготовки.

После прочтения любой другой вводной книги о языке Python может показаться, что вы изучили его достаточно хорошо. Выполняя многочисленные упражнения, вы, вероятно, даже вполне уверены в своих навыках программирования на языке Python. Однако у вас могут возникнуть вопросы: “И что теперь? Какие виды приложений я могу создать с помощью языка Python?” Если вы изучали язык Python для создания узкоспециализированного проекта, то можете спросить: “Что еще я могу разработать с помощью языка Python?”

О книге

В книге *Python: создание приложений* вы узнаете обо всем, что следует знать о языке Python, и получите новые навыки, позволяющие создавать разнообразные приложения.

Эти главы повышенной сложности предназначены для “быстрого погружения” в разнообразные темы. Если вы углубитесь в определенные области разработки приложений, охваченные в какой-либо из этих глав, то, вероятно, обнаружите, что они содержат более чем достаточно информации, чтобы направить вас в правильном

направлении. *Не ожидайте* всестороннего изложения, потому что это умалило бы универсальность данной книги.

Как и во всех других книгах серии *Core Python*, в этой книге приведено много примеров, которые можно проверить на вашем компьютере. Для того чтобы закрепить усвоенные понятия, в конце каждой главы приводятся как простые, так и сложные упражнения. Они предназначены для проверки ваших знаний и навыков программирования на языке Python. Практический опыт ничем невозможно заменить. Мы полагаем, что вы должны не только получить навыки программирования на языке Python, но и усвоить их за максимально короткий период времени.

Поскольку лучший способ получить навыки — это практика, упражнения представляют собой одно из самых больших преимуществ этой книги. С их помощью вы можете проверить свои знания, полученные из глав, а также получить опыт программирования. Для закрепления навыков нет ничего эффективнее, чем разработка приложений. Вам придется решать легкие, средние и трудные проблемы. По просьбе читателей мы включили в книгу задачи, подразумевающие необходимость писать большие, а не игрушечные и практически бесполезные программы. Ответы на некоторые упражнения приведены в приложении А, а справочные таблицы — в приложении Б.

Я хотел бы поблагодарить всех читателей за советы и предложения. Именно благодаря вам я стал писать книги. Я прошу вас писать мне письма и помочь подготовить *четвертое* издание еще лучше, чем все предыдущие!

Для кого предназначена эта книга

Для всех, кто знает о существовании языка Python и хочет знать больше, развивая свои навыки программирования приложений. Язык Python применяется во многих областях, включая промышленность, информационные технологии, науку, бизнес, индустрию развлечений и др. Это значит, что список пользователей языка Python (и читателей этой книги) включает следующие профессии (но не ограничивается ими):

- разработчики программного обеспечения;
- разработчики систем автоматизированного проектирования;
- разработчики систем контроля качества и средств автоматизации;
- разработчики информационных систем и сетевые администраторы;
- ученые и математики;
- проектировщики и менеджеры, управляющие проектами;
- разработчики мультимедийных и аудиовизуальных систем;
- менеджеры, управляющие логистическими цепочками, и выпускающие менеджеры;
- веб-мастера и штат управления контентом;
- инженеры технической поддержки;
- разработчики и администраторы баз данных;
- инженеры, принимающие участие в научно-исследовательских проектах;
- инженеры, которые занимаются интеграцией и обслуживанием программного обеспечения;

- университетские преподаватели;
- разработчики веб-служб;
- разработчики финансового программного обеспечения;
- и многие другие!

Список знаменитых компаний, использующих язык Python, включает: Google, Yahoo!, NASA, Lucasfilm/Industrial Light and Magic, Red Hat, Zope, Disney, Pixar и Dreamworks.

Автор и язык Python

Я открыл для себя язык Python примерно десять лет назад, работая в компании Four11. В то время основным продуктом этой компании была служба каталогов Four11.com White Page. Язык Python использовался для разработки следующего продукта: веб-службы электронной почты Rocketmail, которая в итоге эволюционировала в службу Yahoo! Mail.

Изучать язык Python и разрабатывать службу Yahoo! Mail было интересно. Я помог перепроектировать адресную книгу и механизм проверки правописания. В то время язык Python стал частью многих других сайтов Yahoo!, включая службы People Search, Yellow Pages, а также Maps and Driving Directions. Фактически я был ведущим разработчиком службы People Search. Хотя в то время я плохо знал язык Python, выучить его было очень легко — он много проще других языков, которые я осваивал прежде. Из-за дефицита учебников, существовавшего в то время, в качестве основных источников знаний мне пришлось использовать справочники *Library Reference* и *Quick Reference Guide*; это и стало главной мотивацией для написания книги, которую вы читаете сейчас. Со времени моей работы в компании Yahoo! я мог использовать язык Python для решения любых интересных задач. В каждом из этих случаев я смог использовать мощь языка Python для своевременного решения поставленной задачи. Я также разработал несколько курсов по языку Python и использовал эту книгу для преподавания. Книги серии *Core Python* превосходно помогают не только *учиться*, но и *учить*. Как инженер я знаю, как сложно изучить, освоить и применить новую технологию. Как профессиональный преподаватель я также знаю то, что необходимо для *эффективного обучения учеников*. Эти книги дают опыт, необходимый для выявления реальных аналогий и подсказок, которые невозможно получить у “простого учителя” или «обычного книжного автора”.

Особенности стиля изложения: технический, но простой

Мой опыт показывает, что для успешного и быстрого освоения языка Python нужна не книга для новичков или справочник по основам информатики, а книга, ориентированная на технические подробности. Для того чтобы ускорить процесс обучения, мы будем вводить теоретические понятия, иллюстрируя их соответствующими примерами. В конце каждой главы вы найдете многочисленные упражнения, закрепляющие некоторые изложенные понятия и идеи.

Мы не стремимся конкурировать со стилем Брюса Эккеля (Bruce Eckel) (см. рецензии на первое издание на веб-сайте <http://corepython.com>). Это не сухой университетский учебник. Наша цель состоит в том, чтобы разговаривать с вами, как будто вы сидите в аудитории. Как вечный студент я постоянно ставлю себя на место своих учеников и говорю им то, что им следует слышать, чтобы изучить понятия настолько быстро и полно, насколько это возможно. Чтение этой книги вам покажется быстрым и легким, но при этом вы не упустите из виду технические детали.

Как инженер я знаю то, что должен сказать вам, чтобы объяснить понятие из языка Python. Как учитель могу взять технические детали и перевести их на язык, который легко понять и моментально усвоить. Вы можете извлечь из этого то, что принесет вам наибольшую пользу, и при этом вы еще больше полюбите программировать на языке Python.

Как вы могли заметить, несмотря на то, что я — единственный автор книги, я использую третье множественное число, т.е. говорю “мы” и “наш”, потому что мы вместе движемся к общей цели — освоению языка Python.

О третьем издании

В момент выхода первого издания книги язык Python вступал в свою вторую эру вместе с выпуском версии 2.0. С тех пор язык подвергся существенным улучшениям, которые способствовали полному дальнейшему успеху, принятию и росту его популярности. В нем были устранены недостатки и добавлены новые функциональные возможности, отражающие новый уровень его мощи и богатый опыт многочисленных разработчиков.

Второе издание книги вышло в 2006 году, в разгар господства языка Python, одновременно с появлением его самой популярной версии 2.5. Второе издание вызвало восторженные отзывы и превзошло по продажам первое. Язык Python получил многочисленные награды, в частности:

- Tiobe (www.tiobe.com)
 - Язык года (2007, 2010)
- LinuxJournal (linuxjournal.com)
 - Лучший язык программирования (2009–2011)
 - Лучший язык описания сценариев (2006–2008, 2010, 2011)
- Премии членов сообщества LinuxQuestions.org
 - Язык года (2007–2010)

Эти награды еще больше способствовали популярности языка Python.

В настоящее время появилось третье поколение — версия Python 3. Книга *Core Python Programming* тоже перешла на третий этап, и я благодарен издательству Prentice Hall за предложение написать ее третье издание.

Поскольку версия 3.x не имеет обратной совместимости с версиями Python 1 и 2, для ее полной адаптации в промышленности потребуется определенное время. Мы рады помочь вам совершить этот переход. Программы в этом издании будут представлены как в версии Python 2, так и в версии Python 3 (только по возможности, поскольку пока не все можно перенести из одной версии в другую). Мы также обсудим различные инструменты и методы, необходимые для перехода.

Изменения, внесенные в версии 3.x, продолжают тенденцию к улучшению языка и позволяют сделать большой шаг к удалению некоторых его последних недостатков, а также большой скачок в развитии языка. Структура книги также значительно изменилась. Из-за своего размера и объема книга Core Python Programming требовала пересмотра для выпуска третьего издания.

По этой причине издательство Prentice Hall и я решили, что лучше всего оставить логическую структуру частей I и II предыдущих изданий, в которых представлено ядро языка и вопросы, связанные с разработкой сложных приложений соответственно, и разделить книгу на два тома.

Вы держите в руках вторую часть третьего издания книги Core Python Programming. Хорошая новость заключается в том, что для понимания второй части первая часть не требуется. Мы лишь хотели бы, чтобы читатели имели умеренный опыт по программированию на языке Python. Если вы лишь недавно изучили язык Python и стали программировать на нем или уже имеете опыт и хотите перейти на новый уровень, то вы нашли то, что нужно!

Читатели предыдущих изданий книги Core Python Programming уже знают, что мы подробно излагаем основные вопросы, не ограничиваясь синтаксисом (разве для изучения синтаксиса нужна книга?). Зная, как устроен язык Python, включая отношение между объектами данных и механизмом управлением памятью, можно стать более эффективным программистом. Эти вопросы остались в первой части и стали основой книги Core Python Language Fundamentals.

Как и прежде, я продолжаю вести веб-сайт и блог, а также писать статьи, чтобы информация оставалась как можно более свежей независимо от версии языка Python.

В новое издание добавлены следующие темы.

- Примеры реализации электронной почты с помощью веб (глава 3).
- Использование библиотеки Tile/Ttk (глава 5).
- Использование базы данных MongoDB (глава 6).
- Более подробные примеры использования программ Outlook и PowerPoint (глава 7).
- Стандарт WSGI (глава 10).
- Использование социальной сети Twitter (глава 13).
- Использование службы Google+ (глава 15).

Кроме того, мы рады предложить три совершенно новые главы: главу 11, “Веб-платформы: Django”; главу 12, “Облачные вычисления: Google App Engine”, и главу 14, “Обработка текста”. В них описаны новые или существующие области применения языка Python. Все существующие главы были обновлены с учетом последних версий языка Python, включая новый материал. В начале каждой главы приводится список тем, которые в ней рассматриваются.

Путеводитель по главам

Книга разделена на три части. Первая часть, занимающая почти две трети всего объема, посвящена основным инструментам, входящим в любой пакет для разработки приложений (с упором на язык Python, разумеется).

Во второй части излагаются разнообразные темы, связанные с веб-программированием. Завершается книга экспериментальными главами, которые касаются совершенно новых вопросов и будут уточняться в следующих изданиях. Все три части охватывают одну тему — создание приложений на языке Python. Мы рады помочь читателям разобраться во многих ключевых аспектах программирования на языке Python.

Ниже приводится подробное описание глав.

Часть I. Общие прикладные темы

Глава 1. Регулярные выражения

Регулярные выражения — мощный инструмент, позволяющий выявлять шаблоны, извлекать их, а также выполнять операции поиска и замены.

Глава 2. Сетевое программирование

В настоящее время многие приложения связаны с работой в сетях. В этой главы мы научим вас создавать клиентов и серверы с помощью протоколов TCP/IP и UDP/IP, а также расскажем, что такое SocketServer и Twisted.

Глава 3. Программирование интернет-клиентов

Большинство интернет-протоколов, используемых в настоящее время, разработаны с помощью сокетов. В главе 3 мы исследуем высокоуровневые библиотеки, используемые для создания клиентов этих интернет-протоколов. В частности, мы изучим протокол передачи файлов (FTP), протокол передачи новостей Usenet (NNTP) и множество протоколов электронной почты (SMTP, POP3, IMAP4).

Глава 4. Многопоточное программирование

Многопоточное программирование — один из способов улучшить производительность приложения с помощью параллельной работы. Эта глава посвящена вопросам реализации потоков на языке Python. В ней объясняются основные понятия и показывается, как правильно создать многопоточное приложение на языке Python и какие сценария использования являются лучшими.

Глава 5. Программирование графического пользовательского интерфейса

Набор графических инструментов Tkinter (в версии Python 3 переименованный в tkinter), основанный на библиотеке Tk, по умолчанию является основной библиотекой для разработки графического пользовательского инструмента на языке Python. С его помощью мы покажем, как создать простые приложения с графическим пользовательским интерфейсом. Один из лучших способов обучения — копирование. Используя эти приложения, вы быстро освоите эту тему. В главе кратко рассмотрены и другие графические библиотеки, такие как Tix, Pmw, wxPython, PyGTK и Ttk/Tile.

Глава 6. Программирование баз данных

Язык Python позволяет упростить программирование баз данных. Сначала мы рассмотрим основные понятия, а затем изучим интерфейс программирования приложений, управляющих базами данных (DB-API). После этого будет показано, как установить соединение с реальной базой данных и выполнить запросы и другие операции с помощью языка Python. Если Вы предпочитаете автоматический подход, основанный на языке Structured Query Language (SQL), и просто хотите работать с объектами, не интересуясь устройством базы данных, то можете использовать объектно-реляционные преобразователи. В заключение мы вводим читателей в мир нереляционных баз данных, экспериментируя с базой данных MongoDB, играющей роль примера технологии NoSQL.

Глава 7. Программирование приложений для работы с Microsoft Office

Нравится Вам это или нет, но мы живем в мире, в котором нам приходится вступать в контакт с персональными компьютерами, работающими под управлением операционной системы Microsoft Windows. Независимо от того, насколько часто мы это делаем, язык Python может облегчить нашу жизнь. В этой главе исследуется программирование клиентов COM с помощью языка Python для управления и взаимодействия с приложениями Office, такими как Word, Excel, PowerPoint и Outlook. В предыдущих изданиях эта глава была экспериментальной, а теперь она расширена и стала самостоятельной.

Глава 8. Создание расширений для языка Python

Ранее мы уже подчеркивали, насколько мощными являются повторное использование кода и расширения языка. В языке Python этими расширениями являются модули и пакеты. Однако помимо этого существует возможность разработки низкоуровневых программ на языках C/C++, C# и Java. Эти расширения легко интегрируются в язык Python. Создание расширений на низкоуровневых языках позволяет повысить производительность приложений и усилить безопасность (поскольку исходный код раскрывать необязательно). В этой главе шаг за шагом рассматривается процесс расширения языка Python с помощью языка C.

Часть II. Разработка веб-приложений

Глава 9. Веб-клиенты и веб-серверы

Продолжая обсуждение клиент-серверной архитектуры, начатое в главе 2, мы применяем эту концепцию к веб. В этой главе мы изучим разнообразные инструменты веб-клиентов, методы анализа веб-контента и, наконец, покажем, как написать свой веб-сервер на языке Python.

Глава 10. Веб-программирование: интерфейсы CGI и WSGI

Главная работа веб-сервера состоит в том, чтобы получать запросы клиента и возвращать результаты. Но как серверы получают эти данные? Несмотря на то что они действительно хороши при возвращении результатов, у них обычно нет возможностей или логических механизмов, необходимых для этого; основная работа выполняется в другом месте. Интерфейс CGI позволяет серверам запускать другие программы, выполняющие эту обработку, однако он не допускает масштабирования и поэтому не используется на практике; тем не менее его концепции все еще применяются, независимо от того, какую платформу вы используете. Таким образом, большая часть главы посвящена изучению интерфейса CGI. Вы также узнаете, как интерфейс WSGI помогает разработчикам приложений, предоставляя им общий программный интерфейс. Кроме того, вы увидите, как интерфейс WSGI помогает разработчикам платформ, которые должны соединять веб-серверы на одной стороне с кодом приложения на другом, работать так, чтобы разработчики приложений могли не беспокоиться о платформе выполнения.

Глава 11. Веб-платформы: Django

Язык Python обеспечивает разработку веб-приложений с помощью платформы Django, ставшей одной из самых популярных. В этой главе будет описан этот каркас и показано, как написать простые веб-приложения. Эти знания позволят вам при необходимости приступить к изучению других веб-платформ.

Глава 12. Облачные вычисления: Google App Engine

Облачные вычисления покоряют индустрию. Миру широко известны инфраструктурные службы AWS компании Amazon и онлайн-приложения, такие как Gmail и Yahoo! Mail. Однако существуют мощные альтернативные платформы, позволяющие

не нагружать пользователей лишними знаниями об инфраструктуре и предоставляющие им больше гибкости благодаря управлению их приложениями и кодом. В этой главе содержится исчерпывающее введение в первую службу на языке Python — Google App Engine. Обладая этими знаниями, Вы сможете изучить другие аналогичные службы.

Глава 13. Веб-службы

В этой главе изучаются высокоуровневые веб-службы (основанные на протоколе HTTP). Мы опишем как старую службу (Yahoo! Finance), так и новую (Twitter). Вы научитесь работать с обеими службами, используя язык Python и знания, полученные в предыдущих главах.

Часть III. Дополнительная и экспериментальная

Глава 14. Обработка текста

Первая дополнительная глава посвящена вопросам обработки с помощью языка Python. Сначала мы исследуем формат CSV, затем JSON и, наконец, XML. В конце главы мы воспользуемся клиент-серверной архитектурой в сочетании с языком XML для создания службы вызова удаленных процедур (RPC) с помощью протокола XML-RPC.

Глава 15. Разное

Эта глава содержит дополнительный материал, который будет разработан в следующем издании. Она посвящена интерпретатору Java/Jython и службе Google+.

Соглашения

Все результаты работы программ и исходные коды выделены с помощью моноширинного шрифта. Ключевые слова языка Python выделены **полужирным моноширинным** шрифтом. Строки вывода начинаются с трех символов “больше” (>>>), обозначающих приглашение интерпретатора Python. Звездочка (*) перед названием главы, раздела или упражнения означает сложный и/или факультативный материал.



Примечание



Модуль



Подсказка

2.5

Номер версии, в которой появились новые возможности языка Python

Сайты, посвященные книге

Мы приветствуем любые замечания — хорошие, плохие и ужасные. Если у вас есть комментарии, предложения, комплименты, жалобы, указания на ошибки, вопросы и что-нибудь еще, пишите по адресу corepython@yahoo.com.

На веб-сайте <http://corepython.com> вы найдете список замеченных ошибок, исходные коды, обновления, анонсы, упражнения, файлы для загрузки и другую информацию о книге. Вы можете принять участие в обсуждении книг серии *Core Python* на странице службы Google+ по адресу <http://plus.ly/corepython>.

Благодарности рецензентам и соавторам третьего издания

Глория Вилладсен (Gloria Willadsen) (главный рецензент)

Мартин Омандер (Martin Omander) (рецензент и соавтор главы 11, “Веб-платформы: Django,” создатель приложения TweetApprover и соавтор раздела 15.2 “Служба Google+” в главе 15, “Разное”).

Дарлин Вонг (Darlene Wong)

Брайм Вердьер (Bryce Verdier)

Эрик Вальстад (Eric Walstad)

Поль Биссекс (Paul Bissex) (соавтор книги Python Web Development with Django)

Йохан “proprry” Ефросин (Johan “proprry” Euphrosine)¹

Энтони Валлон (Anthony Vallone)

Вдохновители

Моя жена Фей (Faye), которая продолжает поражать меня способностью управлять домашним хозяйством, заботиться о детях и их графике, кормить нас всех, вести финансы и в состоянии сделать это, в то время как я витаю в облаках или пишу книги.

Редакционная коллегия

Марк Тауб (Mark Taub) (главный редактор)

Дебра Уильямс Кэли (Debra Williams Cauley) (рецензент издательства)

Джон Фуллер (John Fuller) (ответственный редактор)

Элизабет Райан (Elizabeth Ryan) (редактор проекта)

Боб Рассел, Octal Publishing, Inc. (литературный редактор)

Дайан Рассел, Octal Publishing, Inc. (технический редактор)

Благодарности ко второму изданию

Рецензенты и соавторы

Шаннон -jj Беренс (Shannon -jj Behrens) (главный рецензент)

Майкл Сантом (Michael Santos) (главный рецензент)

Рик Кван (Rick Kwan)

Линделл Алдерманн (Lindell Aldermann) (один из авторов раздела о Юникоде в главе 6)

Вей-Йип Тун (Wai-Yip Tung) (один из авторов примера о Юникоде в главе 20)

Эрик Фостер-Джонсон (Eric Foster-Johnson) (соавтор по книге Beginning Python)

Алекс Мартелли (Alex Martelli) (редактор книги Python Cookbook и автор книги Python in a Nutshell)

Ларри Розенштейн (Larry Rosenstein)

Джим Орош (Jim Orosz)

Кришна Сринивасан (Krishna Srinivasan)

Чак Кунг (Chuck Kung)

¹ Proprry — псевдоним Йохана Ефросина в социальных сетях.

Вдохновители

Мои прекрасные дети и хомяк.

Благодарности к первому изданию

Рецензенты и соавторы

Гвидо ван Россум (Guido van Rossum) (создатель языка Python)
Доусон Тонг (Dowson Tong)
Джеймс С. Алстром (James C. Ahlstrom) (соавтор по книге Internet Programming with Python)
С. Канделария де Пам (S. Candelaria de Ram)
Кэй С. Хорстманн (Cay S. Horstmann) (соавтор по книге Core Java и Core JavaServer Faces)
Майкл Сантом (Michael Santos)
Грэг Уард (Greg Ward) (создатель пакета distutils и его документации)
Винсент С. Рубино (Vincent C. Rubino)
Мартийн Фаассен (Martijn Faassen)
Эмиль ван Себил (Emile van Seville)
Реймонд Цай (Raymond Tsai)
Алберт Л. Андерс (Albert L. Anders) (один из авторов главы о проекте MT Programming)
Фредрик Лунд (Fredrik Lundh) (автор книги Python Standard Library)
Камерон Лейрд (Cameron Laird)
Фред Л. Дрейк, мл. (Fred L. Drake, Jr.) (соавтор по книге Python & XML и редактор официальной документации по языку Python)
Джереми Хилтон (Jeremy Hylton)
Стив Йошимото (Steve Yoshimoto)
Ааз Марух (Aahz Maruch) (автор по книге Python for Dummies)
Джефффри Е.Ф. Фридл (Jeffrey E. F. Friedl) (автор книги Mastering Regular Expressions)
Питер Клерхут (Pieter Claerhout)
Катриона (Кейт) Джонстон (Catriona (Kate) Johnston)
Дэвин Эшер (David Ascher) (соавтор по книге Learning Python и редактор книги Python Cookbook)
Рег Чарни (Reg Charney)
Кристиан Тисмер (Christian Tismer) (создатель версии Stackless Python)
Джейсон Стилвелл (Jason Stillwell)
и мои студенты из филиала Калифорнийского университета в Санта-Крузе (UC Santa Cruz Extension)

Вдохновители

Я хотел бы выразить глубокую благодарность Джеймсу П. Прайору (James P. Prior), моему университетскому преподавателю по программированию.

Огромное спасибо Луизе Мозер (Louise Moser) и Майклу Меллиар-Смиту (P. Michael Melliar-Smith) (моим научным руководителям во время работы над диссертацией в Калифорнийском университете в Санта-Барбаре).

Благодарю Алана Парсонса (Alan Parsons), Эрика Вулфсона (Eric Woolfson), Эндрю Пауэлла (Andrew Powell), Яна Бейрсона (Ian Bairnson), Стюарта Эллиотта (Stuart Elliott), Дэвида Пэйтона (David Paton) и других участников проекта, а также членов форумов Projectologist и Roadkiller (за музыку, поддержку и приятное времяпрепровождение).

Я хотел бы поблагодарить свою семью, друзей и Господа Бога, уберегшего и наставившего меня на верный путь. Я благодарен всем, кто верил в меня на протяжении последних двадцати лет (вы знаете, кто я такой!), — я ничего не смог бы сделать без вас.

И в заключение хотел бы поблагодарить вас, мои читатели, и сообщество любителей языка Python. Я рад научить вас программированию на языке Python и надеюсь, что вы получите удовольствие на протяжении нашего, уже третьего, путешествия.

Уэсли Дж. Чан (Wesley J. Chun)

Силиконовая Долина, Калифорния (Silicon Valley, CA)

(Это не столько место, сколько состояние душевного здоровья.)

Октябрь 2001 г.; обновлено в июле 2006 г.,

марте 2009 г. и марте 2012 г.

ОБ АВТОРЕ

Уэсли Чан вошел в мир вычислений во время учебы в школе, используя языки BASIC и ассемблер 6502 на компьютерах Commodore. Затем настала пора Паскаля на компьютере Apple II и Фортрана на перфокартах. Эти языки сделали его осторожным и внимательным разработчиком, потому что отсылка колоды перфокарт на большую ЭВМ и получение результатов занимали одну неделю. Уэсли также помог классу журналистики перейти от пишущих машинок на компьютеры Osborne 1 CP/M. Еще будучи школьником, он получил первую работу преподавателя языка BASIC для студентов четвертого, пятого и шестого уровня, а также их родителей.

После окончания школы Уэсли поступил в Калифорнийский университет в Беркли (University of California at Berkeley) и закончил его с оценками АВ по прикладной математике (компьютерные науки) и ля-минор по музыке (классическое пианино). Участь в Калифорнии, он программировал на Паскале, Logo и С. Одна из его летних интернатур была посвящена программированию на 4GL и созданию справочника для начинающего пользователя. Через несколько лет он продолжил свои исследования в Калифорнийском университете в Санта-Барбаре, получив степень магистра по информатике (распределенные системы). В это время он также преподавал программирование на языке С. Статья, основанная на его магистерской диссертации, была представлена на приз лучшей на 29-й конференции HICSS, а ее последующая версия была опубликована в Journal of High Performance Computing, издаваемом университетом Сингапура (University of Singapore).

Уэсли работал в индустрии программного обеспечения, продолжая преподавать и писать, издавать книги и представляя на конференции сотни докладов и лекций, включая курсы по языку Python как публичные, так и корпоративные. Уэсли получил опыт работы с языком Python, проектируя механизм проверки правописания и адресную книгу для службы Yahoo! Mail в версии 1.4. Он тогда стал ведущим инженером по разработке службы Yahoo! People Search. Покинув проект Yahoo!, он написал первое издание этой книги и затем совершил кругосветное путешествие. Потом он использовал язык Python в самых разных сферах — от поиска местных товаров,

механизма защиты от спама, антивирусных почтовых приложений, а также игр и приложений для социальной сети Facebook до программного обеспечения для анализа перелома позвоночника.

В свободное время Уэсли наслаждается игрой на фортепьяно, боулингом, баскетболом, велосипедным спортом, фризби, покером, путешествиями и семьей. Он добровольно участвует в работе групп пользователей языка Python, списка рассылки Tutor и конференций PyCon.

Кроме того, он поддерживает проект Alan Parsons Project Monster Discography. Если вы считаете себя фанатом рок-группы Alan Parsons, но у вас нет альбома "Freudiana", вы обязательно должны найти его! Пока Уэсли писал книгу, он работал адвокатом разработчиков (Developed Advocate) в компании Google, представляя его облачные продукты. Уэсли живет в Силиконовой Долине, и вы можете найти его по адресу @wespy или plus.ly/wespy.

ЧАСТЬ

Общие прикладные темы

ГЛАВА

1

Регулярные выражения

В этой главе...

- Общее назначение
- Специальные знаки и символы
- Регулярные выражения и язык Python
- Некоторые примеры регулярных выражений
- Более сложный пример регулярного выражения

Некоторые люди, столкнувшись с проблемой при обработке данных, думают:

“Я знаю, что мне нужно использовать регулярные выражения”.

Теперь им придется решать две проблемы.

Джейми “jwz” Жавински (Zawinski), август 1997 г.

1.1. Общее назначение

В наши дни манипулирование текстом или данными представляет собой очень важную задачу. Чтобы убедиться в этом, достаточно внимательно рассмотреть, для чего в основном применяются современные компьютеры. В число выполняемых ими задач входят обработка текста, формирование веб-страниц из так называемых “заполняемых форм”, анализ потоков информации, поступающих из баз данных, обработка котировок акций, поддержка работы служб рассылки новостей. На этом данный список отнюдь не исчерпывается. Программируя компьютеры на решение задач обработки информации, мы не всегда можем буквально задать текст или данные, подлежащие обработке, поэтому возникает необходимость воспользоваться некоторым способом представления информации с помощью шаблонов, которые компьютер может распознавать и выполнять операции на их основе.

В качестве примера можно рассмотреть следующую ситуацию. В компанию, занимающуюся разработкой программного обеспечения, обратился клиент с просьбой обеспечить для него архивирование всей электронной почты, полученной за прошедший период. При этом, безусловно, желательно, чтобы вся работа по подборке и перенаправлению писем была выполнена исключительно с помощью компьютерной программы, поскольку клиент не желает, чтобы какой-то посторонний человек читал его электронную почту и обрабатывал ее вручную. И действительно, вряд ли кому-то понравится, если его переписка окажется в чужих руках, даже если задача обработки вручную будет сводиться к просмотру даты получения писем. Еще одним примером задачи обработки электронной почты, которая может быть выполнена автоматически, является поиск писем со строкой темы наподобие “ILOVEYOU”, которая указывает на то, что сообщение заражено вирусом, и удаление этих сообщений электронной почты из личного архива пользователя. Даже эти элементарные примеры наглядно показывают, что требуется определенный способ, позволяющий запрограммировать компьютеры на поиск шаблонов в тексте.

Инфраструктуру, обеспечивающую возможность решать сложные задачи сопоставления текста с шаблонами, извлечения данных, а также поиска и замены подстрок в данных, предоставляют регулярные выражения. Проще говоря, *регулярное выражение* (regular expression или кратко regex) можно определить как строку, состоящую из специальных знаков и символов, которые указывают на шаблонное повторение или с помощью нескольких символов описывают совокупность похожих строк в виде одного шаблона (рис. 1.1). Иными словами, шаблоны служат для сопоставления с целым рядом строк. Дело в том, что шаблон регулярного выражения, который сопоставляется только с одной строкой, не представляет особого интереса и не может служить эффективной программной конструкцией. Вряд ли кому-то захочется подробно перечислять все то, что можно представить одним кратким выражением.

В языке Python для поддержки регулярных выражений применяется стандартный библиотечный модуль `re`. В этом вступительном подразделе приведено краткое и сжатое введение. Поскольку это введение является немногословным, в нем будут рассматриваться лишь самые общие характеристики регулярных выражений, используемых в

повседневном программировании на языке Python. Нет сомнения в том, что читатели, приступающие к изучению этой главы, имеют разный опыт в программировании. Поэтому мы настоятельно рекомендуем прочитать всю официальную вспомогательную документацию, а также другие книги по этой интересной теме. После ознакомления даже с самыми основными сведениями о структуре текста люди начинают воспринимать то, что раньше казалось простейшими строками, совсем под другим углом зрения!



Рис. 1.1. Регулярные выражения, подобные приведенному на этом рисунке, можно использовать для распознавания допустимых идентификаторов Python. Регулярное выражение `[A-Za-z]\w+` означает, что первый символ должен быть алфавитным, т.е. должен относиться к диапазону `A-Z` или `a-z`, а за ним следует по крайней мере один (+) алфавитно-цифровой символ (`\w`). Регулярное выражение можно сравнить с фильтром. Пример, приведенный на рис. 1.1, показывает, что в этот фильтр могут поступать произвольные строки, но благополучно минуют его лишь те строки, которые соответствуют регулярному выражению. Как показано на этом рисунке, одним из примеров строк, которые не преодолели фильтр, была строка `"4xZ"`, поскольку она начинается с цифры



Различия между поиском и сопоставлением

В главе будут часто упоминаться такие термины, как поиск и сопоставление. При формальном описании применения регулярных выражений для обнаружения шаблонов в строках под термином “сопоставление” подразумевается *распознавание шаблонов (pattern-matching)*. В языке Python предусмотрены два основных способа распознавания шаблонов: *поиск*, т.е. обнаружение совпадения с шаблоном в любой части строки, и *сопоставление*,

т.е. осуществление попытки сравнить шаблон со всей строкой (от начала строки и до конца). Поиск производится с помощью функции или метода `search()`, а для сопоставления применяется функция или метод `match()`. Но, вообще говоря, когда речь идет о применении шаблонов как таковых мы всегда используем термин “сопоставление” и проводим различия между “поиском” и “сопоставлением”, рассматривая лишь конкретную проблематику выполнения операций сопоставления с шаблонами в языке Python.

1.1.1. Первое знакомство с регулярными выражениями

Как уже было сказано, регулярные выражения представляют собой строки, содержащие текст и специальные символы, которые описывают шаблон, применяемый для распознавания нескольких строк. Мы также кратко коснулись вопроса о том, что представляет собой алфавит регулярных выражений. Применительно к решению задач обработки обычного текста алфавит, используемый для формирования регулярных выражений, представляет собой множество всех прописных и строчных букв в сочетании с цифровыми знаками. В некоторых регулярных выражениях могут также применяться специализированные алфавиты; в качестве примера можно указать алфавит, состоящий только из символов "0" и "1". Множество строк, описываемых этим алфавитом, включает все двоичные строки, т.е. "0", "1", "00", "01", "10", "11", "100" и т.д.

Теперь рассмотрим примеры самых простых регулярных выражений, которые позволяют понять, что тематика регулярных выражений, которую принято считать очень сложной, иногда допускает довольно простое описание. Используя стандартный алфавит для обычного текста, мы представим некоторые простые регулярные выражения и строки, которые могут быть описаны с помощью соответствующих шаблонов. В частности, весьма простыми являются следующие регулярные выражения, которые можно рассматривать как действительно “тривиальные”. Они состоят просто из строкового шаблона, который сопоставляется лишь с одной строкой: строкой, определяемой регулярным выражением. А теперь обратимся к примерам регулярных выражений, за которыми следуют сопоставляемые с ними строки (табл. 1.1).

Таблица 1.1. Первое знакомство с регулярными выражениями

Шаблон регулярного выражения	Сопоставляемая строка
foo	foo
Python	Python
abc123	abc123

Первым шаблоном регулярного выражения в приведенной выше таблице является "foo". В этом шаблоне отсутствуют какие-либо специальные знаки, которые позволяли бы провести сопоставление с каким-либо другим знаком, кроме приведенных в шаблоне, поэтому единственной строкой, сопоставляемой с этим шаблоном, является строка "foo". То же относится к строкам "Python" и "abc123". Но все возможности регулярных выражений раскрываются лишь тогда, когда используются специальные символы для определения кодировок, сопоставления с подгруппами и повторения шаблона. Именно эти специальные знаки позволяют применять регулярные выражения для сопоставления не с одной строкой, а с целым рядом строк.

1.2. Специальные знаки и символы

В этом разделе рассматриваются наиболее широко применяемые специальные символы и знаки, называемые также *метасимволами*, при использовании которых регулярные выражения обретают свою мощь и гибкость. Наиболее распространенные из этих знаков и символов приведены в табл. 1.2.

Таблица 1.2. Знаки и специальные символы, наиболее широко применяемые в регулярных выражениях

Обозначение	Описание	Пример регулярно-го выражения
Знаки		
<code>literal</code>	Строка содержит символьный литерал <code>literal</code>	<code>foo</code>
<code>re1 re2</code>	Строка содержит регулярные выражения <code>re1</code> или <code>re2</code>	<code>fool bar</code>
<code>.</code>	Соответствует <i>любому символу</i> (кроме <code>\n</code>)	<code>b.b</code>
<code>^</code>	Соответствует <i>началу строки</i>	<code>^Dear</code>
<code>\$</code>	Соответствует <i>концу строки</i>	<code>/bin/*sh\$</code>
<code>*</code>	Предшествующее регулярное выражение встречается в строке <i>любое количество</i> (или не встречается вообще)	<code>[A-Za-z0-9]*</code>
<code>+</code>	Предшествующее регулярное выражение встречается в строке <i>не менее одного раза</i>	<code>[a-z]+\com</code>
<code>?</code>	Предшествующее регулярное выражение встречается в строке <i>только один раз или вообще в ней не встречается</i>	<code>goo?</code>
<code>{N}</code>	Предшествующее регулярное выражение встречается в строке <i>N раз</i>	<code>[0-9]{3}</code>
<code>{M,N}</code>	Предшествующее регулярное выражение встречается в строке <i>от M до N раз</i>	<code>[0-9]{5,9}</code>
<code>[...]</code>	Соответствует любому отдельному символу из <i>класса символов</i>	<code>[aeiou]</code>
<code>[...-y...]</code>	Соответствует любому отдельному символу <i>в диапазоне от x до y</i>	<code>[0-9], [A-Za-z]</code>
<code>[^...]</code>	<i>Не соответствует</i> ни одному символу из класса символов, включая любые диапазоны, если они заданы	<code>[^aeiou], [^A-Za-z0-9_]</code>
<code>(* + ? {}) ?</code>	Предусматривает применение "нежадных" версий приведенных выше знаков вхождения/повторения (<code>*</code> , <code>+</code> , <code>?</code> , <code>{}</code>)	<code>.*?[a-z]</code>
<code>(...)</code>	Соответствует регулярному выражению, заключенному в круглые скобки, и сохраняет его в памяти как <i>подгруппу</i>	<code>([0-9]{3})? , f(oo u)bar</code>
Специальные символы		
<code>\d</code>	Соответствует любой <i>десятичной цифре</i> , так же, как <code>[0-9]</code> (<code>\D</code> является обратным по отношению к <code>\d</code> : не соответствует ни одной цифре)	<code>data\d+.txt</code>
<code>\w</code>	Соответствует любому <i>алфавитно-цифровому символу</i> , эквивалентно <code>[A-Za-z0-9_]</code> (<code>\W</code> является обратным по отношению к <code>\w</code>)	<code>[A-Za-z_]\w+</code>
<code>\s</code>	Соответствует любому <i>пробельному символу</i> , эквивалентно <code>[\n\t\r\v\f]</code> (<code>\S</code> является обратным по отношению к <code>\s</code>)	<code>of\s the</code>
<code>\b</code>	Позиция, соответствующая <i>границе слова</i> (<code>\B</code> является обратным по отношению к <code>\b</code>),	<code>\bThe\b</code>
<code>\N</code>	Соответствует <i>сохраненной подгруппе N</i> (см. (...) выше)	<code>price:\16</code>

Обозначение	Описание	Пример регулярно-го выражения
\с	Буквально соответствует <i>любому специальному символу с</i> (т.е. игнорирует особое значение этого литерала)	\., \\\, *
\A (\Z)	Позиция, соответствующая <i>началу (концу)</i> строки (см. также ^ и \$ выше)	\ADear
Расширенная система обозначений		
(?iLmsux)	Внедряет один или несколько специальных "флагов" непосредственно в регулярное выражение (применяется вместо способа, основанного на использовании функции/метода)	(?x), (?im)
(?:...)	Обозначает группу, содержимое которой <i>не сохраняется в памяти</i>	(?:\w+\.) *
(?P<name>...)	Обозначает группу, заданную именем, а не числовым идентификатором	(?P<data>)
(?P=name)	Сопоставляется с текстом, который был перед этим сгруппирован оператором (?P<name>) в той же строке	(?P=data)
(?#...)	Задаёт примечание; все содержимое примечания игнорируется	(?#comment)
(?=...)	Обеспечивает сопоставление, если далее следует шаблон ..., без сохранения в памяти сопоставленной части входной строки; эта операция именуется <i>положительной опережающей проверкой</i> (positive lookahead assertion)	(?= .com)
(?!...)	Обеспечивает сопоставление, если далее не следует шаблон ..., без сохранения в памяти сопоставленной части входной строки; эта операция именуется <i>отрицательной опережающей проверкой</i> (negative lookahead assertion)	(?! .net)
(?<=...)	Обеспечивает сопоставление, если далее находится шаблон ..., без сохранения в памяти сопоставленной части входной строки; эта операция именуется <i>положительной ретроспективной проверкой</i> (positive lookbehind assertion)	(?<=800-)
(?<!...)	Обеспечивает сопоставление, если далее не находится шаблон ..., без сохранения в памяти сопоставленной части входной строки; эта операция именуется <i>отрицательной ретроспективной проверкой</i> (positive lookbehind assertion)	(?<!192\.168\.)
(?(id/name)Y N)	Обеспечивает сопоставление регулярного выражения Y по условию, если группа с указанным идентификатором или именем существует; в противном случае возвращает N; часть N является необязательной	(?(1)y x

1.2.1. Сопоставление нескольких шаблонов регулярных выражений с помощью оператора чередования (|)

В регулярных выражениях операция чередования обозначается с помощью символа канала (|), который представлен на клавиатуре вертикальной чертой (pipeline symbol). Символ канала используется для отделения друг от друга разных регулярных выражений. В качестве примера ниже приведены некоторые шаблоны, в которых используется чередование, наряду со строками, с которыми они сопоставляются (табл. 1.3).

Таблица 1.3. Применение оператора чередования (|)

Шаблон регулярного выражения	Сопоставленные строки
at home	at, home
r2d2 c3po	r2d2, c3po
bat bet bit	bat, bet, bit

Даже один этот символ обеспечивает существенное повышение гибкости применяемых регулярных выражений, поскольку с его помощью можно обеспечить сопоставление не с одной строкой, а с несколькими. Операцию чередования (alternation), применяемую в регулярных выражениях, иногда именуют также операцией объединения (union) или логического ИЛИ (logical OR).

1.2.2. Сопоставление с любым отдельным символом (.)

Знак точки (.) обеспечивает сопоставление с любым отдельным символом, кроме \n. (В регулярных выражениях Python может использоваться так называемый флаг компиляции S или DOTALL, который позволяет отменить это правило и включить \n в число сопоставляемых символов.) Знак точки сопоставляется с любой буквой, цифрой, пробельным символом (не включая "\n"), печатаемым или непечатаемым знаком (табл. 1.4).

Таблица 1.4. Применение для сопоставления знака точки (.)

Шаблон регулярного выражения	Сопоставленные строки
f.o	В этом примере точка между "f" и "o" представляет любой символ, например, как в строках fao, f9o, f#o и т.д.
..	Любая пара символов
.end	Любой символ перед строкой end

Вопрос. Что делать, если необходимо сравнить символы с самой точкой?

Ответ. Чтобы явно указать символ точки, необходимо экранировать его функциональное назначение с помощью обратной косой черты, как в примере "\."

1.2.3. Сопоставление с началом или концом строки или с границей слова (^, \$, \b, \B)

Предусмотрены также знаки и подобные по назначению специальные символы, указывающие на то, что поиск соответствия шаблону должен осуществляться в начале или в конце строки. Для сопоставления с шаблоном, начиная с начала строки, необходимо использовать знак вставки (^) или специальный символ \A (прописная буква "A", которая следует за обратной косой чертой). Последний вариант применяется в основном на компьютерах с клавиатурой, на которой отсутствует знак вставки (такой как международная клавиатура). Аналогичным образом знак доллара (\$) или специальный символ \Z применяется для сопоставления с шаблоном, начиная с конца строки.

Шаблоны, в которых используются эти знаки, отличаются от большинства других шаблонов, рассматриваемых в этой главе, поскольку они дополнительно определяют местоположение, или позицию, где должно осуществляться сопоставление. В предыдущем примечании было отмечено, какое различие проводится между сопоставлением (попыткой найти соответствие шаблона со всей строкой, начиная с ее начала) и поиском (попыткой найти соответствие с шаблоном в другом месте строки). С учетом

сказанного рассмотрим несколько примеров применения шаблонов поиска на основе регулярных выражений, “привязанных к границе” (табл. 1.5).

Таблица 1.5. Сопоставление с началом или концом строки

Шаблон регулярного выражения	Сопоставленные строки
<code>^From</code>	Любая строка, которая начинается с подстроки <code>From</code>
<code>/bin/tcsh\$</code>	Любая строка, которая заканчивается подстрокой <code>/bin/tcsh</code>
<code>^Subject: hi\$</code>	Любая строка, полностью совпадающая со строкой <code>Subject: hi</code>

И в этом случае, если любой из этих символов (или оба эти символа) должен быть сопоставлен буквально, то необходимо использовать экранирующую обратную косую черту. Например, если бы нужно было провести сопоставление с любой строкой, которая заканчивается знаком доллара, то одним из возможных решений по применению регулярного выражения мог стать шаблон `.*\$$`.

Специальные символы `\b` и `\B` применяются для сопоставления с границами слова. Различие между этими специальными символами заключается в том, что `\b` позволяет сопоставить шаблон с границей слова, а это означает, что этот шаблон должен находиться в начале слова, без учета того, предшествуют ли этому слову какие-либо другие символы (слово расположено в середине строки) или нет (слово стоит в начале строки). Аналогичным образом, специальный символ `\B` позволяет сформировать шаблон, который обеспечивает сопоставление, только если он обнаруживается в середине слова (т.е. не на границе слова). Ниже приведены некоторые примеры (табл. 1.6).

Таблица 1.6. Сопоставление с учетом границы слова

Шаблон регулярного выражения	Сопоставленные строки
<code>the</code>	Любая строка, содержащая подстроку <code>the</code>
<code>\bthe</code>	Любое слово, которое начинается с подстроки <code>the</code>
<code>\bthe\b</code>	Сопоставляется только со словом <code>the</code>
<code>\Bthe</code>	Любая строка, которая содержит подстроку <code>the</code> , но с этой подстроки не начинается слово

1.2.4. Создание классов символов (`[]`)

Безусловно, знак точки хорошо подходит для тех случаев, когда необходимо обеспечить сопоставление с любым знаком, но иногда требуется провести сопоставление лишь с конкретным набором символов. По этой причине была предусмотрена возможность применения в шаблонах знаков квадратных скобок (`[]`). В регулярном выражении обеспечивается сопоставление с любым из символов, заключенных в квадратные скобки. Ниже приведены некоторые примеры (табл. 1.7).

Таблица 1.7. Создание классов символов (`[]`)

Шаблон регулярного выражения	Сопоставленные строки
<code>b[aeiut]</code>	<code>bat, bet, bit, but</code>
<code>[cr][23][dp][o2]</code>	Строка из четырех символов: в начале следует "c" или "r", затем — "2" или "3", после этого — "d" или "p" и, наконец, "o" или "2". Примерами могут служить подстроки <code>c2do, r3p2, r2d2, c3po</code> и т.д.

По отношению к регулярному выражению `[cr][23][dp][o2]` можно привести одно дополнительное примечание: в более строгой версии этого регулярного выражения может потребоваться, чтобы допустимыми строками были только `"r2d2"` или `"c3po"`. Но квадратные скобки просто реализуют функциональные возможности логической операции ИЛИ, поэтому с их помощью нельзя предписать выполнение такого требования. Единственным решением в этом случае становится применение символа канала, как в примере `r2d2|c3po`.

Что же касается регулярных выражений, действие которых распространяется на один символ, то знаки канала и квадратных скобок являются эквивалентными. Например, начнем с регулярного выражения `"ab"`, которое сопоставляется только со строкой, состоящей из символа `"a"`, за которым следует `"b"`. Если бы потребовалось провести сопоставление с любой из однобуквенных строк, например, с `"a"` или `"b"`, то можно было бы применить регулярное выражение `[ab]`. Но `"a"` и `"b"` — это отдельные строки, поэтому можно также выбрать регулярное выражение `a|b`. Тем не менее, если бы потребовалось провести сопоставление со строкой, применяя шаблон `"ab"`, который чередуется с `"cd"`, то использование квадратных скобок стало бы невозможным, поскольку их действие распространяется только на отдельные символы. В этом случае единственным решением становится применение шаблона `ab|cd`, по аналогии с только что описанным вариантом `r2d2/c3po`.

1.2.5. Формирование диапазонов (–) и отрицаний диапазонов (^)

Квадратные скобки позволяют задавать не только наборы из отдельных символов, но и диапазоны символов. Для обозначения диапазона символов применяется пара символов, заключенных в квадратные скобки, между которыми проставлен знак дефиса. В качестве примера можно указать диапазоны `A–Z`, `a–z` и `0–9`, применяемые для обозначения прописных букв, строчных букв и цифровых знаков соответственно. Диапазоны задаются как фрагменты лексикографического ряда символов, поэтому при их использовании можно не ограничиваться только алфавитно-цифровыми символами. Кроме того, если непосредственно за открывающейся левой скобкой следует знак вставки (^), это равносильно указанию, что не должно проводиться сопоставление ни с одним из указанных в квадратных скобках символов в данной кодировке (табл. 1.8).

Таблица 1.8. Формирование диапазонов (–) и отрицаний диапазонов (^)

Шаблон регулярного выражения	Сопоставленные строки
<code>z.[0–9]</code>	Буква <code>"z"</code> , за которой следует любой символ, за которым следует одна цифра
<code>[r–u][env–y][us]</code>	Буква <code>"r"</code> , <code>"s"</code> , <code>"t"</code> или <code>"u"</code> , за которой следует буква <code>"e"</code> , <code>"n"</code> , <code>"v"</code> , <code>"w"</code> , <code>"x"</code> или <code>"y"</code> , за которой следует буква <code>"u"</code> или <code>"s"</code>
<code>[^aeiou]</code>	Знак, отличный от гласной буквы (Упражнение. Можно ли в этой формулировке вместо "знак, отличный от гласной буквы" применить "знак согласной буквы"?)
<code>^[t\n]</code>	Знак, отличный от знака табуляции или <code>\n</code>
<code>["–a]</code>	В системе ASCII — все символы, которые расположены между <code>" "</code> и <code>"a"</code> , т.е. между знаками с порядковыми номерами 34 и 97

1.2.6. Использование операторов замыкания (*, +, ?, { }) для представления нескольких вхождений/повторений

В этом разделе рассматриваются обозначения, наиболее широко применяемые в регулярных выражениях, а именно специальные знаки *, + и ?. Эти знаки могут использоваться для сопоставления с одним, несколькими или ни с одним вхождением шаблона в строке. Оператор звездочки (*) применяется для сопоставления с любым количеством вхождений регулярного выражения, находящегося непосредственно слева от него (включая вариант, когда это выражение отсутствует). В теории языков и компиляторов операция, обозначаемая этим оператором, известна как *замыкание Клини* (Kleene). Оператор плюса (+) обеспечивает сопоставление с одним или несколькими вхождениями регулярного выражения (этот оператор именуется также оператором *положительного замыкания*), а оператор (?) позволяет провести сопоставление с регулярным выражением, которое встречается только один раз или вообще не встречается.

Применяется также оператор фигурной скобки ({ }), в котором может быть задано одно числовое значение или два числовых значения, разделенные запятыми. Этот вариант указывает, что сопоставление должно быть проведено точно с N вхождениями (в варианте { N }) или с количеством вхождений в диапазоне; например, { M, N } сопоставляется с вхождениями в количестве от M до N . Эти знаки также могут быть экранированы с использованием символа обратной косой черты; * сопоставляется со звездочкой и т.д.

Можно отметить, что в предыдущей таблице вопросительный знак используется в двух разных значениях (применение в разных значениях называется *перегрузкой*), иными словами, в одном случае он показывает сопоставление с регулярным выражением, которое встречается только один раз или не встречается ни разу, а в другом имеет иное значение: если вопросительный знак следует за любым сопоставлением, в котором используются операторы замыкания, то служит для обработчика регулярных выражений указанием, что сопоставление должно производиться с минимально возможным количеством повторений.

Что подразумевается под “минимально возможным количеством повторений”? Если сопоставление с шаблоном осуществляется с использованием операторов группирования, то обработчик регулярных выражений предпринимает попытку “охватить” или “поглотить” настолько большое количество символов путем сопоставления с шаблоном, насколько это возможно. Этот режим работы обработчика регулярных выражений принято называть *жадным* (greedy). Вопросительный знак указывает обработчику, что он должен “вовремя остановиться” и по возможности выбрать как можно меньше подлежащих сопоставлению символов, оставляя как можно больше последующих символов для согласования со следующим шаблоном (если таковой имеется). Ближе к концу этой главы будет приведен превосходный пример ситуации, в которой нельзя обойтись без отказа от жадного поведения обработчика, т.е. без перехода в нежадный режим. А пока вернемся к рассмотрению операторов замыкания (табл. 1.9).

Таблица 1.9. Использование операторов замыкания

Шаблон регулярного выражения	Сопоставленные строки
<code>[dn]ot?</code>	Буквы "d" или "n", за которыми следует буква "o", а затем, самое большее, — одна буква "t". Таким образом, это регулярное выражение сопоставляется со словами <code>do</code> , <code>no</code> , <code>dot</code> , <code>not</code>
<code>0?[1-9]</code>	Любой цифровой символ, которому может предшествовать цифра "0". В качестве примера можно указать множество числовых обозначений месяцев от января до сентября, представленных с помощью одной или двух цифр
<code>[0-9]{15,16}</code>	Пятнадцать или шестнадцать цифр (например, номера кредитных карточек)
<code></?[>]+></code>	Строки, которые сопоставляются со всеми допустимыми (и недопустимыми) тегами HTML
<code>[KQRBNP][a-h][1-8]-[a-h][1-8]</code>	Допустимый шахматный ход в алгебраической нотации (только ходы; взятия фигур, шахи и др. не представлены); т.е. строки, которые начинаются с любой из букв "K", "Q", "R", "B", "N" или "P", обозначающих шахматные фигуры, за которой следует разделенная дефисами пара обозначений клеток шахматной доски от "a1" до "h8" (и всего, что находится между ними), в которой первое обозначение клетки указывает прежнее местонахождение фигуры, а второе — новое местонахождение

1.2.7. Специальные символы, обозначающие наборы символов

Как было указано выше, в языке регулярных выражений предусмотрены специальные символы, которые могут представлять наборы символов. Например, вместо использования диапазона "0-9" можно просто задать специальный символ `\d` для указания на сопоставление с любой десятичной цифрой. Еще один специальный символ, `\w`, служит сокращением для `A-Za-z0-9_` и может использоваться для обозначения всего класса алфавитно-цифровых символов, а специальный символ `\s` обозначает все пробельные символы. Варианты этих специальных символов с прописными буквами, такие как `\D`, указывают на сопоставление с символами, не относящимися к соответствующему классу; в данном случае речь идет о сопоставлении с любым символом, отличным от десятичной цифры (то же, что и `[^0-9]`).

Используя указанные сокращения, представим несколько более сложных примеров (табл. 1.10).

Таблица 1.10. Примеры применения специальных символов

Шаблон регулярного выражения	Сопоставленные строки
<code>\w+-\d+</code>	Алфавитно-цифровая строка и число, разделенные дефисом
<code>[A-Za-z]\w*</code>	Первый символ — алфавитный; следующие символы (если они присутствуют) могут быть алфавитно-цифровыми. Это регулярное выражение почти эквивалентно выражению, которое описывает множество допустимых идентификаторов Python (см. примеры)
<code>\d{3}-\d{3}-\d{4}</code>	Номера телефонов в формате, принятом в США, с префиксом кода города, как в примере <code>800-555-1212</code>
<code>\w+@\w+\.</code>	Простые адреса электронной почты в форме <code>XXX@YYY.com</code>

1.2.8. Обозначение групп с применением круглых скобок ()

Выше было описано, как можно обеспечить сопоставление со строкой и пропуск не сопоставленных частей строки, но в некоторых случаях недостаточно просто провести сопоставление. Требуется также извлечь данные, которые были обнаружены в этой операции. Иногда возникает необходимость не только узнать, соответствует ли вся строка заданным условиям поиска, но и получить конкретные строки или подстроки, которые явились частью успешного сопоставления. Такая цель вполне может быть достигнута. Для ее достижения необходимо заключить регулярное выражение в круглые скобки.

Пара круглых скобок () в регулярном выражении позволяет решить любую из следующих задач (или обе эти задачи).

- Выполнить группирование регулярных выражений.
- Провести сопоставление с подгруппами.

В качестве одного из наглядных примеров, показывающих, для чего следует проводить группирование регулярных выражений, можно привести ситуацию, в которой со строкой сравниваются два разных регулярных выражения. Необходимость в группировании регулярного выражения возникает также в том случае, если оператор повторения требуется применить ко всему регулярному выражению, а не к отдельному символу или классу символов.

Побочным эффектом круглых скобок является то, что подстрока, сопоставленная с шаблоном в круглых скобках, сохраняется в памяти для использования в будущем. Данные из подгрупп могут быть в дальнейшем применены в том же сопоставлении или поиске либо извлечены для последующей обработки. Некоторые примеры извлечения подгрупп будут рассматриваться в конце раздела 1.3.9.

Сопоставление с подгруппами открывает одно из очень важных направлений работы с регулярными выражениями. В частности, необходимость использования подгрупп обусловлена тем, что иногда кроме самого сопоставления с шаблоном требуется извлечь подстроку, сопоставленную с шаблоном. Например, предположим, что по условиям задачи необходимо не только провести сопоставление с шаблоном `\w+ - \d+`, но и отдельно сохранить первую, алфавитную, часть, и вторую, цифровую. Извлечение данных может также потребоваться, например, по той причине, что после каждого успешного сопоставления нужно будет определить, какими именно являются те строки, которые были сопоставлены с применяемыми шаблонами регулярных выражений.

После добавления круглых скобок к обоим подшаблонам, в данном случае, `(\w+) - (\d+)`, появляется возможность обеспечить доступ к каждой из сопоставленных подгрупп отдельно. Если бы нельзя было применять группирование, то пришлось бы написать специальную программу для распознавания соответствия, а затем вызывать отдельную процедуру (которую также нужно было бы написать) для интерпретации этого соответствия, чтобы извлечь необходимые части. Но той же цели можно достичь с помощью одной из функциональных возможностей модуля `re` языка Python, поэтому нет смысла повторно изобретать колесо (табл. 1.11).

Таблица 1.11. Группирование с применением круглых скобок ()

Шаблон регулярного выражения	Сопоставленные строки
<code>\d+(\.\d*)?</code>	Строки, представляющие числа с плавающей точкой в простом формате, т.е. любое количество цифр, за которым следует необязательная отдельная десятичная точка, затем нуль или большее количество цифровых символов, например, "0.004", "2", "75." и т.д.
<code>(Mr?s?\.)?[A-Z][a-z]* [A-Za-z-]+</code>	Имя и фамилия, в которых имя может быть представлено сокращенно (должно начинаться с прописной буквы, за которой следуют строчные буквы остальной части имени, если она задана), фамилия представлена полностью, а впереди находится необязательное обращение, "Mr.", "Mrs.", "Ms." или "M.". При указании фамилии предусмотрена возможность задавать несколько ее компонентов с использованием дефисов, прописных и строчных букв

1.2.9. Расширенный синтаксис регулярных выражений

Перейдем к рассмотрению последнего аспекта регулярных выражений, который еще не упоминался в этой главе. Речь идет о расширенных регулярных выражениях, которые начинаются с вопросительного знака, (`?. . .`). Описанию таких конструкций не будет отведено много времени, поскольку они в основном применяются для представления флагов, выполнения опережающей или ретроспективной проверки, а также проверки по условию перед распознаванием соответствия. При этом следует отметить, что круглые скобки, используемые в расширенных регулярных выражениях, не означают группирование, за исключением конструкции (`?P<name>`). Иначе говоря, все эти конструкции, кроме (`?P<name>`), *не создают* группу. Несмотря на то, что расширенные регулярные выражения применяются довольно редко, о них следует знать, поскольку в некоторых ситуациях без них довольно трудно обойтись (табл. 1.12).

Таблица 1.12. Применение расширенных регулярных выражений

Шаблон регулярного выражения	Определение расширенных обозначений
<code>(?:\w+\.)*</code>	Строки, которые оканчиваются точкой, такие как "google.", "twitter.", "facebook.". Обнаруженные при этом сопоставления не сохраняются для дальнейшего использования и не могут быть извлечены
<code>(?#comment)</code>	Эта конструкция не определяет сопоставление и применяется исключительно для ввода комментария
<code>(?= .com)</code>	Сопоставление происходит, только если далее следует подстрока ".com"; обработчик не изымает ни одной части целевой строки.
<code>(?! .net)</code>	Сопоставление происходит, только если далее не следует подстрока ".net"
<code>(?<=800-)</code>	Сопоставление происходит, только если данной строке предшествует подстрока "800-"; эта конструкция может применяться, допустим, для поиска номеров телефонов. В этом случае обработчик не изымает ни одной части входной строки
<code>(?<!192\ .168\.)</code>	Сопоставление происходит, только если данной строке не предшествует подстрока "192.168.". Эта конструкция может применяться, например, для фильтрации IP-адресов класса C
<code>(?(1)y x)</code>	Если сопоставленная группа 1 (<code>\1</code>) существует, провести сопоставление с <code>y</code> ; в противном случае — с <code>x</code>

1.3. Регулярные выражения и язык Python

2.5

Теперь, когда мы знаем все о регулярных выражениях, можем приступить к изучению того, как язык Python поддерживает в настоящее время регулярные выражения с помощью модуля `re`, который был введен на том этапе развития языка, который можно считать древней историей (Python 1.5), заменив устаревшие модули `regex` и `regsub`. Оба эти модуля были удалены из дистрибутива языка Python в версии 2.5, и начиная с этой версии импорт любого из них приводит к генерированию исключения `ImportError`.

Модуль `re` поддерживает более мощные и стандартизированные регулярные выражения в стиле Perl (Perl 5), что позволяет предоставлять нескольким потокам общий доступ к одним и тем же скомпилированным объектам регулярных выражений и поддерживать именованные подгруппы.

1.3.1. Модуль `re`. Основные функции и методы

Наиболее широко применяемые функции и методы модуля `re` перечислены в табл. 1.13. Многие из этих функций являются также доступными в качестве методов объектов скомпилированных регулярных выражений (объектов регулярных выражений и объектов сопоставления регулярных выражений). В этом подразделе рассматриваются две основные функции (применяемые также как методы), `match()` и `search()`, а также функция `compile()`. В следующем разделе будут представлены и другие функции, но для получения более подробных сведений рекомендуем обратиться к документации Python.

Таблица 1.13. Общие атрибуты регулярного выражения

Функция/метод	Описание
Только функция модуля <code>re</code>	
<code>compile(pattern, flags=0)</code>	Компилирует шаблон регулярного выражения <code>pattern</code> с необязательными флагами <code>flags</code> и возвращает объект регулярного выражения
Функции модуля <code>re</code> и методы объекта регулярного выражения	
<code>match(pattern, string, flags=0)</code>	Предпринимает попытку сопоставить шаблон <code>pattern</code> со строкой <code>string</code> , учитывая необязательные флаги <code>flags</code> ; возвращает объект сопоставления в случае успеха или значение <code>None</code> в случае неудачного завершения
<code>search(pattern, string, flags=0)</code>	Выполняет поиск первого вхождения шаблона <code>pattern</code> в строке с учетом необязательных флагов <code>flags</code> ; возвращает объект сопоставления в случае успеха или значение <code>None</code> в случае неудачного завершения
<code>findall(pattern, string[, flags])^a</code>	Осуществляет поиск всех (неперекрывающихся) вхождений шаблона <code>pattern</code> в строке <code>string</code> ; возвращает список сопоставлений
<code>finditer(pattern, string[, flags])^b</code>	То же, что и <code>findall()</code> , но вместо списка возвращает итератор; для каждого сопоставления итератор возвращает объект сопоставления

Окончание табл. 1.13

Функция/метод	Описание
<code>split(pattern, string, max=0)</code> ^c	Проводит разбиение строки <code>string</code> на основе разделителя регулярного выражения <code>pattern</code> и в случае успеха возвращает список полученных сопоставлений, в котором содержится не более чем <code>max</code> фрагментов (по умолчанию разбиение проводится по всем вхождениям)
<code>sub(pattern, repl, string, count=0)</code> ^c	Заменяет все вхождения шаблона регулярного выражения <code>pattern</code> в строке <code>string</code> подстрокой <code>repl</code> , выполняя подстановку вместо всех вхождений, если не задан параметр <code>count</code> (см. также функцию <code>subn()</code> , которая, кроме этого, возвращает количество выполненных подстановок)
<code>purge()</code>	Очищает кеш неявно скомпилированных шаблонов регулярного выражения
Общие методы объекта сопоставления (описания других методов см. в документации)	
<code>group(num=0)</code>	Возвращает все сопоставление (или конкретную подгруппу <code>num</code>)
<code>groups(default=None)</code>	Возвращает все сопоставленные подгруппы в виде кортежа (если сопоставления отсутствуют, кортеж пуст)
<code>groupdict(default=None)</code>	Возвращает словарь, содержащий все согласованные именованные подгруппы, в котором ключами являются имена подгрупп (если сопоставления отсутствуют, словарь пуст)
Общие атрибуты модуля (флаги для большинства функций, применяемых для работы с регулярными выражениями)	
<code>re.I, re.IGNORECASE</code>	Сопоставление без учета регистра
<code>re.L, re.LOCALE</code>	Правила сопоставления с регулярными выражениями, в которых применяются специальные символы <code>\w</code> , <code>\W</code> , <code>\b</code> , <code>\B</code> , <code>\s</code> , <code>\S</code> , зависят от региональной установки
<code>re.M, re.MULTILINE</code>	Применение этого флага приводит к тому, что специальные символы <code>^</code> и <code>\$</code> соответственно сопоставляются с началом и концом каждой строки текста (обозначенной символами конца строки) в целевой строке, а не исключительно с началом и концом всей целевой строки
<code>re.S, re.DOTALL</code>	Специальный символ точки (<code>.</code>) обычно сопоставляется с любым отдельным символом, кроме <code>\n</code> ; этот флаг указывает, что точка должна сопоставляться также и с ним
<code>re.X, re.VERBOSE</code>	Все пробельные символы и знак <code>#</code> (а также весь текст, который следует за знаком <code>#</code> в той же строке) пропускаются, что позволяет вводить комментарии и способствует повышению удобства для чтения. Пробельные символы и знак <code>#</code> не исключаются, если они представлены в классе символов или экранированы наклонной чертой влево

^a Впервые введено в Python 1.5.2; параметр `flags` добавлен в версии 2.4.^b Впервые введено в Python 2.2; параметр `flags` добавлен в версии 2.4.^c Параметр `flags` добавлен в версиях 2.7 и 3.1.



Компиляция регулярного выражения (компилировать или не компилировать)

В главе “Среда выполнения” книги *Core Python Programming* и в готовящейся к выпуску книге *Core Python Language Fundamentals* приведено описание того, каким образом код Python в конечном итоге компилируется в шестнадцатеричный код, который затем выполняется интерпретатором. В частности, в этих книгах указано, что передавая функции `eval()`, инструкции `exec` (в версии 2.x) или функции `exec()` (в версии 3.x) объектный, а не исходный код, можно повысить производительность благодаря тому, что процесс компиляции не приходится выполнять повторно. Иными словами, использование предварительно скомпилированного объектного кода способствует ускорению работы по сравнению с использованием исходного кода, поскольку в последнем случае интерпретатор так или иначе должен компилировать исходный код в объектный каждый раз перед выполнением.

Такой же принцип использования распространяется на регулярные выражения — шаблоны регулярных выражений должны быть скомпилированы в объекты регулярных выражений, прежде чем может произойти какое-либо сопоставление с шаблонами. Что касается регулярных выражений, сопоставление с которыми должно осуществляться много раз в ходе выполнения программы, то мы настоятельно рекомендуем использовать предварительную компиляцию, поскольку перед использованием регулярных выражений их компиляция все равно так или иначе происходит, так что лучше заблаговременно и однократно осуществить эту операцию в целях повышения производительности. Такую возможность предоставляет функция `re.compile()`.

Безусловно, функции модуля `re` сами кешируют скомпилированные объекты, поэтому специально предусматривать компиляцию для каждого случая применения `search()` и `match()` с одним и тем же шаблоном регулярного выражения не требуется. Тем не менее компиляция с помощью функции `re.compile()` позволяет сократить количество операций поиска в кеше и избавиться от необходимости снова и снова выполнять вызовы `search()` и `match()` применительно к одной и той же строке. Количество скомпилированных объектов регулярных выражений, которые сохраняются в кеше, может изменяться с каждой новой версией и в документации не указано. Для очистки этого кеша может применяться функция `purge()`.

1.3.2. Компиляция регулярных выражений с применением функции `compile()`

Почти все функции модуля `re`, которые вскоре будут описаны, доступны в качестве методов объектов регулярных выражений. Еще раз отметим, что предварительная компиляция не является обязательной, несмотря на то, что мы рекомендуем ее проводить. Компиляция создает объект регулярного выражения, поэтому в дальнейшей работе используются методы этого объекта; если же компиляция не проводится, применяются обычные функции. Для удобства предусмотрено, что имена функций и методов аналогичного назначения остаются одинаковыми. (По этой причине мы не приводим отдельно описания функций и методов модуля, которые являются идентичными и носят одинаковые имена, например, `search()`, `match()` и т.д.; об этом следует всегда помнить.) В большинстве примеров, приведенных в этой книге, используются сами регулярные выражения, а не скомпилированные объекты регулярных выражений, поскольку это позволяет представить примеры в немного более краткой форме. Тем не менее в нескольких примерах применяется компиляция, чтобы можно было показать, как выглядит в этом случае исходный код сценария.

Если компиляция проводится в целях получения специализированного объекта регулярного выражения, то в качестве параметров могут быть заданы необязательные флаги. Эти флаги позволяют указать, что сопоставление должно проводиться без учета регистра, задать системные настройки региональной установки для использования при сопоставлении алфавитно-цифровых символов и т.д. Дополнительные сведения об этих флагах (`re.IGNORECASE`, `re.MULTILINE`, `re.DOTALL`, `re.VERBOSE` и пр.) можно найти в табл. 1.13 и в официальной документации. Флаги могут применяться в различных сочетаниях, создаваемых с помощью побитового оператора ИЛИ (`|`).

Эти флаги доступны также в качестве параметров для большинства функций модуля `re`. Однако если флаги должны использоваться с методами, то они уже должны быть встроены в скомпилированные объекты регулярных выражений. Еще один вариант состоит в том, что можно использовать обозначение `(?F)`, непосредственно внедренное в регулярное выражение, где *F* может включать один или несколько из следующих символов: *i* (обозначает `re.I/IGNORECASE`), *m* (обозначает `re.M/MULTILINE`), *s* (обозначает `re.S/DOTALL`) и т.д. Если должно быть задано несколько символов, то они объединяются в одной строке без использования побитового оператора ИЛИ; например, `(?im)` указывает, что применяются два флага, `re.IGNORECASE` и `re.MULTILINE`.

1.3.3. Объекты сопоставления и методы

`group()` и `groups()`

При работе с регулярными выражениями, кроме самого объекта регулярного выражения, применяется еще один объект — объект сопоставления. Объекты сопоставления представляют собой объекты, возвращаемые после успешных вызовов `match()` или `search()`. Объекты сопоставления имеют два основных метода — `group()` и `groups()`.

Метод `group()` при вызове без параметров возвращает все сопоставление, а если в вызове метода указана конкретная группа, возвращает эту подгруппу. Метод `groups()` просто возвращает кортеж, состоящий из одной или всех подгрупп. В отличие от метода `group()`, который при вызове без параметров возвращает все сопоставление, метод `groups()`, будучи вызванным без параметров, возвращает пустой кортеж.

Регулярные выражения Python позволяют также определять именованные сопоставления, которые не рассматриваются в этом вводном разделе. Рекомендуем читателю ознакомиться со всей документацией по модулю `re` для получения полных сведений о наиболее сложных нюансах его применения, которые были здесь пропущены.

1.3.4. Согласование со строками с применением функции `match()`

В первую очередь рассмотрим принадлежащие к модулю `re` функцию `match()` и имеющий то же имя метод объекта регулярного выражения. В функции `match()` предпринимаются попытки сопоставить шаблон со строкой, начиная с ее начала. Если сопоставление произведено успешно, возвращается объект сопоставления; в случае неудачного завершения возвращается `None`. Для получения результатов успешного сопоставления используется метод `group()` объекта сопоставления. Ниже приведен пример применения `match()` (и `group()`).

```
>>> m = re.match('foo', 'foo') # строка соответствует шаблону
>>> if m is not None:           # в случае успеха показать результаты сопоставления
...     m.group()
...
'foo'
```

Шаблон "foo" точно сопоставляется со строкой "foo". С помощью интерактивного интерпретатора можно также проверить, действительно ли `m` представляет собой пример объекта сопоставления:

```
>>> m # проверка того, что объект сопоставления возвращает <re.MatchObject instance
at 80ebf48>
```

Ниже приведен пример сопоставления, потерпевшего неудачу, в результате чего возвращено значение `None`.

```
>>> m = re.match('foo', 'bar') # шаблон не соответствует строке
>>> if m is not None: m.group() # (Однострочная версия с конструкцией if)
...
>>>
```

В предыдущем примере попытка сопоставления окончилась неудачей, поэтому `m` присвоено значение `None` и не предприняты какие-либо действия, поскольку именно это было предусмотрено в применении оператора `if`. В остальных рассматриваемых примерах мы постараемся для краткости обойтись без проверки с помощью оператора `if`, если это возможно, но на практике рекомендуется всегда предусматривать такую проверку, чтобы предотвратить возникновение исключений `AttributeError`. (Дело в том, что при неудачном завершении возвращается значение `None`, которое не имеет атрибута (метода) `group()`.)

Сопоставление выполняется успешно, даже если строка длиннее шаблона при условии, что обеспечивается сопоставление с шаблоном, начиная с начала строки. Например, применение шаблона "foo" позволяет найти сопоставление в строке "food on the table", поскольку при этом происходит сопоставление с шаблоном с начала строки:

```
>>> m = re.match('foo', 'food on the table') # сопоставление выполняется успешно
>>> m.group()
'foo'
```

Вполне очевидно, что сопоставление начиная с начала строки было осуществлено успешно, несмотря на то, что строка длиннее, чем шаблон. Подстрока "foo" представляет результат сопоставления, который был извлечен из более длинной строки.

Иногда можно даже вообще обойти этап сохранения результатов, поскольку это допускает объектно-ориентированный характер Python:

```
>>> re.match('foo', 'food on the table').group()
'foo'
```

Следует отметить, что в одном из предыдущих примеров вследствие неудачного сопоставления было сгенерировано исключение `AttributeError`.

1.3.5. Поиск шаблона в строке с помощью функции `search()` (поиск вместо сопоставления)

Чаще всего искомые шаблоны находятся где-либо в середине строки, а не в самом ее начале. Именно для поиска таких шаблонов применяется функция `search()`. Поиск осуществляется точно так же, как сопоставление, не считая того, что при поиске решается задача обнаружения первого вхождения заданного шаблона регулярного выражения в произвольной позиции строкового параметра. И в этом случае при успешном завершении происходит возврат объекта сопоставления; в противном случае возвращается значение `None`.

Рассмотрим на примерах, в чем состоят различия между функциями `match()` и `search()`. Вначале предпримем попытку сопоставления шаблона со строкой, имеющей большую по сравнению с ним длину. На этот раз попытаемся сопоставить ту же строку "foo" со строкой "seafood":

```
>>> m = re.match('foo', 'seafood')    # сопоставление не произошло
>>> if m is not None: m.group()
...
>>>
```

Вполне очевидно, в данном случае попытка сопоставления оказалась неудачной. В функции `match()` осуществляется попытка сопоставить шаблон со строкой с самого начала. В данном случае буква "f" в шаблоне сопоставляется с буквой "s" в строке, что немедленно приводит к неудачному завершению. Но мы видим, что строка "foo" присутствует в строке "seafood" (в другом месте), поэтому необходимо найти способ заставить интерпретатор Python подтвердить ее наличие. Для этого предназначена функция `search()`, которая вместо сопоставления всей строки с самого начала, ищет первое вхождение шаблона в строке. В функции `search()` обработка строки производится исключительно слева направо.

```
>>> m = re.search('foo', 'seafood')    # search() вместо match()
>>> if m is not None: m.group()
...
'foo'    # search() работает там, где не работает match()
>>>
```

Отметим также, что функциям `match()`, как и `search()`, можно передавать необязательные флаги `flags`, которые описаны ранее в разделе 1.3.2. Наконец, следует отметить, что эквивалентным методом объекта регулярного выражения можно передавать необязательные параметры `pos` и `endpos`, позволяющие указать границы поиска в целевой строке.

В оставшейся части этого подраздела будет приведено несколько примеров с использованием методов объекта регулярного выражения, возвращаемого функциями `match()` и `search()`, а также методов объекта сопоставления, возвращаемого функциями `group()` и `groups()`, которые показывают различные способы использования регулярных выражений в языке Python. В этих примерах будут показаны почти все специальные символы и знаки, которые входят в состав синтаксиса регулярных выражений.

1.3.6. Сопоставление с несколькими строками (|)

В разделе 1.2 был показан пример использования символа канала в регулярном выражении `bat|bet|bit`. Ниже приведен фрагмент кода, демонстрирующий применение этого регулярного выражения в языке Python.

```
>>> bt = 'bat|bet|bit'          # шаблон регулярного выражения: bat, bet, bit
>>> m = re.match(bt, 'bat')     # обнаружено соответствие с 'bat'
>>> if m is not None: m.group()
...
'bat'
>>> m = re.match(bt, 'blt')     # соответствие с 'blt' не обнаружено
>>> if m is not None: m.group()
...
>>> m = re.match(bt, 'He bit me!') # не соответствует строке
>>> if m is not None: m.group()
...
>>> m = re.search(bt, 'He bit me!') # search() нашла подстроку 'bit'
>>> if m is not None: m.group()
...
'bit'
```

1.3.7. Сопоставление с любым отдельным символом (.)

В следующих примерах показано, что точка не может быть сопоставлена со специальным символом `\n` или с отсутствующим символом, т.е. с пустой строкой:

```
>>> anyend = '.end'
>>> m = re.match(anyend, 'bend') # точка соответствует 'b'
>>> if m is not None: m.group()
...
'bend'
>>> m = re.match(anyend, 'end')  # нет соответствий
>>> if m is not None: m.group()
...
>>> m = re.match(anyend, '\nend') # любой символ, кроме \n
>>> if m is not None: m.group()
...
>>> m = re.search('.end', 'The end.') # точка соответствует пробелу
>>> if m is not None: m.group()
...
' end'
```

В следующем примере поиска точки в числе с плавающей точкой (в числе с десятичной точкой) с помощью регулярного выражения знак точки экранируется с использованием обратной косой черты, поэтому он утрачивает свое функциональное назначение:

```
>>> patt314 = '3.14'          # точка в регулярном выражении
>>> pi_patt = '3\\.14'        # точка, заданная как литерал (десятичная точка)
>>> m = re.match(pi_patt, '3.14') # точное соответствие
>>> if m is not None: m.group()
...
'3.14'
>>> m = re.match(patt314, '3014') # точка соответствует цифре '0'
```

```
>>> if m is not None: m.group()
...
'3014'
>>> m = re.match(patt314, '3.14') # точка соответствует знаку '.'
>>> if m is not None: m.group()
...
'3.14'
```

1.3.8. Создание классов символов ([[]])

Выше в данной главе подробно рассматривалось регулярное выражение `[cr][23][dp][o2]` и было показано, чем оно отличается от `r2d2|c3po`. В следующих примерах будет показано, что регулярное выражение `r2d2|c3po` является более строгим по сравнению с `[cr][23][dp][o2]`:

```
>>> m = re.match('[cr][23][dp][o2]', 'c3po') # соответствует 'c3po'
>>> if m is not None: m.group()
...
'c3po'
>>> m = re.match('[cr][23][dp][o2]', 'c2do') # соответствует 'c2do'
>>> if m is not None: m.group()
...
'c2do'
>>> m = re.match('r2d2|c3po', 'c2do') # не соответствует 'c2do'
>>> if m is not None: m.group()
...
>>> m = re.match('r2d2|c3po', 'r2d2') # соответствует 'r2d2'
>>> if m is not None: m.group()
...
'r2d2'
```

1.3.9. Повторение, специальные символы и группирование

В регулярных выражениях чаще всего применяются конструкции, предусматривающие использование специальных символов, поиск нескольких вхождений шаблонов регулярного выражения и применение круглых скобок для группирования и извлечения шаблонов вторичного сопоставления. Одним из конкретных рассматриваемых регулярных выражений было выражение, описывающее адреса электронной почты в наиболее простой форме (`\w+@\w+\.`com). Однако иногда возникает необходимость обеспечить сопоставление с более сложными адресами электронной почты, чем допускает это регулярное выражение. Приведенное выше регулярное выражение потребует изменить, в частности, для того, чтобы оно поддерживало дополнительное имя хоста, которое предшествует домену, например `http://www.xxx.com/`, а не обеспечивало применение лишь конструкции типа `xxx.com`, обозначающей весь домен. Для указания на то, что имя хоста является необязательным, мы создадим шаблон, который сопоставляется с именем хоста (за которым следует точка), применим вслед за ним оператор `?`, указывающий, что допускается вхождение одной копии этого шаблона или его отсутствие, после этого вставим необязательное регулярное выражение в предыдущий вариант регулярного выражения следующим образом: `\w+@(\w+\.)?\w+\.`com. Как показывают следующие примеры, теперь перед подстрокой `.com` могут быть заданы одно или два имени:

```
>>> patt = '\w+@(\w+\.)?\w+\.com'
>>> re.match(patt, 'nobody@xxx.com').group()
'nobody@xxx.com'
>>> re.match(patt, 'nobody@www.xxx.com').group()
'nobody@www.xxx.com'
```

Кроме того, можно дополнительно расширить рассматриваемый пример, чтобы обеспечить применение любого количества промежуточных имен поддоменов с помощью следующего шаблона. Заслуживает внимания то, насколько важным оказалось небольшое изменение — переход от использования `? к *`: `\w+@(\w+\.)*\w+\.com`:

```
>>> patt = '\w+@(\w+\.)*\w+\.com'
>>> re.match(patt, 'nobody@www.xxx.yyy.zzz.com').group()
'nobody@www.xxx.yyy.zzz.com'
```

Тем не менее необходимо сделать оговорку, что для обеспечения сопоставления со всеми возможными символами недостаточно использовать исключительно алфавитно-цифровые символы, которые могут входить в состав адресов электронной почты. Приведенные выше шаблоны регулярного выражения не соответствуют таким доменам, как `xxx-yyy.com`, или другим доменам, в именах которых содержатся символы класса `\W`.

Выше уже рассматривались преимущества использования круглых скобок для сопоставления с подгруппами и сохранения подгрупп для дальнейшей обработки в сравнении с тем подходом, который предусматривает применение отдельной процедуры для продолжения интерпретации строки после определения основного соответствия регулярному выражению. В частности, было показано, как применяется простой шаблон регулярного выражения, состоящий из строки алфавитно-цифровых символов и числа, разделенных дефисом, `\w+-\d+`, и описано, что добавление подгрупп в целях формирования нового регулярного выражения, `(\w+) - (\d+)`, позволяет решить эту задачу. Рассмотрим, как действует этот исходный вариант регулярного выражения:

```
>>> m = re.match('\w\w\w-\d\d\d', 'abc-123')
>>> if m is not None: m.group()
...
'abc-123'

>>> m = re.match('\w\w\w-\d\d\d', 'abc-xyz')
>>> if m is not None: m.group()
...
>>>
```

В предыдущем примере кода было приведено регулярное выражение, предназначенное для распознавания трех алфавитно-цифровых символов, за которыми следуют три цифры. Проверка этого регулярного выражения на примере сопоставления со строкой `"abc-123"` приводит к получению положительных результатов, тогда как попытка провести сопоставление со строкой `"abc-xyz"` заканчивается неудачей. Теперь попытаемся внести изменения в это регулярное выражение так, как описано выше, чтобы получить возможность извлекать с его помощью алфавитно-цифровую строку и число. Следует отметить, что мы теперь можем использовать метод `group()` для получения доступа к отдельным подгруппам или метод `groups()`, позволяющий получить кортеж всех сопоставленных подгрупп:


```
>>> m = re.match('(\w\w\w)-(\d\d\d)', 'abc-123')
>>> m.group()      # все сопоставление
'abc-123'
>>> m.group(1)     # подгруппа 1
'abc'
>>> m.group(2)     # подгруппа 2
'123'
>>> m.groups()     # все подгруппы
('abc', '123')
```

Легко видеть, что метод `group()` при вызове без параметров позволяет получить результаты всего сопоставления, но может также использоваться для выборки отдельных подгрупп из сопоставления. Можно также использовать метод `groups()` для получения кортежа всех сопоставлений в подстроке.

Ниже приведен более простой пример, в котором демонстрируются различные перестановки групп. По-видимому, этот пример позволяет еще лучше понять, что происходит при использовании групп:

```
>>> m = re.match('ab', 'ab')      # подгрупп нет
>>> m.group()                     # полное соответствие
'ab'
>>> m.groups()                   # все подгруппы
()
>>>
>>> m = re.match('(ab)', 'ab')    # одна подгруппа
>>> m.group()                     # полное соответствие
'ab'
>>> m.group(1)                   # подгруппа 1
'ab'
>>> m.groups()                   # все подгруппы
('ab',)
>>>
>>> m = re.match('(a)(b)', 'ab')  # две подгруппы
>>> m.group()                     # полное соответствие
'ab'
>>> m.group(1)                   # подгруппа 1
'a'
>>> m.group(2)                   # подгруппа 2
'b'
>>> m.groups()                   # все подгруппы
('a', 'b')
>>>
>>> m = re.match('(a(b))', 'ab')  # две подгруппы
>>> m.group()                     # полное соответствие
'ab'
>>> m.group(1)                   # подгруппа 1
'ab'
>>> m.group(2)                   # подгруппа 2
'b'
>>> m.groups()                   # все подгруппы
('ab', 'b')
```

1.3.10. Сопоставление с началом и концом строк и с границами слов

Следующие примеры посвящены изучению позиционных операторов регулярных выражений. Эти примеры в большей степени подходят для поиска, а не для сопоставления, поскольку действие `match()` всегда начинается с начала строки.

```
>>> m = re.search('^The', 'The end.')          # есть соответствие
>>> if m is not None: m.group()
...
'The'
>>> m = re.search('^The', 'end. The')          # не в начале
>>> if m is not None: m.group()
...
>>> m = re.search(r'\bthe', 'bite the dog')    # на границе
>>> if m is not None: m.group()
...
'the'
>>> m = re.search(r'\bthe', 'bitethe dog')     # границы нет
>>> if m is not None: m.group()
...
>>> m = re.search(r'\Bthe', 'bitethe dog')     # границы нет
>>> if m is not None: m.group()
...
'the'
```

В этом примере мы впервые применяем *бесформатные строки* (строки в форме `r'string'`). Чтобы ознакомиться с описанием того, с какой целью применяются строки такого типа, перейдите к примечанию “Использование бесформатных строк Python”, которое находится ближе к концу этой главы. Вообще говоря, во многих случаях в регулярных выражениях целесообразно применять именно бесформатные строки.

Есть еще четыре функции модуля `re` (которые могут также применяться в качестве методов объектов регулярного выражения), о которых следует знать: `findall()`, `sub()`, `subn()` и `split()`.

1.3.11. Поиск каждого вхождения с помощью функций `findall()` и `finditer()`

Функция `findall()` обеспечивает поиск всех неперекрывающихся вхождений шаблона регулярного выражения в строке. Эта функция аналогична функции `search()` в том, что осуществляет поиск в строке, но отличается от функций `match()` и `search()` тем, что вызов функции `findall()` всегда возвращает список. Возвращаемый список пуст, если не были обнаружены какие-либо вхождения, но в случае успеха полученный список состоит из всех найденных соответствий (сгруппированных слева направо, в той последовательности, в какой они представлены в строке).

```
>>> re.findall('car', 'car') ['car']
>>> re.findall('car', 'scary')
['car']
>>> re.findall('car', 'carry the barcardi to the car') ['car', 'car', 'car']
```

Поиск с применением регулярного выражения с подгруппами приводит к получению более сложного по структуре списка. Это обусловлено тем, что подгруппы обеспечивают выделение более мелких шаблонов, которые вместе составляют единое регулярное выражение. Поэтому с помощью подгрупп, например, можно выделить из всего номера телефона код города или извлечь из всего адреса электронной почты регистрационное имя, входящее в состав этого адреса.

Если регулярное выражение включает подгруппы, то функция `findall()` возвращает список, состоящий из кортежей, каждый из которых соответствует отдельному результату успешного сопоставления с регулярным выражением. Элементы кортежей соответствуют подгруппам этого выражения. Количество кортежей в списке определяется количеством найденных соответствий с регулярным выражением. В частности, если обнаружено только одно соответствие, список состоит из одного кортежа. На первых порах бывает трудно разобраться, как действует функция `findall()`, поэтому рекомендуем читателям рассмотреть несколько примеров, чтобы понять, что происходит.

2.2

Функция `finditer()`, которая была введена в состав языка в версии Python 2.2, аналогична по своему назначению функции `findall()`, поскольку позволяет найти в строке все вхождения регулярного выражения, но предъявляет меньше требований к оперативной памяти. Основное различие между функциями `findall()` и `finditer()` состоит в том, что последняя возвращает итератор, а не список, а итератор позволяет обрабатывать объекты сопоставления один за другим в цикле. В следующем примере, где представлено несколько групп в одной строке, демонстрируются различия между этими функциями:

```
>>> s = 'This and that.'
>>> re.findall(r'(th\w+) and (th\w+)', s, re.I) [('This', 'that')]
>>> re.finditer(r'(th\w+) and (th\w+)', s,
... re.I).next().groups() ('This', 'that')
>>> re.finditer(r'(th\w+) and (th\w+)', s,
... re.I).next().group(1)
'This'
>>> re.finditer(r'(th\w+) and (th\w+)', s,
... re.I).next().group(2)
'that'
>>> [g.groups() for g in re.finditer(r'(th\w+) and (th\w+)',
... s, re.I)] [('This', 'that')]
```

В следующем примере обнаруживается несколько соответствий с одной группой в одной строке:

```
>>> re.findall(r'(th\w+)', s, re.I) ['This', 'that']
>>> it = re.finditer(r'(th\w+)', s, re.I)
>>> g = it.next()
>>> g.groups() ('This',)
>>> g.group(1)
'This'
>>> g = it.next()
>>> g.groups() ('that',)
>>> g.group(1)
'that'
>>> [g.group(1) for g in re.finditer(r'(th\w+)', s, re.I)] ['This', 'that']
```

Важно отметить, что для получения с помощью функции `finditer()` таких же результатов, как при использовании `findall()`, необходимо было выполнить дополнительную работу.

Наконец, как и функции `match()` и `search()`, методы `findall()` и `finditer()` могут получать необязательные параметры `pos` и `endpos`, позволяющие задавать границы поиска в целевой строке, как было описано ранее в этой главе.

1.3.12. Поиск и замена с помощью функций `sub()` и `subn()`

Для поиска и замены предусмотрены две функции (два метода): `sub()` и `subn()`. Они являются почти полностью идентичными и заменяют все сопоставленные вхождения шаблона регулярного выражения в строке с учетом определенных условий. Для замены обычно применяется строка, но может быть также задана функция, которая возвращает строку замены. Функция `subn()` по своему назначению полностью аналогична функции `sub()`, но возвращает также общее количество выполненных подстановок; строка, полученная в результате подстановки, и количество подстановок возвращаются в виде кортежа из двух элементов.

```
>>> re.sub('X', 'Mr. Smith', 'attn: X\n\nDear X,\n')
'attn: Mr. Smith\012\012Dear Mr. Smith,\012'
>>>
>>> re.subn('X', 'Mr. Smith', 'attn: X\n\nDear X,\n') ('attn: Mr. Smith\012\012Dear Mr.
Smith,\012', 2)
>>>
>>> print re.sub('X', 'Mr. Smith', 'attn: X\n\nDear X,\n')
attn: Mr. Smith

Dear Mr. Smith,

>>> re.sub('[ae]', 'X', 'abcdef')
'XbcdXf'
>>> re.subn('[ae]', 'X', 'abcdef') ('XbcdXf', 2)
```

Как было показано в одном из предыдущих разделов, предоставляется возможность не только извлекать из результатов сопоставления содержимое группы с указанным номером, применяя метод `group()` объекта сопоставления, но и задавать номер группы в строке замены с помощью конструкции `\N`, где `N` представляет номер группы. В следующем примере показано, как преобразовать представление даты из формата, принятого в США, `MM/DD/YY{, YY}`, в формат, используемый в некоторых других странах, `DD/MM/YY{, YY}`:

```
>>> re.sub(r'(\d{1,2})/(\d{1,2})/(\d{2})\d{4}',
... r'\2/\1/\3', '2/20/91') # Да, язык Python...
'20/2/91'
>>> re.sub(r'(\d{1,2})/(\d{1,2})/(\d{2})\d{4}',
... r'\2/\1/\3', '2/20/1991') # ... создан более 20 лет тому назад!
'20/2/1991'
```

1.3.13. Разбиение (по шаблону разграничения) с помощью метода `split()`

Метод `split()` объекта регулярного выражения, представленный в модуле `re`, действует подобно своему аналогу — функции, предназначенной для обработки строк, но позволяет проводить разбиение не только по входениям фиксированной строки, но и по входениям шаблона регулярного выражения, что позволяет существенно расширить возможности разбиения строк. Если требуется ограничить количество отдельных фрагментов строки, полученных в результате ее разбиения в местах входения шаблона, то можно определить максимальное количество разбиений с помощью параметра `max`, задавая его значение, отличное от нуля.

Если разделитель задан не с помощью регулярного выражения, в котором используются специальные знаки, обеспечивающие сопоставление с несколькими шаблонами, то конструкция `re.split()` действует точно так же, как `str.split()`, как показано в следующем примере (демонстрирующем разбиение строки в тех местах, где находятся отдельные двоеточия):

```
>>> re.split(':', 'str1:str2:str3') ['str1', 'str2', 'str3']
```

Этот пример достаточно простой, но иногда возникает необходимость решать более сложную задачу. В качестве примера можно указать создание средства синтаксического анализа для веб-сайта, такого как Google или Yahoo! Maps. Могут возникать и такие ситуации, когда, допустим, приходится обрабатывать данные почтового адреса, введенные пользователями, которые вправе указывать только город и штат, или город и почтовый индекс, или все эти три значения. Для этого требуется применение более мощных средств обработки по сравнению с применявшимися ранее обычными функциями разбиения по строкам:

```
>>> import re
>>> DATA = (
...     'Mountain View, CA 94040',
...     'Sunnyvale, CA',
...     'Los Altos, 94023',
...     'Cupertino 95014',
...     'Palo Alto CA',
... )
>>> for datum in DATA:
...     print re.split(' |(?= (?:\d{5}){[A-Z]{2})) ', datum)
...
['Mountain View', 'CA', '94040'] ['Sunnyvale', 'CA']
['Los Altos', '94023'] ['Cupertino', '95014'] ['Palo Alto', 'CA']
```

В приведенном выше регулярном выражении предусмотрен простой компонент, который обеспечивает разбиение по строке, состоящей из запятой и пробела (", "). Более сложным является последнее регулярное выражение, в котором применяются некоторые расширенные обозначения, описанные в следующем подразделе. На словах действия, выполняемые этим регулярным выражением, можно описать следующим образом: осуществлять также разбиение по одному пробелу, но только если за этим пробелом непосредственно следуют пять цифр (почтовый индекс) или две прописные буквы (сокращенное обозначение штата США). Это позволяет избежать разбиения по пробелам в названиях городов, состоящих из нескольких слов.

Безусловно, это всего лишь упрощенное регулярное выражение, которое может стать отправной точкой для создания приложения, обеспечивающего интерпретацию информации о местоположении. В этом регулярном выражении не осуществляется обработка (или происходит неудачное завершение) сокращенных обозначений штатов строчными буквами или полного написания названий штатов, названий улиц, кодов стран, почтовых индексов в формате ZIP+4 (почтовых индексов с девятью цифрами), географических координат — широты и долготы, многочисленных пробелов и т.д. Это регулярное выражение приведено лишь в качестве простой демонстрации того, что формат вызова `re.split()` предоставляет большие возможности по сравнению с `str.split()`.

Приведенные выше примеры показывают, что решение задач разбиения строк становится намного более эффективным при использовании регулярных выражений. Но это не означает, что всегда следует прибегать лишь к этому способу разбиения. Необходимо выбирать инструмент, наиболее подходящий для выполнения задания. Если вполне можно воспользоваться разбиением с помощью функций и методов для работы со строками, то нет смысла усложнять программу и делать ее менее производительной, используя регулярные выражения.

1.3.14. Расширенный синтаксис (? . . .)

В регулярных выражениях Python поддерживается целый ряд расширенных обозначений. В данном разделе рассматриваются некоторые из этих расширенных обозначений и представлены примеры их использования.

Прежде всего, набор параметров `(?iImsux)` позволяет пользователям задавать один или несколько флагов непосредственно в регулярном выражении, не прибегая к применению функции `compile()` или других функций модуля `re`. Ниже приведено несколько примеров, в которых используется флаг `re.I/IGNORECASE`, иногда в сочетании с флагом `re.M/MULTILINE`:

```
>>> re.findall(r'(?i)yes', 'yes? Yes. YES!!!') ['yes', 'Yes', 'YES']
>>> re.findall(r'(?i)th\w+', 'The quickest way is through this tunnel.')
['The', 'through', 'this']
>>> re.findall(r'(?im)^(th\w+)', """
... This line is the first,
... another line,
... that line, it's the best
... """)
['This line is the first', 'that line']
```

В предыдущих примерах обработка данных без учета регистра символов была довольно простой. В последнем примере показано, что применение флага, обеспечивающего так называемую “многострочную” обработку, позволяет выполнять поиск отдельно в нескольких подстроках целевой строки, а не рассматривать всю строку как единое целое. Заслуживает внимания то, что происходит пропуск отдельных экземпляров слова “the”, поскольку они не находятся в начале соответствующих подстрок.

В следующих двух примерах демонстрируется использование флага `re.S/DOTALL`. Этот флаг указывает, что точка (.) может применяться для представления символов `\n` (тогда как обычно она представляет все символы, кроме `\n`):

```
>>> re.findall(r'th.+', '''
... The first line
... the second line
```

```
... the third line

... '')
['the second line', 'the third line']
>>> re.findall(r'(?s)th.+', '')
... The first line
... the second line
... the third line
... '')
['the second line\nthe third line\n']
```

Весьма интересным является флаг `re.X/VERBOSE`; он позволяет пользователям создавать регулярные выражения, более удобные для восприятия, поскольку подавляет пробельные символы в регулярных выражениях (кроме тех, что представлены в классах символов или экранированы обратной косой чертой). Кроме того, этот флаг позволяет использовать знаки `#` (называемые также знаками решетки, комментария или фунта) для обозначения начала комментария. Знак `#` также не рассматривается как обозначение комментария, если представлен в классе символов или экранирован обратной косой чертой:

```
>>> re.search(r'''(?x)
...  \((\d{3})\) # код города
...  [ ]        # пробел
...  (\d{3})    # префикс
...  -         # дефис
...  (\d{4})    # номер конечной точки
... ''', '(800) 555-1212').groups() ('800', '555', '1212')
```

Обозначение `(?:...)` также имеет широкие перспективы применения. Оно позволяет группировать части регулярного выражения, но не сохранять полученные группы для их извлечения или использования в дальнейшем. Это удобно, если необходимо группирование, но нет смысла сохранять результаты сопоставления с некоторыми группами, которые так и не будут использоваться:

```
>>> re.findall(r'http://(?:\w+\.)*(\w+\.com)',
...  'http://google.com/ http://www.google.com/ http://code.google.com')
['google.com', 'google.com', 'google.com']
>>> re.search(r'\((?P<areacode>\d{3})\) (?P<prefix>\d{3})-(?:\d{4})',
...  '(800) 555-1212').groupdict()
{'areacode': '800', 'prefix': '555'}
```

Обозначения `(?P<name>)` и `(?P=name)` применяются в сочетании друг с другом. Первое из них позволяет сохранять результаты сопоставления с использованием идентификаторов групп, представленных в виде имени. Обычно группы обозначаются с помощью последовательных чисел, начиная с единицы и заканчивая *N*. Для групп с числовыми обозначениями применяется способ выборки в формате `\1`, `\2`, ..., `\N`. Выборка содержимого группы с обозначением в виде имени, *name*, осуществляется с помощью конструкции `\g<name>`:

```
>>> re.sub(r'\((?P<areacode>\d{3})\) (?P<prefix>\d{3})-(?:\d{4})',
...  '\(\g<areacode>\) \g<prefix>-xxxx', '(800) 555-1212')
'(800) 555-xxxx'
```

Использование последнего варианта позволяет повторно применять шаблоны в одном и том же регулярном выражении, не указывая снова одинаковый шаблон

в том же регулярном выражении, как в этом примере. Очевидно, что показанная здесь конструкция позволяет проверить, правильно ли выполнена нормализация номеров телефонов. Это не слишком эстетически привлекательные и краткие версии. За ними следует более качественный пример использования конструкции `(?x)`, который показывает, как должен выглядеть код, более удобный для чтения:

```
>>> bool(re.match(r'\((?P<areacode>\d{3})\) (?P<prefix>\d{3})- (?P<number>\d{4})
(?P=areacode)-(?P=prefix)-(?P=number)
1:(?P=areacode) (?P=prefix) (?P=number)',
... '(800) 555-1212 800-555-1212 18005551212')) True
>>> bool(re.match(r'^(?x)
...
... # сопоставление с (800) 555-1212, сохранение кода города, префикса и номера
... \((?P<areacode>\d{3})\) [ ] (?P<prefix>\d{3})-(?P<number>\d{4})
...
... # пробел
... [ ]
...
... # сопоставление с 800-555-1212
... (?P=areacode)-(?P=prefix)-(?P=number)
...
... # пробел
... [ ]
...
... # сопоставление с 18005551212
... 1(?P=areacode) (?P=prefix) (?P=number)
...
... ', '(800) 555-1212 800-555-1212 18005551212')) True
```

Обозначения `(?=...)` и `(?!...)` можно использовать для опережающей проверки в целевой строке, не продвигаясь фактически по этим символам. Первую операцию принято называть *положительной опережающей проверкой*, а вторая представляет собой *отрицательную опережающую проверку*. В следующих примерах показано, как обеспечить обработку данных об именах и фамилиях, если нас интересуют только люди, которые имеют фамилию "van Rossum". За ними следует пример, который показывает, как пропустить адреса электронной почты, начинающиеся со строки "noreply" или "postmaster".

Третий фрагмент кода представляет собой еще одну демонстрацию различий между функциями `findall()` и `finditer()`. Функция `finditer()` используется для формирования списка адресов электронной почты, в котором представлены одинаковые регистрационные имена, находящиеся в разных доменах. Эта функция требует меньше оперативной памяти, поскольку позволяет пропустить создание промежуточного списка, который в дальнейшем должен быть отброшен.

```
>>> re.findall(r'\w+(?= van Rossum)',
... '''
... Guido van Rossum
... Tim Peters
... Alex Martelli
... Just van Rossum
... Raymond Hettinger
... ''')
['Guido', 'Just']
>>> re.findall(r'(?m)^\s+(?!noreply|postmaster) (\w+)',
... '''
```



```
... sales@phptr.com
... postmaster@phptr.com
... eng@phptr.com
... noreply@phptr.com
... admin@phptr.com
... '')
['sales', 'eng', 'admin']
>>> ['%s@aw.com' % e.group(1) for e in \
re.finditer(r'(?m)^\s+(?!noreply|postmaster) (\w+)',
... '')
... sales@phptr.com
... postmaster@phptr.com
... eng@phptr.com
... noreply@phptr.com
... admin@phptr.com
... '']]
['sales@aw.com', 'eng@aw.com', 'admin@aw.com']
```

В последних примерах демонстрируется использование условного сопоставления с регулярным выражением. Здесь рассматривается задача обработки строк, состоящих из символов специализированного алфавита, который включает только символы 'x' и 'y'. В результате обработки должна быть получена сжатая строка, в которой два одинаковых символа заменяются одним. Иными словами, в результирующей строке не должно быть двух подряд идущих одинаковых символов. В ней должны присутствовать только символы 'x', за которыми следуют символы 'y', или наоборот:

```
>>> bool(re.search(r'(?:(x)|y)(?!(1)y|x)', 'xy'))
True
>>> bool(re.search(r'(?:(x)|y)(?!(1)y|x)', 'xx'))
False
```

1.3.15. Разное

Специальные символы регулярных выражений и специальные символы ASCII иногда можно перепутать. Например, символ `\n` (символ ASCII) используется для представления символа обозначения конца строки. С другой стороны, символ `\d` (символ регулярного выражения) служит для сопоставления в регулярном выражении с отдельным цифровым знаком.

Таким образом, при использовании символов ASCII, совпадающих с символами регулярных выражений, возникают недоразумения. Поэтому в следующем примечании рекомендуется использовать бесформатные строки Python, что позволяет избежать любых подобных проблем. Еще одно предостережение: состав наборов алфавитно-цифровых символов, представляемых обозначениями `\w` и `\W`, изменяется в зависимости от флага языковой настройки (`re.L/LOCALE`) и флага Юникода (`re.U/UNICODE`).



Использование бесформатных строк Python

В некоторых из предыдущих примеров уже было продемонстрировано использование бесформатных строк. Необходимость во введении такого типа строк, как бесформатные строки, была во многом обусловлена потребностями формирования удобных регулярных выражений. Обычные строки подвергаются дополнительной обработке, а в ходе этого могут возникать конфликты между символами ASCII и специальными символами

регулярных выражений. В качестве примера можно указать, что специальный знак `\b` представляет символ ASCII, обозначающий возврат на один символ, но `\b` является также специальным символом регулярного выражения, обозначающим, что при сопоставлении должна учитываться граница слова. Предположим, что в шаблоне регулярного выражения необходимо применить два символа `\b`, которые отнюдь не должны рассматриваться как знаки возврата на один символ. Если шаблон не задан с помощью бесформатной строки, то приходится экранировать в шаблоне обратную косую черту в символе обозначение границы слова с использованием еще одной обратной косой черты, что приводит к получению обозначения `\\b`.

Таким образом, при формировании шаблонов в связи с необходимостью экранировать знаки обратной косой черты еще одной обратной косой чертой приходится затрачивать дополнительные усилия, которые еще больше возрастают, если количество специальных символов в строке достаточно велико. Это приводит к еще большей путанице. Автор описал бесформатные строки в главе “Последовательности” книги *Core Python Programming* и в книге *Core Python Language Fundamentals*. Бесформатные строки могут использоваться (и часто используются) для того, чтобы упростить формат регулярных выражений. В действительности многие программисты на языке Python дали себе зарок отказаться от использования простых строк при определении регулярных выражений и задают в регулярных выражениях только бесформатные строки.

Ниже приведены некоторые примеры демонстрации различия между `\b` как знаком возврата на один символ и `\b` как знаком регулярного выражения, с применением и без применения бесформатных строк.

```
>>> m = re.match('\bblow', 'blow') # возврат на один символ,
>>>                                     # соответствия нет
>>> if m: m.group()
...
>>> m = re.match('\\bblow', 'blow') # экранировано с помощью \,
>>>                                     # теперь соответствие есть
>>> if m: m.group()
...
'blow'
>>> m = re.match(r'\bblow', 'blow') # использование вместо этого
                                     # бесформатной строки
>>> if m: m.group()
...
'blow'
```

Возвращаясь к предыдущим примерам в этой главе, можно отметить, что в них не возникали какие-либо сложности при использовании символа `\d` в регулярных выражениях, притом что бесформатные строки не применялись. Это связано с тем, что в этих примерах не был задан имеющий такой же вид специальный символ ASCII, поэтому компилятор регулярных выражений мог однозначно определить, что в данном случае применяется символ сопоставления с десятичной цифрой.

1.4. Некоторые примеры регулярных выражений

Ниже приведено несколько примеров кода регулярных выражений Python, которые в большей степени напоминают применяемые на практике. Рассмотрим, например, вывод в системе POSIX (под системами POSIX подразумеваются системы, относящиеся к разновидностям Unix, такие как Linux, Mac OS X и т.д.) результатов работы команды `who`, содержащий список всех пользователей, зарегистрированных в системе:

```
$ who
wesley      console      Jun 20 20:33
wesley      pts/9         Jun 22 01:38 (192.168.0.6)
wesley      pts/1         Jun 20 20:33 (:0.0)
wesley      pts/2         Jun 20 20:33 (:0.0)
wesley      pts/4         Jun 20 20:33 (:0.0)
wesley      pts/3         Jun 20 20:33 (:0.0)
wesley      pts/5         Jun 20 20:33 (:0.0)
wesley      pts/6         Jun 20 20:33 (:0.0)
wesley      pts/7         Jun 20 20:33 (:0.0)
wesley      pts/8         Jun 20 20:33 (:0.0)
```

Предположим, что из этих результатов необходимо извлечь некоторую информацию, связанную с регистрацией пользователей, такую как регистрационное имя; терминал, с которого зарегистрировался пользователь; время и место регистрации пользователя. В предыдущем примере применение варианта `str.split()` было бы неэффективным, поскольку в выводе количество пробелов изменяется не единообразно и не согласованно. Еще одна проблема состоит в том, что значения месяца, дня и времени в отметках времени регистрации разделены пробелами. После обработки желательно объединить эти поля в одно.

Необходимо найти какой-то способ представления шаблона, который решает задачу “разбиения по двум или большему количеству пробелов”. Это можно легко сделать с помощью регулярных выражений. Не задумываясь слишком долго, мы составили шаблон регулярного выражения `\s\s+`, который предназначен для сопоставления по меньшей мере с двумя пробельными символами.

Подготовим программу и назовем ее `rewho.py`, которая читает вывод команды `who` и сохраняет, предположим, в файле с именем `whodata.txt`. Первый рассмотренный нами сценарий `rewho.py` может выглядеть примерно так:

```
import re
f = open('whodata.txt', 'r')
for eachLine in f:
    print re.split(r'\s\s+', eachLine)
f.close()
```

В приведенном выше коде также используются бесформатные строки (отличающиеся тем, что в них перед открывающейся кавычкой стоит буква “r” или “R”). В данном случае бесформатные строки применялись в основном для предотвращения преобразования специальных строковых символов, таких как `\n`, которые не относятся к категории специальных символов для шаблонов регулярных выражений. Для нас желательно, чтобы шаблоны регулярных выражений, в которых имеются символы обратной косой черты, рассматривались компилятором буквально; в противном случае пришлось бы обозначать эти символы еще одной обратной косой чертой, чтобы при их обработке не возникала неоднозначность.

Теперь перейдем к выполнению команды `who`, сохранению ее вывода в файле `whodata.txt` и последующему вызову сценария `rewho.py` для просмотра результатов:

```
$ who > whodata.txt
$ rewho.py
['wesley', 'console', 'Jun 20 20:33\012']
['wesley', 'pts/9', 'Jun 22 01:38\011(192.168.0.6)\012']
['wesley', 'pts/1', 'Jun 20 20:33\011(:0.0)\012']
```

```
['wesley', 'pts/2', 'Jun 20 20:33\011(:0.0)\012']
['wesley', 'pts/4', 'Jun 20 20:33\011(:0.0)\012']
['wesley', 'pts/3', 'Jun 20 20:33\011(:0.0)\012']
['wesley', 'pts/5', 'Jun 20 20:33\011(:0.0)\012']
['wesley', 'pts/6', 'Jun 20 20:33\011(:0.0)\012']
['wesley', 'pts/7', 'Jun 20 20:33\011(:0.0)\012']
['wesley', 'pts/8', 'Jun 20 20:33\011(:0.0)\012']
```

Эта первая попытка принесла приемлемые результаты, но оказалась не совсем правильной. Прежде всего, мы не предвидели появления отдельных знаков табуляции (символ ASCII \011) в составе вывода (ведь на экране знак табуляции не отображается и его применение приводит просто к сдвигу вправо следующей части строки). Кроме того, возможно, мы не отнеслись достаточно внимательно к обработке символов \n (символ ASCII \012), которыми завершается каждая строка на экране. Перейдем к исправлению этих недостатков, а также постараемся в целом повысить качество нашего приложения путем внесения еще нескольких изменений.

Для этого целесообразно в первую очередь обеспечить вызов команды `who` непосредственно из сценария, а не выполнять ее вне сценария и сохранять вывод в файле `whodata.txt`. Очевидно, что в последнем случае приходится дополнительно осуществлять определенные действия, а это не всегда оправдано на практике. Для вызова из нашей программы другой программы прибегнем к использованию команды `os.popen()`. Безусловно, после включения в очередную версию Python модуля `subprocess` команда `os.popen()` стала рассматриваться как устаревшая, но все же эта команда проще в использовании, а нашей основной задачей является иллюстрация применения функциональных средств `re.split()`.

Избавимся от заключительных символов \n (с помощью оператора `str.rstrip()`) и введем дополнительно обнаружение отдельных знаков табуляции как еще одних, альтернативных разграничителей для `re.split()`. В примере 1.1 представлен заключительный вариант сценария `rewho.py` для версии Python 2:

Пример 1.1. Разбиение вывода команды `who` в системе POSIX (`rewho.py`)

В этом сценарии осуществляется вызов команды `who` и происходит синтаксический анализ полученных входных данных путем разбиения этих данных по пробельным символам различных типов.

```
1  #!/usr/bin/env python
2
3  import os
4  import re
5
6  f = os.popen('who', 'r')
7  for eachLine in f:
8      print re.split(r'\s+|\t', eachLine.rstrip())
9  f.close()
```

3.x

В примере 1.2 представлен сценарий `rewho3.py`, предназначенный для версии Python 3, с дополнительным усложнением. Основным отличием от сценария для версии Python 2 является применение функции `print()` (а не инструкции `print`). Вся эта строка выделена курсивом, который показывает важные различия между версиями Python 2 и Python 3. Инструкция `with`,

2.5-2.6

которая была введена как экспериментальная в версии 2.5 и стала официальной в версии 2.6, работает с объектами, в которых специально предусмотрена ее поддержка.

Пример 1.2. Вариант сценария `rewho.py` для версии Python 3 (`rewho3.py`)

В эквивалентном варианте `rewho.py` для версии Python 3 просто осуществлена замена инструкции `print` функцией `print()`. При использовании инструкции `with` (которая стала доступной в версии Python 2.5) следует учитывать, что диспетчер контекста для файла (Python 2) или объекта ввода-вывода (Python 3) автоматически вызывает функцию `f.close()` от имени программиста.

```
1  #!/usr/bin/env python
2
3  import os
4  import re
5
6  with os.popen('who', 'r') as f
7  for eachLine in f
8  print(re.split(r'\s+||t', eachLine.strip()))
```

Объекты, для которых реализованы диспетчеры контекста, становятся применимыми для использования с инструкцией `with`. Дополнительные сведения об использовании инструкции `with` и управлении контекстом приведены в главе “Ошибки и исключения” книг *Core Python Programming* и *Core Python Language Fundamentals*. Не следует забывать, что в обоих вариантах, приведенных в этой книге (`rewho.py` или `rewho3.py`), применяется команда `who`, доступная только в системах POSIX. Кроме того, эта команда доступна на компьютерах с операционной системой Windows, на которых развернута оболочка Cygwin. На персональных компьютерах, работающих под управлением операционной системы Windows, можно попытаться вместо этого воспользоваться командой `tasklist`, но при этом потребуется выполнить дополнительную настройку. Пример организации сценария с использованием этой команды приведен далее в настоящей книге.

В примере 1.3 показан сценарий `rewhoU.py` (имя которого представляет собой сокращение от “*rewho universal*”), полученный в результате объединения сценариев `rewho.py` и `rewho3.py`. Этот сценарий может применяться в обеих версиях интерпретатора, Python 2 и Python 3. Мы пойдем на хитрость и избежим применения и того и другого средства печати, в одном случае — инструкции `print`, а в другом — функции `print()`, заменив их функцией, которую редко можно встретить в прикладном коде, предусмотренной в обеих версиях — 2.x и 3.x: `distutils.log.warn()`. Это функция, которая выводит данные в виде одной строки, поэтому, если форматированный вывод должен быть сложнее, его необходимо объединить в отдельную строку, а затем выполнить вызов. Для того чтобы указать, по какому назначению функция `distutils.log.warn()` используется в нашем сценарии, присвоим ей имя `printf()`.

В этом примере также используется инструкция `with`. Это означает, что для выполнения сценария необходимо иметь по крайней мере версию 2.6. Вернее, это не совсем так. Как уже было сказано выше, инструкция `with` была впервые введена в версии 2.5 в качестве экспериментальной. Иными словами, ее можно использовать и в этой версии, но включить следующую дополнительную инструкцию: `from _future_ import with_statement`. Но если вы все еще применяете версию 2.4 или одну из

предыдущих версий, то не будете иметь доступа к импорту указанной инструкции и вам придется использовать пример 1.1.

Пример 1.3. Универсальная версия сценария `rewho.py` (`rewhoU.py`)

Этот сценарий работает в обеих версиях, Python 2 и Python 3, поскольку в нем вместо инструкции `print` или функции `print()` применяется более простая функция, рассчитанная на обе версии. Кроме того, этот сценарий включает инструкцию `with`, которая впервые появилась в Python 2.5.

```
1  #!/usr/bin/env python
2
3  import os
4  from distutils.log import warn as printf
5  import re
6
7  with os.popen('who', 'r') as f:
8  for eachLine in f:
9  printf(re.split(r'\s+|\t', eachLine.strip()))
```

Создание сценария `rewhoU.py` представляет собой один из примеров того, как можно разработать универсальный сценарий, который позволяет избежать необходимости поддерживать два варианта одного и того же сценария для Python 2 и для Python 3.

Вызов на выполнение этого сценария в любом из интерпретаторов приведет к получению откорректированного и более удобного вывода:

```
$ rewho.py
['wesley', 'console', 'Feb 22 14:12']
['wesley', 'ttys000', 'Feb 22 14:18']
['wesley', 'ttys001', 'Feb 22 14:49']
['wesley', 'ttys002', 'Feb 25 00:13', '(192.168.0.20)']
['wesley', 'ttys003', 'Feb 24 23:49', '(192.168.0.20)']
```

Кроме того, следует помнить, что функция `re.split()` принимает также необязательный параметр с определением флагов, описанный ранее в этой главе.

Аналогичный пример можно реализовать и на компьютерах с операционной системой Windows, используя команду `tasklist` вместо `who`. Вывод этой команды приведен ниже.

```
C:\WINDOWS\system32>tasklist
```

Image Name	PID	Session Name	Session#	Mem Usage
System Idle Process	0	Console	0	28 K
System	4	Console	0	240 K
smss.exe	708	Console	0	420 K
csrss.exe	764	Console	0	4,876 K
winlogon.exe	788	Console	0	3,268 K
services.exe	836	Console	0	3,932 K
...				

Вполне очевидно, что в этом выводе содержится другая информация по сравнению с выводом команды `who`, но формат его аналогичен, поэтому остается возможность снова применить выработанное ранее решение, проводя разбиение с помощью

`re.split()` по одному или большему количеству пробелов (в данном случае проблема со знаками табуляции не возникает).

Но приходится сталкиваться с другой проблемой, связанной с тем, что в именах команд могут быть пробелы, а в сформированных результатах такие имена должны быть представлены без разбиения на отдельные части, разделенные пробелами. Это же относится к информации об использовании памяти, которая представлена в формате "NNN K", где NNN — объем памяти, а K — обозначение единицы измерения объема (килобайты). Эти компоненты данных также не следует отделять друг от друга, поэтому предположим, что лучше, например, удалять в таких фрагментах строк по меньшей мере один пробел.

Однако это не позволяет справиться со всей ситуацией. Приходится учитывать, что столбцы с идентификатором процесса (Process ID — PID) и именем сеанса (Session Name) разграничены только одним пробелом. Это означает, что после исключения по указанной выше причине по меньшей мере одного пробела информация об идентификаторе процесса и имени сеанса будет представлена без разделения, как один результат. Предположим, что решено скопировать один из предыдущих сценариев, назвать его `retasklist.py`, заменить команду `who` на команду `tasklist /nh` (опция `/nh` подавляет заголовки столбцов) и применить регулярное выражение `\s\s+`. Это приведет к получению вывода, который выглядит следующим образом:

```
z:\corepython\chl>python retasklist.py
['']
['System Idle Process', '0 Console', '0', '28 K']
['System', '4 Console', '0', '240 K']
['smss.exe', '708 Console', '0', '420 K']
['csrss.exe', '764 Console', '0', '5,028 K']
['winlogon.exe', '788 Console', '0', '3,284 K']
['services.exe', '836 Console', '0', '3,924 K']
...
```

Эти результаты показывают, что действительно удалось представить столбцы с именами команд и объемами памяти отдельно, без их разбиения, но, кроме того, непреднамеренно произошло соединение содержимого столбцов PID и Session Name. Это означает, что нужно отказаться от функции `split` и прибегнуть к использованию только сопоставления с помощью регулярного выражения. Реализуем этот замысел и вначале исключим содержимое столбцов Session Name и Number, поскольку это содержимое не требуется в итоговых результатах. В примере 1.4 показан заключительный вариант сценария `retasklist.py` для версии Python 2.

Пример 1.4. Обработка вывода команды DOS `tasklist` (`retasklist.py`)

В этом сценарии используются регулярное выражение и функция `findall()` для интерпретации вывода команды DOS `tasklist` в целях получения только интересующих нас данных. Для переноса этого сценария в версию Python 3 достаточно лишь перейти к использованию функции `print()`.

```
1  #!/usr/bin/env python
2
3  import os
4  import re
5
6  f = os.popen('tasklist /nh', 'r')
7  for eachLine in f:
```

```

8  print re.findall(
9      r'([\\w.]+(?: [\\w.]+)*)\\s\\s+(\\d+) \\w+\\s\\s+\\d+\\s\\s+([\\d,]+ K)',
10     eachLine.rstrip())
11  f.close()

```

После выполнения сценария формируется желаемый (усеченный) вывод:

```

Z:\corepython\chl>python retasklist.py
[]
[('System Idle Process', '0', '28 K')]
[('System', '4', '240 K')]
[('smss.exe', '708', '420 K')]
[('csrss.exe', '764', '5,016 K')]
[('winlogon.exe', '788', '3,284 K')]
[('services.exe', '836', '3,932 K')]
...

```

Тщательно составленное регулярное выражение позволяет пройти все пять столбцов выходной строки, группируя только те значения, которые требуются: имя команды, ее идентификатор процесса и объем занимаемой памяти. При этом используется много средств создания регулярных выражений, описанных выше в этой главе.

Эта книга написана с учебной целью, поэтому все сценарии, которые рассматривались в данной главе, были предназначены исключительно в целях отображения полученного вывода для последующего просмотра пользователем. На практике чаще всего вместо этого применяется другой подход: обработка полученных данных, сохранение их в базе данных, использование полученного вывода в целях формирования отчетов для вышестоящих руководителей и т.д.

1.5. Более сложный пример регулярного выражения

Теперь рассмотрим более сложный пример, в котором реализованы различные способы использования регулярных выражений для манипулирования строками. На первом этапе подготовим некоторый код, который предназначен для формирования случайных данных, с которыми можно будет в дальнейшем работать (при этом в нашу задачу не входит получение случайных данных с помощью каких-либо сложных алгоритмов). В примере 1.5 представлен сценарий `gendata.py`, предназначенный для формирования необходимого набора данных. В данном случае программа просто отображает сформированный набор строк на стандартном устройстве вывода, но ее выходные данные вполне могут быть перенаправлены в тестовый файл.

Пример 1.5. Генератор данных для примера регулярного выражения (`gendata.py`)

Этот сценарий создает случайные данные, на которых можно поупражняться в создании регулярных выражений, и выводит сформированные данные на экран. Для того чтобы перенести этот сценарий в версию Python 3, достаточно просто заменить инструкцию `print` функцией, переключиться с функции `xrange()` снова на функцию `range()` и вместо `sys.maxint` воспользоваться `sys.maxsize`.

```

1  #!/usr/bin/env python
2
3  from random import randrange, choice
4  from string import ascii_lowercase as lc
5  from sys import maxint
6  from time import ctime
7
8  tlds = ('com', 'edu', 'net', 'org', 'gov')
9
10 for i in xrange(randrange(5, 11)):
11     dtint = randrange(maxint)          # выборка даты
12     dtstr = ctime(dtint)               # строка даты
13     llen = randrange(4, 8)            # имя входа короче
14     login = ''.join(choice(lc) for j in range(llen))
15     dlen = randrange(llen, 13)         # имя домена длиннее
16     dom = ''.join(choice(lc) for j in xrange(dlen))
17     print '%s::%s@%s.%s::%d-%d-%d' % (dtstr, login,
18         dom, choice(tlds), dtint, llen, dlen)

```

В этом сценарии формируются строки с тремя полями, разграниченными парой двоеточий (т.е. двойным двоеточием). Первое поле содержит случайное (32-разрядное) целое число, которое преобразовано в дату. Следующее поле — сформированный случайным образом адрес электронной почты, а последнее поле — ряд целых чисел, разделенных одинарным тире (-).

При выполнении этого кода сформировался следующий вывод (очевидно, что при следующем прогоне он станет другим), после чего полученные данные были сохранены на локальном компьютере в файле `redata.txt`:

```

Thu Jul 22 19:21:19 2004::izsp@dicqdhytvhv.edu::1090549279-4-11
Sun Jul 13 22:42:11 2008::zgeu@dxaijbjgkniy.com::1216014131-4-11
Sat May 5 16:36:23 1990::fclihw@alwdbzpsdg.edu::641950583-6-10
Thu Feb 15 17:46:04 2007::uzifzf@fpyivihw.gov::1171590364-6-8
Thu Jun 26 19:08:59 2036::ugxfugt@jkhughs.net::2098145339-7-7
Tue Apr 10 01:04:45 2012::zkwaq@rpxwmtikse.com::1334045085-5-10

```

Читатель может иметь другое мнение, но вывод этой программы вполне подходит для обработки с помощью регулярного выражения. Ниже приведено построчное объяснение, которое позволяет понять, как реализовано несколько регулярных выражений для обработки этих данных, а также подготовиться к выполнению некоторых упражнений, приведенных в конце главы.

Построчное объяснение

Строки 1-6

В рассматриваемом примере сценария потребовалось использование нескольких модулей. Безусловно, мы не рекомендуем использовать инструкцию `from-import` по многим причинам (в частности, без нее проще определить, из какого модуля получена функция, устранить возможный конфликт модулей в сценарии и т.д.), но в данном примере было решено импортировать только конкретные атрибуты из необходимых модулей, чтобы читатель мог сосредоточиться только на этих атрибутах, а также, чтобы сократить объем кода.

Строка 8

Здесь `tlds` — просто ряд доменных имен высокого уровня, из которых случайным образом происходит выборка для формирования примеров адресов электронной почты.

Строки 10-12

При каждом выполнении сценария `gendata.py` формируется от 5 до 10 строк вывода. (В сценарии во всех случаях, в которых требуется получение случайного целого числа, используется функция `random.randrange()`.) Для каждой строки выбирается случайное целое число из всего возможного диапазона случайных чисел (от 0 до $2^{31}-1$ [`sys.maxint`]), которое затем преобразуется в значение даты с помощью функции `time.ctime()`. Системное время в языке Python и в большинстве компьютеров с операционной системой, соответствующей стандарту POSIX, определяется как количество секунд, истекших с начала “эпохи”, или с полуночи 1 января 1970 года по времени UTC/GMT. Для того чтобы выбрать 32-разрядное целое число, которое представляет наиболее отдаленный момент времени от начала эпохи, необходимо отсчитать 2^{32} секунды от начала эпохи.

Строки 13-16

Регистрационное имя для фиктивного адреса электронной почты должно иметь длину от 4 до 7 символов (это означает, что требуется вызов функции `randrange(4, 8)`). Иными словами, происходит случайный выбор от 4 до 7 строчных букв с последующей конкатенацией одной за другой букв с формируемой строкой. Функция `random.choice()` по своему назначению служит для получения последовательности, а затем возврата элемента этой последовательности, выбранного случайным образом. В данном случае последовательность представляет собой множество из всех 26 строчных букв английского алфавита, которое представлено в виде строки `ascii_lowercase`.

Было решено, что основное доменное имя для фиктивного адреса электронной почты не должно превышать по длине 12 символов, но не быть короче регистрационного имени. В этом случае снова используются случайно выбранные строчные буквы для сборки имени буква за буквой.

Строки 17-18

Ключевым компонентом сценария является тот фрагмент кода, в котором все случайно выбранные данные соединяются в одну выходную строку. Вначале следует строка даты, а за ней находится разделитель. После этого происходит сборка сформированного случайным образом адреса электронной почты путем конкатенации регистрационного имени, знака “@”, доменного имени и выбранного случайно домена высокого уровня. Вслед за заключительным двойным двоеточием добавляется случайно сформированная строка с целочисленным значением, в которой используется первоначально выбранное время (для строки даты), а далее располагаются имя входа и имя домена, разделенные одним дефисом.

1.5.1. Сопоставление со строкой

По условиям следующих упражнений создайте по две версии регулярных выражений: менее и более строгие. Рекомендуем проверять эти регулярные выражения в коротком приложении, в котором используется пример файла `redata.txt`,

представленный ранее (или с использованием своих собственных данных, сформированных в результате выполнения сценария `gendata.py`). Этот файл еще не раз потребуется при выполнении упражнений.

Для того чтобы проверить регулярное выражение, прежде чем ввести его в состав своего небольшого приложения, мы импортируем модуль `re` и присвоим в качестве примера одну строку из файла `redata.txt` строковой переменной, которая используется для представления входных данных. Эти инструкции являются одинаковыми в обоих рассматриваемых примерах.

```
>>> import re
>>> data = 'Thu Feb 15 17:46:04 2007::uzifzf@dpviviwh.gov::1171590364-6-8'
In our first example, we will create a regular expression to extract (only) the days
of the week from the timestamps from each line of the data file redata.txt. We will use
the following regex:
```

```
"^Mon|^Tue|^Wed|^Thu|^Fri|^Sat|^Sun"
```

В данном примере сформулировано требование, чтобы строка начиналась с любой из семи перечисленных строк (оператор регулярного выражения `"^"`). Простыми словами можно передать смысл этого регулярного выражения примерно так: строка должна начинаться с "Mon", "Tue", ..., "Sat" или "Sun".

Еще один вариант состоит в том, что можно заменить отдельные операторы вставки одним знаком вставки, сгруппировав строки с сокращенными обозначениями дней недели следующим образом:

```
"^(Mon|Tue|Wed|Thu|Fri|Sat|Sun) "
```

Набор строк заключен в круглые скобки, а это означает, что должна быть обнаружена одна из этих строк для того, чтобы сопоставление было выполнено успешно. Это более удобная версия исходного регулярного выражения, с которого началась разработка, не имеющего круглых скобок. Новая версия регулярного выражения позволяет воспользоваться тем, что теперь можно получить доступ к сопоставленной строке как к подгруппе:

```
>>> patt = '^ (Mon|Tue|Wed|Thu|Fri|Sat|Sun) '
>>> m = re.match(patt, data)
>>> m.group()      # полное соответствие
'Thu'
>>> m.group(1)     # подгруппа 1
'Thu'
>>> m.groups()     # все подгруппы
('Thu',)
```

На первый взгляд это изменение не кажется столь существенным по сравнению с тем, которое было показано в начале данного упражнения, но его преимущество проявится в следующем примере (как и в аналогичных примерах), в котором должны быть предоставлены дополнительные данные в составе регулярного выражения, упрощающие процесс согласования строки, пусть даже эти символьные данные не составляют часть интересующей нас строки.

Однако оба варианта рассматриваемого регулярного выражения являются весьма строгими, поскольку в них конкретно перечислены элементы набора строк. Такое регулярное выражение может стать неприемлемым в среде с поддержкой нескольких национальных языков, в которой используются локализованные полные и

сокращенные названия дней недели. Менее строгое регулярное выражение могло бы выглядеть следующим образом: `^\w{3}`. Оно требует лишь того, чтобы строка началась с трех подряд идущих алфавитно-цифровых символов. В этом случае можно описать смысл регулярного выражения примерно так, что знак вставки указывает на начало строки, знак `\w` обозначает любой отдельно взятый алфавитно-цифровой символ, а конструкция `{3}` показывает, что в строке должны присутствовать 3 подряд идущие копии регулярного выражения, за которым следует `{3}`. При этом, если потребуется группирование, можно также использовать круглые скобки следующим образом, `^(\w{3})`:

```
>>> patt = '^(\w{3})'
>>> m = re.match(patt, data)
>>> if m is not None: m.group()
...
'Thu'
>>> m.group(1)
'Thu'
```

Однако необходимо отметить, что регулярное выражение `^(\w){3}` является неправильным. Если конструкция `{3}` находится в круглых скобках, то вначале происходит сопоставление с тремя подряд идущими алфавитно-цифровыми символами, а затем полученный результат представляется в виде группы. Но после перемещения конструкции `{3}` за пределы круглых скобок полученный результат становится эквивалентным трем отдельным последовательным алфавитно-цифровым символам:

```
>>> patt = '^(\w){3}'
>>> m = re.match(patt, data)
>>> if m is not None: m.group()
...
'Thu'
>>> m.group(1)
'u'
```

Причина, по которой при обращении к подгруппе 1 обнаруживается только буква "u", состоит в том, что в ходе обработки регулярного выражения содержимое подгруппы 1 последовательно заменялось очередным символом. Иными словами, вначале значением `m.group(1)` было "t", затем оно изменилось на "h" и, наконец, стало равным "u". Это — три отдельные (и перекрывающиеся) группы, состоящие каждая из одного алфавитно-цифрового символа, а не одна группа, которая состоит из трех последовательных алфавитно-цифровых символов.

В следующем (и заключительном) примере будет создано регулярное выражение, предназначенное для извлечения числовых полей, находящихся в конце каждой строки файла `redata.txt`.

1.5.2. Сравнение поиска и сопоставления с учетом жадных выражений

Прежде чем приступить к созданию регулярного выражения для выделения указанного поля, рассмотрим, как извлечь числовое значение, находящееся в конце строки данных. Если возникает необходимость обеспечить выборку данных, находящихся в начале или в конце строки, то появляется выбор — использовать для этой цели поиск или сопоставление. Более оправданным является применение поиска, поскольку

точно известно следующее: искомыми данными является ряд из трех целочисленных знаков, эти данные не находятся в начале строки и не составляют всю строку. Если бы было решено использовать сопоставление, то пришлось бы создать регулярное выражение для сопоставления со всей строкой и задать подгруппы для сохранения интересующих нас данных. Для иллюстрации различий между указанными подходами вначале будет организован поиск, а затем сопоставление. Это позволит понять, что в данном случае более удобен поиск.

Необходимо найти три целочисленных знака, разграниченных дефисами, поэтому требуемое регулярное выражение принимает следующий вид: `\d+--\d+--\d+`. Это регулярное выражение можно описать так: "Любое количество цифр (но не меньше одного), за которым следует дефис, затем еще один ряд цифр, еще один дефис и, наконец, последний ряд цифр". Проверим это регулярное выражение, на сей раз с использованием функции `search()`:

```
>>> patt = '\d+--\d+--\d+'
>>> re.search(patt, data).group()      # полное совпадение
'1171590364-6-8'
```

Однако попытка сопоставления окончилась бы неудачей. Рассмотрим, с чем это связано. Сопоставление начинается с начала строки, а искомые числовые строки находятся в конце строки. Должно быть создано другое регулярное выражение, которое сопоставлялось бы со всей строкой. Тем не менее, можно попытаться сэкономить силы, применив конструкцию `.` для указания на присутствие любого произвольного набора символов, за которым следуют те символы, которые нас интересуют:

```
patt = '.*\d+--\d+--\d+'
>>> re.match(patt, data).group()      # полное совпадение
'Thu Feb 15 17:46:04 2007::uzifzf@dpyivihw.gov::1171590364-6-8'
```

Этот замысел оказался удачным, но фактически в данном случае интерес представляет числовое поле, находящееся в конце строки, а не вся строка, поэтому необходимо применить круглые скобки для группирования той части строки, которая нас интересует:

```
>>> patt = '.*(\d+--\d+--\d+)'
>>> re.match(patt, data).group(1)     # подгруппа 1
'4-6-8'
```

Однако требуемые результаты не достигнуты. Необходимо было извлечь `1171590364-6-8`, а не просто `4-6-8`. Почему же первая цифровая строка оказалась урезанной? Проблема заключается в том, что регулярные выражения по самой своей сути являются *жадными*. Это означает, что регулярные выражения с шаблонами подстановочных знаков вычисляются слева направо и при этом предпринимается попытка "захватить" как можно больше символов, сопоставляемых с шаблоном. При использовании приведенного выше регулярного выражения конструкция `.` успешно сопоставлялась с каждым отдельным символом с начала строки, в том числе с большей частью первого целочисленного поля, которое нас интересует. Для успешного сопоставления с конструкцией `\d+` достаточно оставить нетронутой лишь одну цифру, поэтому с ее помощью и была получена цифра "4", а конструкция `.` была сопоставлена со всей начальной частью строки вплоть до указанной первой цифры: "Thu Feb 15 17:46:04 2007::uzifzf@dpyivihw.gov::117159036", как показано на рис. 1.2.

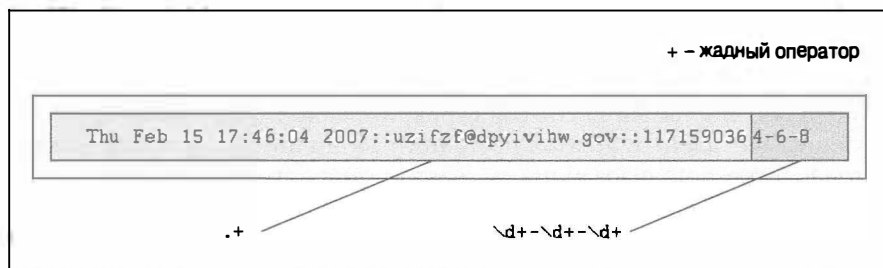


Рис. 1.2. Причина того, что с помощью регулярного выражения не удалось выполнить поставленную задачу: оператор + является жадным

Одно из решений состоит в использовании оператора, указывающего, что предыдущая конструкция "не должна быть жадной": ?. Этот оператор можно использовать после операторов *, + или ?. Оператор *подавления жадного поведения* указывает обработчику регулярных выражений, что стоящая перед ним часть шаблона должна быть сопоставлена с минимально возможным количеством символов. Поэтому желаемый результат может быть получен после размещения оператора ? после конструкции .+, как показано на рис. 1.3.

```
>>> patt = '.*?(\\d+-\\d+-\\d+)'
>>> re.match(patt, data).group(1)    # подгруппа 1
'1171590364-6-8'
```

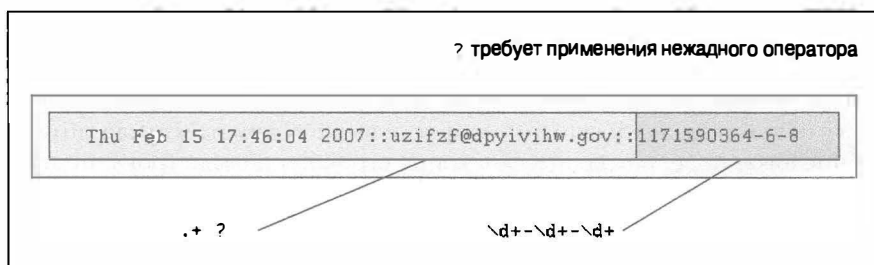


Рис. 1.3. Решение проблемы жадного поведения: применение нежадного запроса на основе знака ?

В данном случае можно применить более простое решение — обеспечить распознавание знаков ":", которые служат в качестве разделителя полей. После этого остается лишь воспользоваться обычным строковым методом strip(':') для получения всех частей строки, а затем выполнить еще одно разбиение с помощью strip('-') по дефису для получения трех цифр, поиск которых был обусловлен в исходном задании. Следует отметить, что данное решение могло быть выбрано с самого начала, но это не было сделано, поскольку применяемый в нем процесс является всего лишь обратным тому, с помощью которого производилось первоначальное формирование тестовых строк с применением сценария gendata.py!

Рассмотрим еще один, заключительный пример. Предположим, что необходимо извлечь только среднюю цифру из поля, состоящего из трех цифр. Ниже приведен один из вариантов решения этой задачи (в котором используется поиск, чтобы можно было отказаться от необходимости сопоставления со всей строкой): - (\\d+) -. Проверка этого шаблона приводит к следующему:

```
>>> patt = '-(\d+)-'  
>>> m = re.search(patt, data)  
>>> m.group()      # полное совпадение  
'-6-'  
>>> m.group(1)     # подгруппа 1  
'6'
```

В этой главе едва ли удалось показать всю мощь регулярных выражений, и ее ограниченный объем не позволил воздать им должное, но мы надеемся, что читатель мог ознакомиться с достаточно информативным введением, чтобы иметь возможность включить это мощное инструментальное средство в свой арсенал приемов программирования. Предлагаем читателю обратиться к документации за более подробными сведениями о том, как использовать регулярные выражения в сценариях на языке Python. Для более глубокого изучения регулярных выражений рекомендуем книгу *Jeffrey E. F. Friedl, Mastering Regular Expressions*.

1.6. Упражнения

Регулярные выражения. Создать регулярные выражения по условиям упражнений 1.1–1.12, которые указаны ниже.

- 1.1. Распознать следующие строки: "bat", "bit", "but", "hat", "hit" или "hut".
- 1.2. Обеспечить сопоставление с любой парой слов, разделенной одним пробелом, такой как имя и фамилия.
- 1.3. Обеспечить сопоставление с любым словом и одной буквой, разделенными запятой и одним пробелом, как принято в США в написании фамилии и первого инициала.
- 1.4. Обеспечить сопоставление со всеми допустимыми идентификаторами Python.
- 1.5. Обеспечить сопоставление с частью почтового адреса, обозначающей улицу, согласно формату, принятому в конкретной стране (регулярное выражение должно быть достаточно общим, чтобы обеспечить сопоставление с названиями улиц, состоящими из любого количества строк, включая тип улицы: проспект, переулок, бульвар и т.д.). Например, в США для обозначения улиц используется примерно такой формат: 1180 Bordeaux Drive. Регулярное выражение должно быть достаточно гибким, чтобы можно было распознавать название улиц, состоящие из нескольких слов, допустим, такие: 3120 De la Cruz Boulevard.
- 1.6. Обеспечить сопоставление с простыми доменными именами Интернета, которые начинаются с подстроки "www." и заканчиваются суффиксом ".com"; например `www.yahoo.com`. Дополнительная возможность более успешно выполнить задание состоит в следующем: если составленное регулярное выражение будет также поддерживать другие доменные имена высокого уровня, такие как `.edu`, `.net` и т.д. (например, `www.foothill.edu`).
- 1.7. Обеспечить сопоставление с множеством строковых представлений всех целочисленных значений, поддерживаемых языком Python.
- 1.8. Обеспечить сопоставление с множеством строковых представлений всех длинных целых чисел, поддерживаемых языком Python.

- 1.9. Обеспечить сопоставление с множеством строковых представлений всех чисел с плавающей точкой, поддерживаемых языком Python.
- 1.10. Обеспечить сопоставление с множеством строковых представлений всех комплексных чисел, поддерживаемых языком Python.
- 1.11. Обеспечить сопоставление с множеством всех допустимых адресов электронной почты (начните с наименее ограничительного регулярного выражения, а затем попытайтесь сделать его настолько строгим, насколько сможете, но все еще предоставляющим требуемые функциональные возможности).
- 1.12. Обеспечить сопоставление с множеством всех допустимых адресов веб-сайта (URL) (начните с наименее ограничительного регулярного выражения, а затем попытайтесь сделать его настолько строгим, насколько сможете, но все еще предоставляющим требуемые функциональные возможности).
- 1.13. Функция `type()`. Встроенная функция `type()` возвращает объект типа, который отображается примерно как следующая строка в формате, принятом в интерпретаторе Python:

```
>>> type(0)
<type 'int'>
>>> type(.34)
<type 'float'>
>>> type(dir)
<type 'builtin_function_or_method'>
```

Создать регулярное выражение, с помощью которого можно извлечь фактическое имя типа из строки. Разработанная функция должна принимать строку наподобие `<type 'int'>` и возвращать `int`. (То же относится к другим типам, таким как `'float'`, `'builtin_function_or_method'` и т.д.)

Примечание. Тем самым реализуется значение, хранимое в атрибуте `__name__` для классов и некоторых встроенных типов.

- 1.14. *Обработка дат.* В разделе 1.2 был представлен шаблон регулярного выражения, предназначенный для сопоставления с числовыми обозначениями месяцев года с января по сентябрь, которые состоят из одной или двух цифр, (`0?[1-9]`). Создайте регулярное выражение, которое представляет числовые обозначения последних трех месяцев в стандартном календаре.
- 1.15. *Обработка номеров кредитных карточек.* Кроме того, в разделе 1.2 рассматривался шаблон регулярного выражения, предназначенный для сопоставления с номерами кредитных карточек (CC), `[0-9]{15,16}`. Однако этот шаблон не позволяет выполнять сопоставление с номерами кредитных карточек, которые содержат дефисы, разделяющие блоки цифр. Создайте регулярное выражение, которое позволяет вставлять дефисы, но только в обусловленных местах. Например, 15-значные номера CC имеют шаблон 4-6-5, указывающий на применение формата “четыре цифры, дефис, шесть цифр, дефис, пять цифр”; а 16-значные номера CC имеют шаблон “четыре цифры, дефис, четыре цифры, дефис, четыре цифры, дефис, четыре цифры”. Не забудьте предусмотреть с помощью фигурных скобок проверку длины всей строки. **Дополнительное задание.** Существует стандартный алгоритм проверки того, является ли номер CC допустимым. Попробуйте написать

сценарий, который не только определяет, правильно ли отформатирован номер СС, но и проверяет его допустимость.

Экспериментирование со сценарием `gendata.py`. Следующий ряд упражнений (1.16–1.27) предназначен специально для работы с данными, формируемыми сценарием `gendata.py`. Прежде чем приступать к упражнениям 1.17 и 1.18, рекомендуется вначале выполнить упражнение 1.16 и составить все регулярные выражения.

- 1.16. Обновите код сценария `gendata.py` так, чтобы данные записывались непосредственно в файл `redata.txt`, а не выводились на экран.
- 1.17. Определите, сколько раз каждый день недели появляется во вновь созданном файле `redata.txt` после каждого вызова сценария. (Еще одна проверка может состоять в подсчете того, сколько раз был случайным образом выбран каждый месяц года.)
- 1.18. Убедитесь в том, что данные в файле `redata.txt` строго соответствуют формату, путем проверки совпадения первой цифры целочисленного поля с отметкой времени, заданной в начале каждой выходной строки.

Создайте регулярные выражения, которые выполняют следующие действия.

- 1.19. Извлечение полной отметки времени из каждой строки.
- 1.20. Извлечение полного адреса электронной почты из каждой строки.
- 1.21. Извлечение только значений месяцев из отметок времени.
- 1.22. Извлечение только значений лет из отметок времени.
- 1.23. Извлечение только значений времени (HH:MM:SS) из отметок времени.
- 1.24. Извлечение только имен входа и имен доменов (и имени основного домена, и имени домена высокого уровня вместе взятых) из адреса электронной почты.
- 1.25. Извлечение только имен входа и имен доменов (имени основного домена и имени домена высокого уровня) из адреса электронной почты.
- 1.26. Замена адреса электронной почты в каждой строке данных произвольно выбранным адресом электронной почты.
- 1.27. Извлечение значений месяцев, дней и лет из отметок времени и вывод их в формате “месяц, число, год”, лишь с однократной итерацией по каждой строке.

Обработка номеров телефонов. Исходя из условий упражнений 1.28 и 1.29, еще раз воспользуйтесь регулярным выражением, введенным в разделе 1.2, в котором сопоставляется номер телефона, но допускается также необязательный префикс кода города: `\d{3}-\d{3}-\d{4}`. Обновите это регулярное выражение с учетом следующих требований.

- 1.28. Код города (первый ряд из трех цифр и сопровождающий их дефис) является необязательным, т.е. регулярное выражение должно сопоставляться не только с 800-555-1212, но и с 555-1212.

- 1.29. Поддерживается код города, не только отделенный дефисом, но и заключенный в круглые скобки; при этом остается условие, что код города является необязательным. Обеспечьте сопоставление регулярного выражения, допустим, со следующими вариантами номеров: 800-555-1212, 555-1212 и (800) 555-1212.

Программы на основе регулярных выражений. В последнем ряде упражнений ставится задача подготовить полезные программные сценарии, предназначенные для обработки оперативных данных.

- 1.30. Формирование кода HTML. На основе списка ссылок (с необязательными короткими описаниями), который может быть предоставлен пользователям с помощью командной строки, введен из другого сценария или получен из базы данных, сформируйте веб-страницу (.html), которая включает все ссылки как точки привязки гипертекста и после развертывания в веб-браузере позволяет пользователю щелкать на этих ссылках и посещать соответствующие сайты. Если предоставлено короткое описание, то в качестве гипертекста должно применяться это описание вместо URL.
- 1.31. Извлечение текста из сообщения Twitter (твита). Иногда возникает необходимость извлечь из твита, отправленного пользователем с помощью службы Twitter, только сам текст, который был отправлен. Создайте функцию, которая принимает на входе твит и необязательный флаг "meta", имеющий по умолчанию значение False, а затем возвращает строку вычищенного твита, в котором удалена вся посторонняя информация, такая как обозначения "RT" команд "retweet", ведущая точка (.) и все теги "#hashtags". Если флаг meta имеет значение True, то должен быть также возвращен словарь, содержащий метаданные. Возвращенные данные могут содержать ключ "RT", значением которого является кортеж строк, полученных от пользователей, переславших сообщение твита, и/или ключ "hashtags" с кортежем тегов hashtags. Если значения не существуют (кортежи пусты), то для них не следует создавать записи "ключ-значение".
- 1.32. Сценарий обработки содержимого экрана Amazon. Создайте сценарий, который мог бы помочь тем, кто следит за появлением своих любимых книг и за тем, как они попали на сайт Amazon (или на любой другой сайт, предназначенный для продажи книг через Интернет, на котором отслеживаются рейтинги книг). Например, на сайте Amazon на любую книгу указывает ссылка в формате `http://amazon.com/dp/ISBN` (скажем, `http://amazon.com/dp/0132678209`). После этого в сценарии может быть предусмотрена возможность изменять имена доменов для перехода на сайты Amazon в других странах, таких как Германия (.de), Франция (.fr), Япония (.jp), Китай (.cn) и Великобритания (.co.uk), для ознакомления с соответствующими рейтингами. Для извлечения данных о рейтингах используйте регулярные выражения или средства синтаксического анализа разметки, такие как BeautifulSoup, lxml или html5lib. После этого предоставьте пользователю возможность передать параметр командной строки с указанием того, должен ли вывод быть представлен в виде обычного текста, допустим, для включения в текст сообщения электронной почты, или отформатирован в коде HTML для просмотра на веб-сайте.

ГЛАВА

2

Сетевое программирование

В этой главе...

- Введение
- Что такое архитектура “клиент–сервер”
- Сокеты: конечные точки связи
- Сетевое программирование на языке Python
- Модуль SocketServer
- Введение в концепцию Twisted
- Связанные модули

Итак, IPv6. Как известно, пространство адресов IPv4 почти полностью исчерпано. В связи с этим я испытываю определенную неловкость, поскольку именно я — тот человек, который решил, что 32-разрядного адреса достаточно для эксперимента с Интернетом. Моим единственным оправданием может служить то, что этот выбор был сделан в 1977 году, и я действительно думал, что это эксперимент. Проблема состоит в том, что эксперимент не закончился, поэтому мы имеем то, что имеем.

Винт Серф (Vint Cerf), январь 2011 г.¹

(из выступления на конференции linux.conf.au)

2.1. Введение

В этом разделе кратко рассматривается сетевое программирование с использованием сокетов. В прежде чем углубляться в эту тему, необходимо вспомнить основы сетевого программирования и описать, как сокет применяется в языке Python; после этого мы покажем, как использовать некоторые модули Python для создания сетевых приложений.

2.2. Что такое архитектура “клиент–сервер”

Понятие архитектуры “клиент–сервер” для разных людей имеет различный смысл, в зависимости от их специализации и от того, идет ли речь о программном обеспечении или о системе аппаратных средств. В любом случае определение этого понятия является довольно простым: сервер (аппаратное устройство или программное обеспечение) предоставляет “услуги”, которые требуются одному или нескольким клиентам (пользователям услуг). Единственным предназначением сервера является ожидание запросов (клиентов), возвращение ответов на них (предоставление услуги) и ожидание следующих запросов.

Клиенты, с другой стороны, обращаются к серверу с конкретным запросом, отправляют все необходимые данные, а затем ожидают ответа сервера, который может либо предоставлять затребованное в запросе, либо содержать указание на причину отказа. Сервер работает неопределенно долго, непрерывно обрабатывая запросы; клиенты выполняют одноразовые запросы для получения услуги, получают эту услугу, после чего завершают текущую операцию. Клиент может в последующем выполнять дополнительные запросы, но они рассматриваются в рамках отдельных операций.

Наиболее широко применяемое в наше время определение понятия архитектуры “клиент–сервер” иллюстрируется на рис. 2.1, где показан пользовательский, или клиентский, компьютер, с помощью которого происходит получение информации от сервера через Интернет. Хотя такая система действительно может служить примером архитектуры “клиент–сервер”, это не единственный вариант данной архитектуры. Кроме того, архитектура “клиент–сервер” может быть реализована не только с помощью программного обеспечения, но и с применением компьютерных аппаратных средств.

¹

Впервые об этом было сказано не позже 2004 г.; см. <http://www.educause.edu/EDUCAUSE+Review/EDUCAUSEReviewMagazineVolume39/MusingsontheInternetPart2/157899>

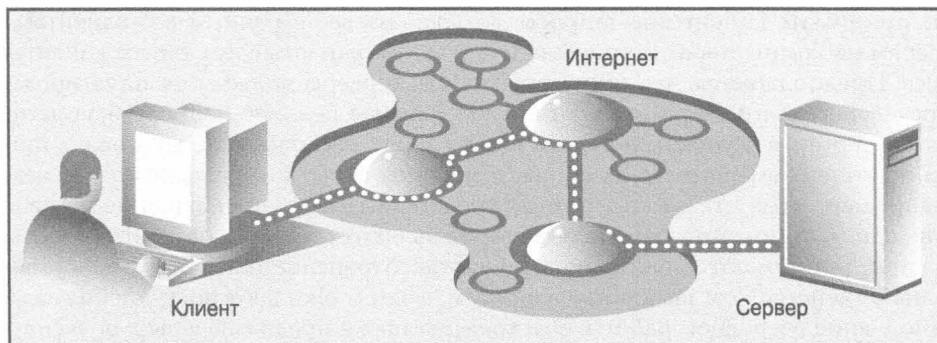


Рис. 2.1. Типичный вариант реализации концепции системы "клиент-сервер" в Интернете

2.2.1. Аппаратная архитектура "клиент-сервер"

Примерами аппаратных серверов могут служить серверы печати. Эти серверы обрабатывают входящие задания на печать и отправляют их на принтер (или на какое-то другое устройство печати), подключенный к такой системе. Как правило, доступ к серверу печати предоставляется по сети, и клиентские компьютеры отправляют на него запросы печати.

Еще одним примером аппаратного сервера может служить файловый сервер. Файловые серверы, как правило, представляют собой компьютеры с большим объемом запоминающих устройств общего назначения, к которым предоставляется дистанционный доступ для клиентов. Клиентские компьютеры монтируют диски серверного компьютера так, как если бы эти диски находились на локальном компьютере. Одной из наиболее широко применяемых сетевых операционных систем, которые поддерживают файловые серверы, является NFS (Network File System — сетевая файловая система) компании Sun Microsystems. Если доступ к сетевому дисковому накопителю организован так, что невозможно определить, установлен ли диск на локальном компьютере или находится в сети, это означает, что система "клиент-сервер" успешно выполняет свою работу. Назначение такой системы состоит в том, чтобы пользователь в своей работе не ощущал разницы между локальным и сетевым диском. Такое функционирование системы доступа обеспечивается с помощью программной реализации.

2.2.2. Программная архитектура "клиент-сервер"

Программные серверы также эксплуатируются на аппаратных средствах, но, в отличие от аппаратных серверов, не имеют выделенных периферийных устройств (принтеров, дисковых накопителей и т.д.). К основным услугам, предоставляемым программными серверами, относятся выполнение программ, поиск и передача данных, агрегирование, обновление данных, а также осуществление других типов запрограммированных действий или операций манипулирования данными.

В наши дни к числу наиболее широко применяемых программных серверов относятся веб-серверы. Частные лица или компании, желающие эксплуатировать собственные веб-серверы, должны приобрести один или несколько компьютеров, подготовить веб-страницы или создать веб-приложения, которые они желают предоставить пользователям, а затем запустить веб-сервер. Назначение такого сервера состоит в том,

чтобы принимать клиентские запросы, возвращать веб-страницы веб-клиентам, т.е. браузерам на компьютерах пользователей, а затем ожидать следующего клиентского запроса. Предполагается, что после запуска эти серверы должны эксплуатироваться неопределенно долго. Разумеется, это недостижимая цель, но непрерывная эксплуатация веб-серверов осуществляется настолько долго, насколько это возможно, при условии отсутствия вмешательства какой-то внешней силы, которая вольно или невольно (например, из-за отказа аппаратных средств) вызывает прекращение их работы.

Еще одним типом программных серверов являются серверы баз данных. Серверы баз данных принимают клиентские запросы на сохранение или поиск информации, выполняют действия согласно этим запросам, а затем ожидают поступления задания на выполнение очередной работы. Эти серверы также предназначены для эксплуатации в течение неопределенно долгого времени.

Последним типом рассматриваемого нами программного сервера является оконный сервер. Такие серверы могут рассматриваться почти как аналогичные аппаратным серверам. Они эксплуатируются на компьютере с подключенным устройством отображения, например, монитором того или иного типа. Клиентами оконного сервера являются программы, для работы которых требуется оконная среда. Такие клиенты принято рассматривать как приложения с графическим интерфейсом пользователя (graphical user interface — GUI). Попытка вызова подобных приложений без оконного сервера, иными словами, в среде поддержки текстового режима, такой как окно DOS или командный интерпретатор Unix, приводит к неудачному завершению. После получения доступа к оконному серверу приложения с графическим интерфейсом функционируют нормально.

Такая среда становится еще более интересной, если применяется наряду с сетевыми средствами. Обычно в качестве дисплея для оконного клиента служит сервер на локальном компьютере, но в некоторых сетевых средах поддержки окон, таких как система X Window, для вывода отображения из графического приложения можно выбрать оконный сервер другого компьютера. В таких ситуациях программа с графическим интерфейсом пользователя может эксплуатироваться на одном компьютере, а выводить изображение на другом!

2.2.3. Кассир банка как пример сервера

Для того чтобы лучше понять, как работает архитектура “клиент-сервер”, достаточно представить себе, допустим, кассира банка, который не ест, не спит, не отдыхает, а обслуживает одного клиента за другим из очереди, которая кажется бесконечной (рис. 2.2). Очередь может достигать большой длины или даже иногда полностью отсутствовать, но в любой момент в ней может появиться новый клиент. Разумеется, в прошлые годы о таком кассире можно было только мечтать, но теперь нашли широкое распространение банкоматы, которые, по-видимому, весьма близки по своему поведению к такой модели.

Очевидно, что такой кассир подходит под определение сервера, который работает в бесконечном цикле. Каждый объект, требующий обслуживания, рассматривается как клиент, требования которого должны быть выполнены. Клиенты поступают на обслуживание к кассиру и обслуживаются им в порядке очереди. После завершения операций с текущим клиентом этот клиент уходит, а сервер приступает к работе со следующим клиентом или переходит в состояние ожидания до прибытия клиента.

Мы уделяем так много внимания этому описанию потому, что рассматриваемый способ организации работы наглядно показывает общий принцип действия

архитектуры “клиент–сервер”. Теперь, после описания основ данного подхода, попытаемся применить его в сетевом программировании, следуя программной модели архитектуры “клиент–сервер”.

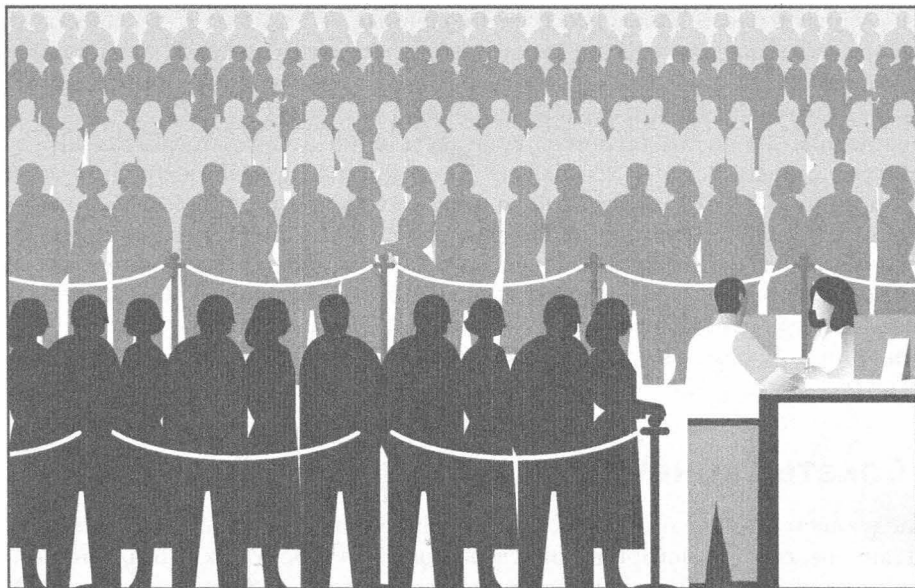


Рис. 2.2. На рисунке показан кассир банка, который работает без каких-либо перерывов, обслуживая клиентские запросы. Кассир работает в бесконечном цикле, получая запросы, обслуживая их, а затем возвращаясь в исходное состояние, чтобы обслужить очередного клиента или ожидать его появления. Очередь клиентов иногда может значительно увеличиваться, а иногда исчезать, но в любом случае кассир всегда должен быть готов к работе

2.2.4. Сетевое программирование по принципу “клиент–сервер”

Прежде чем сервер сможет отвечать на клиентские запросы, необходимо выполнить некоторые предварительные процедуры настройки, чтобы подготовить сервер к предстоящей работе. Должна быть создана *конечная точка связи*, с помощью которой сервер будет принимать (“прослушивать”) поступающие запросы. В этой роли сервер можно сравнить с секретарем директора компании или телефонистом на коммутаторе, отвечающих на запросы, которые поступают по основному телефонному номеру компании. После получения номера телефона, установки оборудования и привлечения к работе оператора может начаться обслуживание.

Аналогичная подготовка должна быть проведена и в мире сетей: как только устанавливается конечная точка связи, сервер, прослушивающий входящие запросы, может войти в бесконечный цикл, ожидая подключения клиентов и отвечая на их запросы. Разумеется, мы не должны оставлять секретаря директора компании без дела, поэтому обязаны указать номер телефона компании на ее фирменном бланке, распространить в виде рекламы или сообщений для прессы и т.д. В противном случае телефон так никогда и не зазвонит!

Точно так же потенциальным клиентам необходимо сообщить, что существует сервер, способный удовлетворять их потребности. Если это не будет сделано, то на сервер не поступит ни один запрос. Представьте себе, что программист только что создал совершенно новый веб-сайт. Это может быть самый потрясающий, поразительный, удивительный, полезный и великолепный веб-сайт из всех существующих, но если его веб-адрес или URL не будет распространен в средствах массовой информации и не появится каким-то образом в рекламе, то о нем никто и никогда не узнает, поэтому сайт так и не примет посетителей.

По-видимому, у читателя теперь есть достаточно полное представление о том, как работает сервер. Вступая в мир сетевого программирования, труднее всего понять именно это. Что же касается функционирования клиента, то его можно описать гораздо проще по сравнению с сервером. Задача клиента заключается лишь в том, что он должен создать свою отдельную конечную точку связи, а затем установить соединение с сервером. После этого клиент может выполнять запросы, в том числе осуществлять весь необходимый обмен данными. После обработки запроса или получения клиентом результата или просто необходимого подтверждения сеанс связи завершается.

2.3. Сокеты: конечные точки связи

В следующем разделе будет изложено вводное описание сокетов. В частности, мы представим некоторые исторические сведения об их происхождении, опишем различные типы сокетов и, наконец, покажем, как с помощью сокетов обеспечить обмен данными между процессами, работающим на разных компьютерах (или на одном и том же компьютере).

2.3.1. Общее определение понятия сокета

Сокет — это сетевая структура данных, реализующая понятие “конечной точки связи”, описанной в предыдущем разделе. Прежде чем установить связь, сетевые приложения сначала должны создать сокеты. Их можно сравнить с телефонными розетками, без подключения к которым невозможно войти в телефонную сеть.

Происхождение понятия сокетов можно проследить до 1970-х годов, когда они вошли в состав версии операционной системы Unix, выпущенной Калифорнийским университетом (Беркли), известной как BSD Unix. Поэтому иногда приходится слышать, как сокеты называют *сокетами Беркли* или *сокетами BSD*. Сокеты были первоначально созданы для приложений, работающих на одном и том же узле, где они должны были обеспечивать взаимодействие одной работающей программы (для обозначения которой принято использовать термин “процесс”) с другой работающей программой. Такой режим работы получил название *межпроцессного взаимодействия* (interprocess communication — IPC). Сокеты подразделяются на две разновидности: файловые и сетевые.

В первую очередь мы рассмотрим семейство сокетов Unix, которое обозначается именем AF_UNIX (в стандарте POSIX.1g наряду с этим применяется также обозначение AF_LOCAL). Как показывает аббревиатура AF (address family), под ним подразумевается *семейство адресов* UNIX. В наиболее широко применяемых программных платформах, включая Python, используется термин *семейство адресов* и в обозначении семейства применяется аббревиатура AF; в более старых системах вместо термина *семейство адресов* применяются термины *домен* или *семейство протоколов*

и обозначение *PF* (protocol family), а не *AF*. Предполагается, что обозначение семейства *AF_UNIX* должно быть заменено обозначением *AF_LOCAL* (стандартизированным в 2000-2001 гг.), но для обеспечения обратной совместимости во многих системах используются оба обозначения. При этом для одной и той же числовой константы, которая служит для внутреннего представления семейства в программах, определяются два псевдонима (*AF_UNIX* и *AF_LOCAL*). В самом языке Python все еще принято использовать обозначение *AF_UNIX*.

Сокеты этого семейства обеспечивают взаимодействие процессов, работающих на одном и том же компьютере, поэтому относятся к разновидности файловых сокетов. Это означает, что инфраструктура, лежащая в их основе, поддерживается файловой системой. Такая организация работы вполне оправдана, поскольку именно к файловой системе может быть предоставлен общий доступ различным процессам, работающим на одном и том же узле.

Сокеты второй разновидности являются сетевыми и имеют собственное семейство, *AF_INET*, которое принято также называть *семейством адресов Интернета*. Предусмотрено еще одно семейство адресов, *AF_INET6*, которое используется для адресации в протоколе Интернета версии 6 (IPv6). Можно встретить и другие семейства адресов, но все они не нашли широкого распространения, поскольку являются специализированными, устаревшими, редко используемыми или оставшимися нереализованными. В настоящее время из всех семейств адресов наиболее широко используется *AF_INET*.

2.5

Кроме того, в версии Python 2.5 была введена поддержка для специального типа сокетов Linux. Семейство сокетов *AF_NETLINK* (не предусматривающее установления логических соединений; см. раздел 2.3.3) обеспечивает межпроцессное взаимодействие между пользовательским кодом и кодом ядра с использованием стандартного интерфейса сокетов BSD. Ввод в действие этого семейства сокетов оказался более изящным и менее рискованным решением по обеспечению взаимодействия непривилегированного и привилегированного кода по сравнению с такими громоздкими решениями, как добавление новых системных вызовов, поддержка каталога */proc* или применение вызовов *"IOCTL"* операционной системы.

2.6

Еще одним средством для Linux (впервые реализованным в версии 2.6) явилась поддержка протокола прозрачного межпроцессного взаимодействия (Transparent Interprocess Communication — *TIPC*). Протокол *TIPC* используется для обеспечения обмена данными между кластерами компьютеров без использования адресации на основе IP. Поддержка протокола *TIPC* в языке Python предусмотрена в форме семейства *AF_TIPC*.

Вообще говоря, в языке Python поддерживаются только семейства *AF_UNIX*, *AF_NETLINK*, *AF_TIPC* и *AF_INET* (*AF_INET6*). В этой главе основное внимание уделено сетевому программированию, поэтому в большей ее части используется семейство *AF_INET*.

2.3.2. Адреса сокетов: пара "хост-порт"

Если сам сокет можно сравнить с телефонной розеткой (частью инфраструктуры, обеспечивающей связь), то имя хоста и номер порта подобны применяемым в сочетании коду города и номеру телефона. Наличие оборудования и возможности устанавливать связь не принесет никакой пользы, если не известно, с кем и как "связываться".

Адрес в Интернете представляет собой пару, состоящую из имени хоста и номера порта. Не зная оба этих компонента, невозможно установить связь по сети. Само собой разумеется, что должен также присутствовать некто, способный принять вызов на другом конце линии связи; в противном случае в трубке будут раздаваться длинные гудки или механический голос произнесет: “К сожалению, этот номер больше не обслуживается. Пожалуйста, проверьте номер и попытайтесь еще раз повторить свой вызов”. В сети также может обнаружиться соответствующая аналогия, если, например, во время серфинга по Интернету при попытке перейти на следующий сайт появляется сообщение: “Не удалось установить связь с сервером. Сервер не отвечает или не доступен”.

Допустимые номера портов изменяются в диапазоне от 0 до 65535, хотя номера меньше 1024 зарезервированы для системы. При использовании системы, совместимой с POSIX (например, Linux, Mac OS X и т.д.), список зарезервированных номеров портов (наряду с обозначениями серверов и протоколов, а также типов сокетов) можно найти в файле `/etc/services`. Список известных номеров портов находится на веб-сайте <http://www.iana.org/assignments/port-numbers>.

2.3.3. Сокеты с установлением и без установления соединения

Сокеты с установлением соединения

Сокеты подразделяются не только по принципу принадлежности к семейству адресов, но и по видам соединений. К первой категории относятся сокеты с установлением соединения (*connection-oriented sockets*). Это означает, что соединение должно быть установлено до начала сеанса связи, как это происходит в обычной телефонной сети. Такой способ организации связи называют также *созданием виртуального канала* (*virtual circuit*) или *применением потокового сокета* (*stream socket*).

Связь с установлением соединения обеспечивает упорядоченную, надежную, не нуждающуюся в дублировании доставку данных, при которой отправитель и получатель могут обмениваться целыми записями, не разбивая их на фрагменты. Это по существу означает, что любое сообщение может быть разделено на фрагменты и после доставки в место назначения все фрагменты будут соединены в должном порядке и переданы ожидающему приложению.

Основным протоколом, с помощью которого реализуются соединения такого типа, является *протокол управления передачей* (который чаще всего упоминается в виде сокращения, TCP — Transmission Control Protocol). Для создания сокетов TCP необходимо указать в качестве типа сокета `SOCK_STREAM`. Имя `SOCK_STREAM` относится к потоковому сокету. В сетевой версии этих сокетов (`AF_INET`) используется *протокол Интернета* (Internet Protocol — IP) для поиска хостов в сети, поэтому вся данная система организации сетевого взаимодействия называется именами обоих протоколов (TCP и IP) — TCP/IP. (Безусловно, протокол TCP можно также использовать с локальными, или несетевыми сокетами, `AF_LOCAL` или `AF_UNIX`, но очевидно, что при этом протокол IP не требуется.)

Сокеты без установления соединений

От виртуальных каналов резко отличаются сокеты *дейтаграммного* типа, которые функционируют без установления соединений. Это означает, что перед началом сеанса связи не требуется устанавливать какие-либо соединения. Иными словами, нет никаких гарантий доставки данных в той же последовательности, в которой

происходила их отправка, надежности или отсутствия дублирования в процессе передачи. К тому же каждая дейтаграмма передается вне связи с другими дейтаграммами. Таким образом, при передаче дейтаграмм не предусматривается разбиение на фрагменты, в отличие от передачи с помощью протоколов, предусматривающих установление соединения.

Доставку сообщений с использованием дейтаграмм можно сравнить с почтовой службой. Почта не гарантирует доставки писем или посылок в той же последовательности, в которой они были отправлены. Фактически некоторые почтовые отправления могут вообще не быть получены! Сложность организации сетей с применением протоколов без установления соединений усугубляется тем, что в процессе передачи может даже происходить *дублирование* сообщений.

Итак, очевидно, что протоколы последнего типа обладают многими недостатками, поэтому возникает вопрос, для чего вообще следует использовать дейтаграммы. (Ведь должны же они иметь *хоть какие-то* преимущества по сравнению с потоковыми сокетам.) Дело в том, что сокеты с установлением логических соединений предоставляют гарантии доставки, поэтому требуются существенные издержки не только для их настройки, но и для поддержания соединения с виртуальным каналом. Дейтаграммы не требуют таких издержек, поэтому являются “менее дорогостоящими”. Протоколы без установления соединений обычно обеспечивают более высокую производительность и могут оказаться наиболее подходящими для приложений некоторых типов.

Основным протоколом, который реализует такие типы соединений, является *протокол пользовательских дейтаграмм* (более известный под его сокращенным названием UDP — User Datagram Protocol). Для создания сокетов UDP необходимо использовать в качестве типа сокета SOCK_DGRAM. Вполне очевидно, что имя SOCK_DGRAM для сокета UDP указывает на его происхождение от слова “дейтаграмма” (datagram). В этих сокетах для поиска хостов в сети также используется протокол Интернета (Internet Protocol — IP), поэтому и сама система применяемых протоколов называется именами обоих протоколов (UDP и IP) — UDP/IP.

2.4. Сетевое программирование на языке Python

Итак, мы вкратце ознакомились с основными сведениями об архитектуре “клиент-сервер”, сокетах и организации сетей, а теперь попытаемся перенести эти понятия в язык Python. Основным модулем Python, используемым в этом разделе, является модуль `socket`. В этом модуле определена функция `socket()`, предназначенная для создания объектов сокетов. Сами сокеты также имеют собственный ряд методов, которые обеспечивают применение сетевого взаимодействия на основе сокетов.

2.4.1. Функция модуля `socket()`

Для создания сокета необходимо воспользоваться функцией `socket.socket()`, которая имеет следующий общий синтаксис:

```
socket(socket_family, socket_type, protocol=0)
```

Здесь в качестве параметра `socket_family` может применяться `AF_UNIX` или `AF_INET`, как было описано выше, а параметр `socket_type` может представлять собой `SOCK_STREAM` или `SOCK_DGRAM`, о чем также шла речь перед этим. Параметр `protocol` обычно не задается и в таком случае применяется значение по умолчанию, равное 0.

Таким образом, для создания сокета TCP/IP можно вызвать функцию `socket`. `socket()` примерно так:

```
tcpSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Аналогичным образом для создания сокета UDP/IP можно выполнить следующее:

```
udpSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Модуль `socket` имеет множество атрибутов, поэтому его импорт в сценарии целесообразно осуществлять не так, как обычно, а в форме `from module import *`. Применение инструкции `from socket import *` приводит к тому, что все атрибуты сокета переносятся в пространство имен сценария, но при этом существенно сокращается объем кода, как показывает следующий пример:

```
tcpSock = socket(AF_INET, SOCK_STREAM)
```

После создания объекта сокета все дальнейшие операции сетевого взаимодействия осуществляются с использованием методов объекта сокета.

2.4.2. Методы объекта сокета (встроенные)

Наиболее широко применяемые методы сокета представлены в табл. 2.1. В следующих разделах будет показано, как создавать клиенты и серверы TCP и UDP с использованием некоторых из этих методов. Безусловно, для нас наибольший интерес представляют сокеты Интернета, но следует отметить, что рассматриваемые здесь методы имеют свои аналоги в локальных (несетевых) сокетах.

Таблица 2.1. Общие методы и атрибуты объекта сокета

Имя	Описание
Методы сокета сервера	
<code>s.bind()</code>	Устанавливает привязку адреса (пары, состоящей из имени хоста и номера порта) к сокету
<code>s.listen()</code>	Устанавливает и запускает приемник TCP
<code>s.accept()</code>	Пассивно принимает клиентский запрос на установление соединения TCP, находясь в состоянии ожидания до поступления запроса на установление (блокирующий режим)
Методы сокета клиента	
<code>s.connect()</code>	Активно инициирует соединение сервера TCP
<code>s.connect_ex()</code>	Представляет собой расширенную версию метода <code>connect()</code> , которая предусматривает возврат сообщений о возникших проблемах в виде кодов ошибок, а не генерирование исключения
Общие методы сокета	
<code>s.recv()</code>	Обеспечивает получение сообщения TCP
<code>s.recv_into()^a</code>	Обеспечивает получение сообщения TCP в указанный буфер
<code>s.send()</code>	Обеспечивает передачу сообщения TCP
<code>s.sendall()</code>	Обеспечивает полную передачу сообщения TCP
<code>s.recvfrom()</code>	Обеспечивает получение сообщения UDP
<code>s.recvfrom_into()^a</code>	Обеспечивает получение сообщения UDP в указанный буфер
<code>s.sendto()</code>	Обеспечивает передачу сообщения UDP

Окончание табл. 2.1

Имя	Описание
<code>s.getpeername()</code>	Задаёт удаленный адрес, подключенный к сокету (TCP)
<code>s.getsockname()</code>	Задаёт адрес текущего сокета
<code>s.getsockopt()</code>	Возвращает значение указанной опции сокета
<code>s.setsockopt()</code>	Задаёт значение для какой-то конкретной опции сокета
<code>s.shutdown()</code>	Осуществляет останов соединения
<code>s.close()</code>	Обеспечивает закрытие сокета
<code>s.detach()^b</code>	Обеспечивает закрытие сокета без закрытия дескриптора файла и возвращает дескриптор
<code>s.ioctl()^c</code>	Осуществляет управление режимом сокета (только в Windows)
Методы сокета, ориентированные на поблочную передачу	
<code>s.setblocking()</code>	Задаёт блокирующий или неблокирующий режим сокета
<code>s.settimeout()^d</code>	Задаёт тайм-аут для блокирующих операций сокета
<code>s.gettimeout()^d</code>	Определяет тайм-аут для блокирующих операций сокета
Файловые методы сокета	
<code>s.fileno()</code>	Определяет дескриптор файла сокета
<code>s.makefile()</code>	Создает объект файла, связанный с сокетом
Атрибуты данных	
<code>s.family^a</code>	Семейство сокетов
<code>s.type^a</code>	Тип сокета
<code>s.proto^a</code>	Протокол сокета

^a Новое в Python 2.5.^b Новое в Python 3.2.^c Новое в Python 2.6; только платформа Windows. В системах POSIX могут использоваться функции модуля `fcntl`.^d Новое в Python 2.3.

Устанавливайте клиенты и серверы на разных компьютерах при организации работы сетевых приложений

Среди многочисленных примеров, представленных в этой главе, часто будет встречаться код и вывод, в которых упоминается хост “localhost” или отображается IP-адрес 127.0.0.1. В подобных примерах клиенты и серверы работают на одном и том же компьютере. Рекомендуем читателю заменять эти имена хостов и копировать код на разные компьютеры, поскольку намного интереснее заниматься разработкой и экспериментами с кодом, который позволяет компьютерам взаимодействовать по сети, а также наблюдать за работой сетевых программ, которые действительно выполняют нетривиальные действия!

2.4.3. Создание сервера TCP

Вначале рассмотрим некоторый общий псевдокод, необходимый для создания обычного сервера TCP, а затем приведем общее описание того, что при этом происходит. Следует учитывать, что это — лишь один из способов проектирования сервера. После того как вы в большей степени освоите проблематику проектирования

серверов, вы сможете вносить изменения в следующий псевдокод, чтобы приспособить его под свои потребности:

```

ss = socket()           # создание сокета сервера
ss.bind()               # привязка сокета к адресу
ss.listen()             # прослушивание запросов на соединение
    inf_loop:           # бесконечный цикл сервера
        cs = ss.accept() # прием клиентского запроса на установление соединения
        comm_loop:      # цикл связи
            cs.recv()/cs.send() # диалоговое окно (параметры приема/передачи)
        cs.close()       # закрытие сокета клиента
ss.close()              # закрытие сокета сервера # (необязательно)

```

Все сокеты создаются с использованием функции `socket.socket()`. Серверы должны оставаться подключенными к порту и ожидать запросов, поэтому их следует обязательно привязывать к локальному адресу. Протокол TCP лежит в основе системы связи с установлением соединений, поэтому должна быть определена некоторая инфраструктура, прежде чем может начаться эксплуатация сервера TCP. В частности, серверы TCP должны “прослушивать” входящие запросы на установление соединений. После завершения этого процесса начальной установки может быть осуществлен запуск бесконечного цикла сервера.

Затем простой (однопоточный) сервер ожидает возврата управления после вызова функции `accept()`, иными словами, ожидает результатов приема запроса на установление соединения. По умолчанию функция `accept()` является блокирующей, а это означает, что после вызова этой функции дальнейшее выполнение сценария приостанавливается до поступления запроса на установление соединения. Сокеты также поддерживают неблокирующий режим; для ознакомления с подробными сведениями о том, когда и как используются неблокирующие сокеты, обратитесь к документации или к учебникам по операционной системе.

После приема запроса на установление соединения функция `accept()` возвращает отдельный сокет клиента для последующего обмена сообщениями. Создание нового сокета клиента можно сравнить с перенаправлением оператором вызова клиента к одному из представителей отдела обслуживания. После того как клиент дозванивается в компанию, оператор на главном телефонном коммутаторе принимает входящий звонок и прослеживает его дальнейшее прохождение до того момента, когда произойдет подключение к другому номеру телефона, принадлежащему тому лицу, которое должно выполнить просьбу клиента.

После этого главная телефонная линия освобождается (как и исходный сокет сервера), что позволяет оператору возобновить ожидание поступления новых звонков (клиентских запросов), в то время как клиент и представитель отдела обслуживания продолжают свой диалог по другой телефонной линии. Аналогичным образом после поступления на сервер входящего запроса открывается новый порт связи для обмена данными непосредственно с клиентом, отправившим вызов, что также позволяет освободить основной порт для приема новых клиентских запросов на установление соединения.



Порождение новых потоков для обработки клиентских запросов

В наших примерах это не реализовано, но довольно часто применяется такой вариант, когда клиентский запрос передается в новый поток или процесс, в котором осуществляется обработка этого клиентского запроса. На основе модуля `socket` создан высоко-

уровневый модуль связи через сокет `SocketServer`, который поддерживает не только многопоточковую обработку, но и порождение процессов для обработки клиентских запросов. Обратитесь к документации для получения подробных сведений о модуле `SocketServer`, а также к примерам в главе 4, “Многопоточковое программирование”.

После создания временного сокета может начаться обмен данными, для чего клиент и сервер приступают к участию в диалоге, в ходе которого происходит передача и прием необходимой информации через этот новый сокет до тех пор, пока соединение не будет закрыто. Закрытие соединения обычно происходит после того, как один из участников обмена данными либо закрывает соединение со своей стороны, либо отправляет пустую строку другому участнику.

В коде, рассматриваемом в данной главе, показано, что после закрытия клиентского соединения сервер снова переходит в состояние ожидания очередного клиентского запроса на установление соединения. Последняя строка кода, в которой происходит закрытие сокета сервера, является необязательной. Дело в том, что эта инструкция никогда не выполняется, поскольку сервер должен работать в нескончаемом цикле. Однако мы оставили эту инструкцию в нашем примере как напоминание читателю о том, что вызов метода `close()` рекомендуется в качестве способа реализации корректной схемы выхода сервера, например, после обнаружения обработчиком некоторого внешнего условия, указывающего на то, что сервер должен быть остановлен. В таких случаях вызов метода `close()` гарантирован.

В примере 2.1 представлен сценарий `tsTserv.py` — программа сервера TCP, которая принимает строку данных, отправленную клиентом, и возвращает ее с отметкой времени (в формате `[timestamp]data`) назад клиенту. (Здесь “`tsTserv`” представляет собой сокращение от `timestamp TCP server` — сервер TCP отметок времени. Прочие файлы именуются аналогичным образом.)

Пример 2.1. Сервер отметок времени (`tsTserv.py`)

В этом сценарии создается сервер TCP, который принимает сообщения от клиентов и возвращает их с префиксом в виде отметки времени.

```
1  #!/usr/bin/env python
2
3  from socket import *
4  from time import ctime
5
6  HOST = ''
7  PORT = 21567
8  BUFSIZ = 1024
9  ADDR = (HOST, PORT)
10
11 tcpSerSock = socket(AF_INET, SOCK_STREAM)
12 tcpSerSock.bind(ADDR)
13 tcpSerSock.listen(5)
14
15 while True:
16     print 'waiting for connection...'
17     tcpCliSock, addr = tcpSerSock.accept()
18     print '...connected from:', addr
19
20     while True:
```

```

21     data = tcpCliSock.recv(BUFSIZ)
22     if not data:
23         break
24     tcpCliSock.send('[%s] %s' % (
25         ctime(), data))
26
27     tcpCliSock.close()
28     tcpSerSock.close()

```

Построчное объяснение

Строки 1–4

В начале сценария находится строка для командного интерпретатора Unix, указывающая путь к интерпретатору языка Python, за которой следуют инструкции импорта функции `time.ctime()` и всех атрибутов модуля `socket`.

Строки 6–13

Переменная `HOST` пуста. Это указывает на то, что в методе `bind()` может использоваться любой доступный адрес. Кроме того, случайным образом выбран номер порта, который, по всей видимости, не является используемым или зарезервированным системой. Для рассматриваемого приложения задан размер буфера, равный 1 Кбайт. Этот размер можно изменить в зависимости от настройки сети и требований к приложению. Параметр в вызове метода `listen()` указывает максимальное количество входящих запросов на установление соединения, которые могут быть приняты, прежде чем сервер начнет отклонять поступающие запросы или отказывать в их выполнении.

Вызов сокета сервера TCP (`tcpSerSock`) представлен в строке 11. Затем следуют инструкции с вызовами, применяемыми для привязки сокета к адресу сервера и запуска приемника TCP.

Строки 15–28

После перехода в бесконечный цикл сервера начинается ожидание запроса на установление соединения (пассивное действие). После получения такого запроса открывается цикл ведения диалога, в котором происходит ожидание отправки клиентом своего сообщения. Если сообщение является пустым, это означает, что клиент завершил свою работу, поэтому происходит выход из цикла ведения диалога, закрытие соединения с клиентом, а затем возврат к ожиданию следующего запроса от клиента. Если сообщение, полученное от клиента, не пусто, то осуществляется форматирование и возврат тех же данных, но снабженных префиксом в виде текущей отметки времени. Последняя строка никогда не выполняется; она служит лишь напоминанием читателю о том, что должен всегда применяться вызов `close()`, если обработчик написан с учетом возможности более корректного выхода, как было описано выше.

Теперь рассмотрим вариант того же сценария для версии Python 3 (`tsTserv3.py`), как показано в примере 2.2:

Пример 2.2. Сервер отметок времени (tsTserv3.py)

В этом сценарии создается сервер TCP, который принимает сообщения от клиентов и возвращает их с префиксом в виде отметки времени.

```

1  #!/usr/bin/env python
2
3  from socket import *
4  from time import ctime
5
6  HOST = ''
7  PORT = 21567
8  BUFSIZ = 1024
9  ADDR = (HOST, PORT)
10
11 tcpSerSock = socket(AF_INET, SOCK_STREAM)
12 tcpSerSock.bind(ADDR)
13 tcpSerSock.listen(5)
14
15 while True:
16     print('waiting for connection...')
17     tcpCliSock, addr = tcpSerSock.accept()
18     print('...connected from:', addr)
19
20     while True:
21         data = tcpCliSock.recv(BUFSIZ)
22         if not data:
23             break
24         tcpCliSock.send(['[%s] %s' % (
25             bytes(ctime(), 'utf-8'), data)])
26
27     tcpCliSock.close()
28 tcpSerSock.close()

```

Соответствующие изменения в строках 16, 18 и 25 обозначены курсивом. Они заключаются в том, что вместо оператора **print** применяется функция, а строки передаются в виде байтов в коде ASCII (с типом данных "string"), а не в Unicode. Ниже в данной книге будет рассматриваться вопрос о том, как перейти от версии Python 2 к версии Python 3 и как обеспечить возможность написания кода, который может выполняться без изменений в версиях интерпретаторов 2.x и 3.x.

Может также возникнуть необходимость предусмотреть еще одну пару вариантов для поддержки протокола IPv6, tsTservV6.py и tsTserv3V6.py, но она здесь не показана. Отметим только, что в них достаточно при создании сокета изменить семейство адресов с AF_INET (IPv4) на AF_INET6 (IPv6). (Для тех, кто не знаком с этими терминами, напомним, что IPv4 обозначает текущий протокол Интернета. Новое поколение протокола Интернета имеет версию 6, поэтому обозначается как "IPv6".)

2.4.4. Создание клиента TCP

Создание клиента намного проще по сравнению с созданием сервера. По аналогии с приведенным выше описанием сервера TCP вначале представим псевдокод с объяснениями, а затем покажем, как фактически выглядит действующий сценарий.

```

cs = socket()           # создание сокета клиента
cs.connect()            # осуществление попытки установить соединение с сервером
comm_loop:              # цикл связи
    cs.send()/cs.recv() # ведение диалога (передача/прием)
cs.close()              # закрытие сокета клиента

```

Как уже было сказано выше, все сокеты создаются с использованием функции `socket.socket()`. Однако в отличие от сервера, клиент после создания для него сокета может немедленно приступить к установлению соединения с сервером с помощью метода сокета `connect()`. После установления соединения клиент может приступить к ведению диалога с сервером. После завершения клиентом своей транзакции он может закрыть свой сокет, завершая тем самым соединение.

Код сценария `tsTclnt.py` представлен в примере 2.3. Этот сценарий показывает, как выполнить подключение к серверу, затем снова и снова выводить запрос к пользователю для ввода одной строки с данными за другой. Сервер возвращает эти данные с отметкой времени, после чего с помощью кода клиентского сценария происходит их вывод для пользователя.

Пример 2.3. Клиент TCP отметок времени (`tsTclnt.py`)

В этом сценарии создается клиент TCP, который формирует запросы пользователю на получение сообщений, подлежащих отправке на сервер, получает эти сообщения от сервера с префиксом в виде отметки времени, а затем отображает результаты для пользователя.

```

1  #!/usr/bin/env python
2
3  from socket import *
4
5  HOST = 'localhost'
6  PORT = 21567
7  BUFSIZ = 1024
8  ADDR = (HOST, PORT)
9
10 tcpCliSock = socket(AF_INET, SOCK_STREAM)
11 tcpCliSock.connect(ADDR)
12
13 while True:
14     data = raw_input('> ')
15     if not data:
16         break
17     tcpCliSock.send(data)
18     data = tcpCliSock.recv(BUFSIZ)
19     if not data:
20         break
21     print data
22
23 tcpCliSock.close()

```

Построчное объяснение

Строки 1–3

В начале сценария содержится строка для командного интерпретатора Unix, указывающая путь к интерпретатору языка Python, за которой следует инструкция импорта всех атрибутов из модуля `socket`.

Строки 5–11

Переменные `HOST` и `PORT` указывают имя хоста и номер порта сервера. Этот тестовый сценарий выполняется (в данном случае) на одном и том же компьютере, поэтому переменная `HOST` содержит локальное имя хоста (если сервер и клиент находятся на разных хостах, это имя необходимо заменить соответствующим образом). Номер порта `PORT` должен быть точно таким же, какой был задан в программе сервера (поскольку в противном случае связь между клиентом и сервером станет невозможной). Кроме того, выбран такой же размер буфера — 1 Кбайт.

Инструкция создания сокета клиента TCP (`tcpCliSock`) содержится в строке 10. За ней следует вызов функции подключения к серверу (активное действие).

Строки 13–23

Клиент также функционирует в бесконечном цикле, но в отличие от цикла сервера, условием выхода из него является действие, выполняемое в самой программе. Цикл клиента завершается при одном из двух следующих условий: пользователь не задает никаких входных данных (строки 14–16), или по каким-то причинам сервер прекращает свое функционирование, поэтому вызов метода `recv()` оканчивается неудачей (строки 18–20). В противном случае в обычной ситуации пользователь вводит те или иные строковые данные, которые передаются серверу на обработку. Затем происходит получение той же входной строки, но с отметкой времени, и отображение ее на экране.

Аналогично тому, что было показано выше для сервера, рассмотрим варианты для клиента Python 3 и IPv6 (`tsTclnt3.py`), начиная с первого, как показано в примере 2.4:

Пример 2.4. Клиент TCP отметок времени для Python 3 (`tsTclnt3.py`)

Эта программа эквивалентна приведенной выше, но предназначена для версии Python 3 `tsTclnt3.py`.

```
1      #!/usr/bin/env python
2
3      from socket import *
4
5      HOST = '127.0.0.1' # or 'localhost'
6      PORT = 21567
7      BUFSIZ = 1024
8      ADDR = (HOST, PORT)
9
10     tcpCliSock = socket(AF_INET, SOCK_STREAM)
11     tcpCliSock.connect(ADDR)
12
13     while True
14         data = input('> ')
```

```
15         if not data
16             break
17         tcpCliSock.send(data)
18         data = tcpCliSock.recv(BUFSIZ)
19         if not data
20             break
21         print(data.decode('utf-8'))
22
23     tcpCliSock.close()
```

Мы должны были не только заменить оператор вызова функции `print`, но и обеспечить декодирование строки, поступающей от сервера. (С помощью функции `distutils.log.warn()` исходный сценарий можно легко преобразовать так, чтобы он мог работать и в версии Python 2, и в версии Python 3, по аналогии с тем, как это было сделано в сценарии `rewhoU.py`, приведенном в главе 1.) Наконец, рассмотрим вариант для IPv6 (в версии Python 2), `tsTclntV6.py`, как показано в примере 2.5.

Пример 2.5. Клиент TCP отметок времени IPv6 (`tsTclntV6.py`)

Это вариант клиента TCP из двух предыдущих примеров, предназначенный для протокола IPv6.

```
1     #!/usr/bin/env python
2
3     from socket import *
4
5     HOST = ':::1'
6     PORT = 21567
7     BUFSIZ = 1024
8     ADDR = (HOST, PORT)
9
10    tcpCliSock = socket(AF_INET6, SOCK_STREAM)
11    tcpCliSock.connect(ADDR)
12
13    while True:
14        data = raw_input('> ')
15        if not data:
16            break
17        tcpCliSock.send(data)
18        data = tcpCliSock.recv(BUFSIZ)
19        if not data:
20            break
21        print data
22
23    tcpCliSock.close()
```

В данном фрагменте потребовалось внести такие изменения: вместо имени сервера `localhost` указать его адрес IPv6, `:::1`, а в запросе применить семейство сокетов `AF_INET6`. Совместное использование изменений из `tsTclnt3.py` и `tsTclntV6.py` позволяет также создать вариант клиента TCP для версии Python 3 и протокола IPv6.

2.4.5. Эксплуатация сервера и клиентов TCP

Теперь приступим к использованию серверных и клиентских программ для ознакомления с тем, как они работают. Следует ли в первую очередь вызывать на выполнение сервер или клиент? Безусловно, если вначале будет вызван на выполнение клиент, то ему не удастся установить соединение, поскольку отсутствует сервер, ожидающий поступления запроса. Сервер рассматривается как пассивный участник соединения, поскольку он приступает к работе первым и пассивно ожидает запросов на установление соединения. Клиент, с другой стороны, активный участник соединения, так как именно клиент активно инициирует соединение. Перефразируем сказанное немного иначе.

Сначала необходимо запустить сервер (до осуществления каких-либо попыток подключения со стороны клиентов).

В рассматриваемом примере используется тот же компьютер, но нет каких-либо препятствий к тому, чтобы развернуть сервер на другом хосте. При использовании такой конфигурации достаточно только изменить имя хоста. (Тот момент, когда впервые удастся запустить сетевое приложение, действующее так, что сервер и клиент работают на разных компьютерах, производит неизгладимое впечатление!)

Ниже приведен соответствующий ввод и вывод из клиентской программы, завершение работы с которой происходит после нажатия клавиши <Return> (или <Enter>) без ввода данных:

```
$ tsTclnt.py
> hi
[Sat Jun 17 17:27:21 2006] hi
> spanish inquisition
[Sat Jun 17 17:27:37 2006] spanish inquisition
>
$
```

Вывод сервера в основном предназначен для диагностики:

```
$ tsTserv.py
waiting for connection...
...connected from: ('127.0.0.1', 1040)
waiting for connection...
```

После создания соединения с клиентом будет получено сообщение "...connected from...". В ходе сеанса клиент продолжает получать "обслуживание" через установленное соединение, а наряду с этим сервер ожидает получения новых запросов от клиентов. В этом варианте программы сервера не предусмотрено корректное завершение работы, поэтому приходится отправлять серверу сигнал прерывания, что приводит к возникновению исключения. Лучший способ, позволяющий предотвратить такую ошибку, состоит в использовании средств нормального завершения работы сервера, о чем было сказано выше.



Обеспечение корректного выхода и вызов метода `close()` сервера

Один из способов обеспечения более корректного выхода заключается в том, чтобы поместить цикл `while` сервера в предложение `except` инструкции `try-except` и

отслеживать исключение `EOFError` или `KeyboardInterrupt`, что позволяло бы закрыть сокет сервера с помощью предложения **except** или **finally**. В реальной эксплуатации может потребоваться применение автоматизированного способа запуска и останова серверов. В таких случаях желательно предусмотреть выставление флага для останова службы с использованием потока, формировать запись в базе данных или создавать специальный файл.

Интересной особенностью этого простого сетевого приложения является то, что оно не только демонстрирует, как происходит обмен данными между клиентом и сервером, но и позволяет ознакомиться с работой своего рода “сервера времени”, поскольку приложение предусматривает получение отметок времени непосредственно от сервера.

2.4.6. Создание сервера UDP

В отличие от серверов TCP, серверы UDP не требуют столь значительной настройки, поскольку не предусматривают установление логических соединений. Практически не требуется выполнение какой-либо подготовительной работы, и сервер может сразу же приступить к ожиданию поступления входящих запросов.

```
ss = socket()           # создание сокета сервера
ss.bind()               # привязка сокета сервера
inf_loop:               # бесконечный цикл сервера
    cs = ss.recvfrom()/ss.sendto() # диалоговое окно (параметры приема/передачи)
ss.close()              # закрытие сокета сервера
```

Как показывает приведенный псевдокод, никаких дополнительных действий не требуется, кроме обычного создания сокета и привязки его к локальному адресу (пара хост/порт). Сервер работает в бесконечном цикле, состоящем в том, что происходит получение сообщения от клиента, формирование отметки времени и возврат сообщения, после чего сервер снова возвращается к ожиданию очередного запроса. И в этом примере вызов `close()` является необязательным и не может быть достигнут, поскольку не предусмотрен выход из цикла. Но эта команда служит напоминанием о том, что закрытие сокета должно стать частью корректной, или интеллектуальной схемы выхода, о которой речь шла выше.

Еще одно существенное различие между серверами UDP и TCP обусловлено следующим: дейтаграммные сокеты применяются без установления логических соединений, поэтому нет необходимости “передавать” клиентский запрос на установление соединения в отдельный сокет для последующего обмена данными. Такие серверы только принимают сообщения, а также, в случае необходимости, отвечают на них.

В примере 2.6 приведен код сценария `tsUser.v.py`, который представляет вариант приведенного выше кода сервера TCP для протокола UDP. Этот сервер принимает сообщения от клиентов и возвращает их клиентам с отметкой времени.

Пример 2.6. Сервер UDP отметок времени (`tsUser.v.py`)

В этом сценарии создается сервер UDP, который принимает сообщения от клиентов и возвращает их с префиксом в виде отметки времени.

```
1  #!/usr/bin/env python
2
3  from socket import *
```

```
4  from time import ctime
5
6  HOST = ''
7  PORT = 21567
8  BUFSIZ = 1024
9  ADDR = (HOST, PORT)
10
11  udpSerSock = socket(AF_INET, SOCK_DGRAM)
12  udpSerSock.bind(ADDR)
13
14  while True:
15      print 'waiting for message...'
16      data, addr = udpSerSock.recvfrom(BUFSIZ)
17      udpSerSock.sendto('[%s] %s' % (
18          ctime(), data), addr)
19      print '...received from and returned to:', addr
20
21  udpSerSock.close()
```

Построчное объяснение

Строки 1–4

Вслед за применяемой в системе Unix строкой, в которой указана программа, применяемая для выполнения сценария, приведены инструкции импорта функции `time.ctime()` и всех атрибутов модуля `socket`, как и в той части приведенных выше примеров, где выполняется подготовка к работе сервера TCP.

Строки 6–12

Переменные `HOST` и `PORT` определены так же, как и в предыдущих примерах, и применяются для тех же целей. Вызов `socket()` отличается только тем, что теперь мы запрашиваем создание дейтаграммного сокета (сокета UDP), а вызов функции `bind()` происходит точно так же, как и в варианте с сервером TCP. Еще раз отметим, что протокол UDP не предусматривает установление логических соединений, поэтому в рассматриваемом сценарии не обеспечивается какое-либо “прослушивание входящих запросов на установление соединения”.

Строки 14–21

После перехода в сценарии в бесконечный цикл сервера происходит (пассивное) ожидание входящих сообщений (дейтаграмм). После поступления такого сообщения происходит его обработка (путем добавления к нему отметки времени), затем отправка этого сообщения клиенту и возврат к ожиданию следующего сообщения. Метод сокета `close()` и в этом примере, как и в предыдущих, указан лишь для напоминания.

2.4.7. Создание клиента UDP

Среди всех четырех вариантов клиентов, которые рассматривались в этом разделе, вариант для клиента UDP имеет наименьший объем кода. Псевдокод этого клиента выглядит следующим образом:

```

cs = socket()                # создание сокета клиента
    comm_loop:                # цикл связи
        cs.sendto()/cs.recvfrom() # ведение диалога (передача/прием)
cs.close()                    # закрытие сокета клиента

```

После создания объекта сокета открывается цикл ведения диалога, в котором происходит обмен сообщениями с сервером. После окончания связи сокет закрывается.

Фактически применяемый код клиента, сценарий `tsUclnt.py`, представлен в примере 2.7.

Пример 2.7. Клиент UDP отметок времени (`tsUclnt.py`)

В этом сценарии создается клиент UDP, который выводит приглашения для пользователя к вводу сообщений, предназначенных для отправки на сервер, получает от сервера ответы с префиксом в виде отметки времени, а затем отображает полученные результаты для пользователя.

```

1  #!/usr/bin/env python
2
3  from socket import *
4
5  HOST = 'localhost'
6  PORT = 21567
7  BUFSIZ = 1024
8  ADDR = (HOST, PORT)
9
10 udpCliSock = socket(AF_INET, SOCK_DGRAM)
11
12 while True:
13     data = raw_input('> ')
14     if not data:
15         break
16     udpCliSock.sendto(data, ADDR)
17     data, ADDR = udpCliSock.recvfrom(BUFSIZ)
18     if not data:
19         break
20     print data
21
22 udpCliSock.close()

```

Построчное объяснение

Строки 1–3

Вслед за начальной строкой Unix снова представлена инструкция импорта всех атрибутов из модуля `socket`, как и в варианте клиента для TCP.

Строки 5–10

В данном примере сервер также эксплуатируется на локальном компьютере, поэтому снова используется хост с именем “localhost”, а в клиентской программе задан тот же номер порта, не говоря уже об одинаковом размере буфера в 1 Кбайт. Для клиента объект сокета создается таким же образом, как и для сервера UDP.

Строки 12–22

Цикл в клиентской программе UDP функционирует почти полностью одинаково с клиентом TCP. Единственное различие состоит в том, что не требуется в первую очередь устанавливать соединение с сервером UDP; просто происходят отправка серверу сообщения и получение ответа. После того как происходит возврат строки с отметкой времени, эта строка отображается на экране, а работа в цикле возобновляется. Когда наступает время завершения ввода, осуществляются выход из цикла и закрытие сокета.

На основе примеров клиентов и серверов TCP можно довольно просто создать эквивалентные сценарии UDP для версии Python 3 и протокола IPv6.

2.4.8. Эксплуатация сервера и клиентов UDP

Клиент UDP действует так же, как клиент TCP:

```
$ tsUclnt.py
> hi
[Sat Jun 17 19:55:36 2006] hi
> spam! spam! spam!
[Sat Jun 17 19:55:40 2006] spam! spam! spam!
>
$
```

Аналогично для сервера:

```
$ tsUclnt.py
waiting for message...
...received from and returned to: ('127.0.0.1', 1025)
waiting for message...
```

Фактически осуществляется вывод информации о клиенте, поскольку сервер может получать сообщения от многих клиентов и отправлять ответы, а такой вывод позволяет проще определять, откуда поступили сообщения. С другой стороны, сервер TCP позволяет непосредственно определять источники сообщений, поскольку каждый клиент устанавливает соединение. Заслуживает внимания то, что в сообщениях сказано "waiting for message" (ожидание сообщения), а не "waiting for connection" (ожидание соединения).

2.4.9. Атрибуты модуля socket

Модуль socket, кроме уже знакомой нам функции `socket.socket()`, содержит еще много атрибутов, которые используются при разработке сетевых приложений. Некоторые из наиболее широко применяемых атрибутов показаны в табл. 2.2.

Таблица 2.2. Атрибуты модуля socket

Имя атрибута	Описание
Атрибуты данных	
<code>AF_UNIX</code> , <code>AF_INET</code> , <code>AF_INET6</code> , ^a	Семейства адресов сокета, поддерживаемые языком Python
<code>AF_NETLINK</code> , ^b <code>AF_TIPC</code>	
<code>SO_STREAM</code> , <code>SO_DGRAM</code>	Типы сокетов (TCP — потоковый, UDP — дейтаграммный)
<code>has_ipv6</code> ^d	Логический флаг, показывающий, поддерживается ли протокол IPv6

Имя атрибута	Описание
Исключения	
<code>error</code>	Ошибка, связанная с сокетом
<code>herror^a</code>	Ошибка, связанная с хостом и адресом
<code>gaierror^a</code>	Ошибка, связанная с адресом
<code>timeout</code>	Истечение срока действия тайм-аута
Функции	
<code>socket()</code>	Создание объекта сокета на основе указанных параметров: семейство адресов, тип сокета и тип протокола (необязательно)
<code>socketpair()^e</code>	Создание пары объектов сокета на основе указанных параметров: семейство адресов, тип сокета и тип протокола (необязательно)
<code>create_connection()</code>	Вспомогательная функция, которая принимает пару параметров определения адреса (хост, порт) и возвращает объект <code>socket</code>
<code>fromfd()</code>	Создание объекта сокета на основе дескриптора открытого файла
<code>ssl()</code>	Иницирует соединение SSL (Secure Socket Layer) через сокет; проверка сертификата не выполняется
<code>getaddrinfo()^a</code>	Получает информацию адреса в виде последовательности пятиэлементных кортежей
<code>getnameinfo()</code>	Принимая в качестве параметра адрес сокета, возвращает двухэлементный кортеж (хост, порт)
<code>getfqdn()^f</code>	Возвращает полное доменное имя
<code>gethostname()</code>	Возвращает текущее имя хоста
<code>gethostbyname()</code>	Отображает имя хоста в его IP-адрес
<code>gethostbyname_ex()</code>	Расширенная версия функции <code>gethostbyname()</code> , возвращающая имя хоста, набор имен хостов псевдонимов и список IP-адресов
<code>gethostbyaddr()</code>	Преобразует IP-адрес в информацию DNS; возвращает такой же трехэлементный кортеж, что и функция <code>gethostbyname_ex()</code>
<code>getprotobyname()</code>	Преобразует имя протокола (например, 'tcp') в числовое обозначение
<code>getservbyname()</code> / <code>getservbyport()</code>	Преобразует имя службы в номер порта, или наоборот; имя протокола является необязательным и для той, и для другой функции
<code>ntohl()</code> / <code>ntohs()</code>	Обеспечивает преобразование внутреннего представления целых чисел из формата сети в формат хоста
<code>htonl()</code> / <code>htons()</code>	Обеспечивает преобразование внутреннего представления целых чисел из формата хоста в формат сети
<code>inet_aton()</code> / <code>inet_ntoa()</code>	Преобразует строку октетов IP-адреса в 32-разрядный упакованный формат, или наоборот (только для адресов IPv4)
<code>inet_pton()</code> / <code>inet_ntop()</code>	Преобразует строку IP-адреса в упакованный двоичный формат, или наоборот (и для адресов IPv4, и для адресов IPv6)
<code>getdefaulttimeout()</code> / <code>setdefaulttimeout()</code>	Возвращает значение по умолчанию тайм-аута сокета в секундах (число с плавающей точкой); задает значение по умолчанию тайм-аута сокета в секундах (число с плавающей точкой)

^a Новое в Python 2.2.^d Новое в Python 2.3.^b Новое в Python 2.5.^e Новое в Python 2.4.^c Новое в Python 2.6.^f Новое в Python 2.0.

Для ознакомления с дополнительными сведения обратитесь к документации модуля `socket` в справочном руководстве по библиотеке Python (Python Library Reference).

2.5. *Модуль SocketServer

3.x

Модуль `SocketServer` представляет собой один из модулей высокого уровня в стандартной библиотеке (переименован в `socketserver` в версиях Python 3.x). Его назначение состоит в предоставлении большого объема стандартного кода, необходимого для создания сетевых клиентов и серверов. Этот модуль обеспечивает создание от имени пользователя различных классов, как показано в табл. 2.3.

Таблица 2.3. Классы модуля `SocketServer`

Класс	Описание
<code>BaseServer</code>	Содержит основные функциональные средства сервера и обработчики прерываний для промежуточных классов. Используется только для порождения других классов, поэтому не приходится создавать экземпляры данного класса. Вместо этого следует использовать класс <code>TCPServer</code> или <code>UDPServer</code>
<code>TCPServer/UDPServer</code>	Основной сетевой синхронный сервер TCP/UDP
<code>UnixStreamServer/ UnixDatagramServer</code>	Основной синхронный сервер TCP/UDP на основе файлов
<code>ForkingMixIn/ThreadingMixIn</code>	Предоставляет основные функциональные возможности ветвления или организации многопоточной работы. Используется только для создания промежуточных классов, применяемых в сочетании с одним из серверных классов для обеспечения определенной асинхронности. Непосредственное создание экземпляров этого класса не предусмотрено
<code>ForkingTCPServer/ ForkingUDPServer</code>	Сочетание <code>ForkingMixIn</code> и <code>TCPServer/UDPServer</code>
<code>ThreadingTCPServer/ ThreadingUDPServer</code>	Сочетание <code>ThreadingMixIn</code> и <code>TCPServer/UDPServer</code>
<code>BaseRequestHandler</code>	Содержит основные функциональные средства обработки запросов на обслуживание. Используется только для порождения других классов, поэтому в программах не создаются экземпляры данного класса. Вместо этого следует использовать <code>StreamRequestHandler</code> или <code>DatagramRequestHandler</code>
<code>StreamRequestHandler/ DatagramRequestHandler</code>	Реализация обработчика служб для серверов TCP/UDP

В этом разделе показано, как создать клиент и сервер TCP, которые предоставляют такие же возможности, как клиент и сервер в основном примере TCP, приведенном выше. Читатель заметит прямые аналоги между соответствующими сценариями, но заслуживает также внимания то, что в последнем примере уже не требуется выполнять некоторые дополнительные действия, поэтому данный код непосредственно применяется как стандартный. В частности, рассматриваемые серверы представляют собой наиболее простые синхронные серверы из всех возможных. (Для того чтобы

узнать, как выполнить настройку сервера в целях асинхронной эксплуатации, перейдите к упражнениям в конце данной главы.)

Отличительными особенностями примеров, рассматриваемых в этом разделе, являются не только то, что они не требуют от программиста углубляться в детали реализации, но и применение в них классов для написания прикладного кода. Применение объектно-ориентированного программирования способствует достижению лучшей организации данных и рациональному распределению функциональных возможностей в соответствии с назначением. Заслуживает также внимания то, что приложения теперь действуют на основе обработки событий, а это означает, что те или иные операции выполняются только в ответ на возникновение тех или иных событий в системе.

В число событий входят передача и получение сообщений. В действительности можно видеть, что определение класса состоит лишь из обработчика событий, предназначенного для получения сообщений от клиентов. Все прочие функциональные средства берутся из используемого класса `SocketServer`. В программировании для графического пользовательского интерфейса (см. главу 5) также применяется управление на основе событий. Сама аналогия между текущими и приведенными выше примерами обнаруживается при рассмотрении заключительной строки кода, которая, как обычно, задает бесконечный цикл сервера, который ожидает поступления клиентских запросов на обслуживание и отвечает на них. Этот сценарий действует фактически так же, как и сценарий с бесконечным циклом `while` в исходном примере с базовым сервером TCP, который рассматривался ранее в этой главе.

В приведенном первоначально цикле сервера применялось блокирование при ожидании запроса, затем происходило обслуживание любого из поступающих сообщений, после чего снова происходил возврат в состояние ожидания. В текущем примере цикл сервера является таковым, что само его формирование в коде сервера не предусматривается, а применяется обработчик прерываний, поэтому сервер может просто вызвать необходимую функцию после получения входящего запроса.

2.5.1. Создание сервера TCP с применением модуля `SocketServer`

Как показано в примере 2.8, сначала происходит импорт классов сервера, а затем определяются такие же константы хоста, как и в предыдущих примерах. За этим следует определение класса обработчика запросов и, наконец, происходит запуск сервера. Дополнительные сведения приведены за этим фрагментом кода.

Пример 2.8. Сервер TCP отметок времени на основе модуля `SocketServer` (`tsTservSS.py`)

В этом сценарии создается сервер TCP отметок времени с использованием классов `SocketServer`, `TCPServer` и `StreamRequestHandler`.

```
1  #!/usr/bin/env python
2
3  from SocketServer import (TCPServer as TCP,
4      StreamRequestHandler as SRH)
5  from time import ctime
6
7  HOST = ''
```

```
8     PORT = 21567
9     ADDR = (HOST, PORT)
10
11     class MyRequestHandler(SRH):
12         def handle(self):
13             print '...connected from:', self.client_address
14             self.wfile.write('%s %s' % (ctime(),
15                                     self.rfile.readline()))
16
17     tcpServ = TCP(ADDR, MyRequestHandler)
18     print 'waiting for connection...'
19     tcpServ.serve_forever()
```

Построчное объяснение

Строки 1–9

2.4

Сценарий начинается с импорта необходимых классов из модуля SocketServer. Заслуживает внимания то, что в данном примере используется многострочное средство импорта, впервые введенное в версию Python 2.4. При использовании одной из предыдущих версий Python необходимо указывать полные имена в формате *module.attribute* или помещать оба атрибута импорта в одной и той же строке:

```
from SocketServer import TCPServer as TCP, StreamRequestHandler as SRH
```

Строки 11–15

В этих строках находятся наиболее важные инструкции сценария. Прежде всего происходит получение обработчика запросов MyRequestHandler как подкласса StreamRequestHandler класса SocketServer и перекрытие его метода handle(), который оформлен в виде заглушки в классе BaseRequest без заданного по умолчанию действия следующим образом:

```
def handle(self):
    pass
```

При получении входящего сообщения от клиента вызывается метод handle(). В классе StreamRequestHandler сокет ввода и вывода рассматриваются как объекты, подобные файлам, поэтому в сценарии используются функция readline() для получения сообщения от клиента и функция write() для отправки строки обратно клиенту.

Соответственно, необходимо предусмотреть обработку дополнительных символов возврата каретки и перевода строки и в коде сервера, и в коде клиента. В действительности обработка этих символов в коде явно не показана, поскольку происходит всего лишь повторное использование указанных символов, полученных от клиента. Кроме этих небольших различий, рассматриваемая программа сервера не намного изменилась по сравнению с приведенной ранее программой сервера.

Строки 17–19

В последней части кода создается сервер TCP на основе заданной информации о хосте и класса обработчика запросов. Затем в сценарии происходит переход к бесконечному циклу, в котором осуществляется ожидание и обслуживание клиентских запросов.

2.5.2. Создание клиента TCP на основе модуля SocketServer

Вполне естественно, что код клиента, показанный в примере 2.9, в большей степени, чем код сервера, напоминает предыдущие варианты сценариев, но и в этом коде должны быть сделаны определенные поправки, чтобы обеспечить его успешную эксплуатацию в сочетании с вновь созданным сервером.

Пример 2.9. Клиент TCP отметок времени на основе модуля SocketServer (tsTclntSS.py)

Это — код клиента TCP отметок времени, который предназначен для обмена данными с файлоподобными объектами `StreamRequestHandler` класса `SocketServer`.

```
1      #!/usr/bin/env python
2
3      from socket import *
4
5      HOST = 'localhost'
6      PORT = 21567
7      BUFSIZ = 1024
8      ADDR = (HOST, PORT)
9
10     while True:
11         tcpCliSock = socket(AF_INET, SOCK_STREAM)
12         tcpCliSock.connect(ADDR)
13         data = raw_input('> ')
14         if not data:
15             break
16         tcpCliSock.send('%s\r\n' % data)
17         data = tcpCliSock.recv(BUFSIZ)
18         if not data:
19             break
20         print data.strip()
21         tcpCliSock.close()
```

Построчное объяснение

Строки 1–8

Этот код не содержит слишком заметных отличий; его скорее можно рассматривать как точную копию применявшегося ранее кода клиента.

Строки 10–21

По умолчанию поведение обработчиков запросов `SocketServer` состоит в том, чтобы принять запрос на установление соединения, получить сообщения от клиента, а затем закрыть соединение. Применение указанной организации работы влечет за собой то, что исключается возможность поддерживать одно и то же соединение на

протяжении всего времени выполнения данного приложения, поэтому каждый раз при отправке сообщения серверу должен создаваться новый сокет.

Благодаря такому поведению функционирование сервера TCP в большей степени напоминает работу сервера UDP; однако такую организацию работы можно изменить путем переопределения соответствующих методов в классах обработчиков запросов. Мы оставляем решение этой задачи в качестве упражнения, которое приведено в конце главы.

Кроме того, что клиент теперь действует не так, как обычно, а отчасти в противоположном направлении (поскольку в нем приходится каждый раз создавать новое соединение), единственным дополнительным отличием является то, о чем было сказано в построчном описании кода сервера: в используемом классе обработчика связь через сокет рассматривается как операции с файлом, поэтому каждый раз приходится кроме текста сообщения передавать символы, обозначающие конец строки (символ возврата каретки и символ перевода строки). Затем на сервере просто сохраняются и повторно используются те же символы, которые были переданы клиентом. После получения назад сообщения от сервера в этом сценарии клиента указанные символы отсекаются с помощью функции `strip()`, а затем вводится дополнительно только символ перевода строки, который подставляется автоматически с помощью инструкции `print`.

2.5.3. Эксплуатация сервера и клиентов TCP

Ниже приведен вывод, полученный в ходе работы клиента TCP на основе модуля `SocketServer`.

```
$ tsTclntSS.py
> 'Tis but a scratch.
[Tue Apr 18 20:55:49 2006] 'Tis but a scratch.
> Just a flesh wound.
[Tue Apr 18 20:55:56 2006] Just a flesh wound.
>
$
```

В следующем код приведен вывод сервера:

```
$ tsTservSS.py
waiting for connection...
...connected from: ('127.0.0.1', 53476)
...connected from: ('127.0.0.1', 53477)
```

Эти примеры вывода аналогичны тем, которые были получены при использовании предыдущих вариантов клиентов и серверов TCP. Однако следует отметить, что подключение к серверу происходит дважды.

2.6. *Введение в концепцию Twisted

Twisted — это полностью управляемая событиями инфраструктура организации сетей, которую можно не только использовать в готовом виде, но и полностью разрабатывать с ее помощью асинхронные сетевые приложения и протоколы. Ко времени написания данной книги инфраструктура Twisted не была введена в состав стандартной библиотеки Python, поэтому должна была загружаться и устанавливаться отдельно (соответствующая ссылка приведена в конце главы). Эта инфраструктура

предоставляет существенную поддержку и позволяет создавать полноценные системы, включая сетевые протоколы, многопоточную организацию, применение средств безопасности и аутентификации, интерактивную переписку/обмен мгновенными сообщениями, интеграцию с базами данных DBM и RDBMS (реляционная СУБД), работу в веб и Интернете, электронную почту, обработку параметров командной строки, интеграцию с инструментарием графического интерфейса пользователя и т.д.

Привлечение Twisted для реализации нашего игрушечного примера в большей степени напоминает использование кувалды для закрепления чертежной кнопки, но мы так или иначе должны с чего-то начать, и приложения, которые рассматриваются в главе, можно считать своего рода программами "hello world" в мире сетевых приложений.

Как и при использовании модуля SocketServer, основные применимые функциональные возможности Twisted заложены в классах этой инфраструктуры. Что касается примеров, рассматриваемых в главе, то для нас являются применимыми классы, находящиеся в подпакетах reactor и protocol компонента internet инфраструктуры Twisted.

2.6.1. Создание сервера TCP на основе классов reactor инфраструктуры Twisted

Код, приведенный в примере 2.10, предназначен для тех же целей, что и в примере на основе модуля SocketServer. Но вместо класса обработчика создается класс протокола и перекрываются несколько методов по такому же принципу, как и при установке обратных вызовов. Кроме того, этот пример является асинхронным. Теперь рассмотрим код сервера.

Пример 2.10. Сервер TCP отметок времени на основе классов reactor инфраструктуры Twisted (tsTservTW.py)

Это — сервер TCP отметок времени, в котором используются классы компонента internet инфраструктуры Twisted.

```

1      #!/usr/bin/env python
2
3      from twisted.internet import protocol, reactor
4      from time import ctime
5
6      PORT = 21567
7
8      class TSServProtocol(protocol.Protocol):
9          def connectionMade(self):
10             clnt = self.clnt = self.transport.getPeer().host
11             print '...connected from:', clnt
12          def dataReceived(self, data):
13             self.transport.write('%s %s' % (
14                 ctime(), data))
15
16      factory = protocol.Factory()
17      factory.protocol = TSServProtocol
18      print 'waiting for connection...'
19      reactor.listenTCP(PORT, factory)
20      reactor.run()
```


Построчное объяснение

Строки 1–6

В этих строках кода, предназначенных для настройки приложения, как обычно, содержатся инструкции импорта модулей, среди которых особого внимания заслуживают инструкции импорта подпакетов `protocol` и `reactor` компонента `twisted.internet`, а также определение константы с номером применяемого порта.

Строки 8–14

Осуществляются порождение класса `Protocol` и вызов модуля `TSServProtocol` для сервера отметок времени. Затем происходит перекрытие метода `connectionMade()`, который вызывается при поступлении от клиента запроса на установление соединения, и метода `dataReceived()`, вызываемого при передаче клиентом фрагментов данных по сети. Объект `reactor` передается в качестве параметра этого метода, что позволяет обращаться к нему сразу же, а не выполнять предварительно операцию его извлечения.

Объект экземпляра `transport` определяет возможность обмена данными с клиентом. Пример позволяет ознакомиться с тем, как этот объект используется в методе `connectionMade()` для получения информации хоста, позволяющей узнать о том, кто подключился к серверу, а также в методе `dataReceived()`, для возврата данных назад клиенту.

Строки 16–20

В последней части кода сценария сервера создается протокол `Factory`. Мы назвали этот протокол фабрикой, поскольку экземпляр данного протокола “изготавливается” каждый раз при получении входящего запроса на установление соединения. Затем с помощью методов `reactor` устанавливается приемник TCP для проверки на наличие запросов к службе; после получения запроса создается экземпляр `TSServProtocol`, позволяющий осуществлять обслуживание текущего клиента.

2.6.2. Создание клиента TCP на основе классов reactor инфраструктуры Twisted

В отличие от клиента TCP на основе модуля `SocketServer`, код клиента, приведенный в примере 2.11, не напоминает всех прочих клиентов, поскольку несет на себе явно выраженный отпечаток сетевой технологии Twisted.

Пример 2.11. Клиент TCP отметок времени на основе классов reactor инфраструктуры Twisted (`tsTcIntTW.py`)

Это — уже неоднократно применявшийся клиент TCP отметок времени, но написанный с точки зрения применения инфраструктуры Twisted.

```

1  #!/usr/bin/env python
2
3  from twisted.internet import protocol, reactor
4
5  HOST = 'localhost'
6  PORT = 21567
7
```

```

8  class TSCIntProtocol(protocol.Protocol):
9      def sendData(self):
10         data = raw_input('> ')
11         if data:
12             print '...sending %s...' % data
13             self.transport.write(data)
14         else:
15             self.transportloseConnection()
16
17     def connectionMade(self):
18         self.sendData()
19
20     def dataReceived(self, data):
21         print data
22         self.sendData()
23
24 class TSCIntFactory(protocol.ClientFactory):
25     protocol = TSCIntProtocol
26     clientConnectionLost = clientConnectionFailed = \
27         lambda self, connector, reason: reactor.stop()
28
29 reactor.connectTCP(HOST, PORT, TSCIntFactory())
30 reactor.run()

```

Построчное объяснение

Строки 1–6

В этом случае какие-либо принципиальные отличия от прежних примеров также отсутствуют, не считая того, что инструкции импорта применяются к компонентам Twisted. Во всем прочем этот код весьма напоминает сценарии клиентов, которые рассматривались выше.

Строки 8–22

Как и в коде сервера, в этом коде показано расширение класса `Protocol` путем перекрытия методов `dataReceived()` и `connectionMade()`. Эти методы в программе клиента имеют такое же назначение, как и в программе сервера. Кроме того, мы добавили свой собственный метод, предназначенный для передачи данных, и присвоили ему имя `sendData()`.

Еще раз отметим, что в этом примере рассматривается код клиента, поэтому именно в этом коде необходимо предусмотреть инициализацию диалога с сервером. Сразу после установления этого соединения мы делаем первый шаг и отправляем сообщение. Сервер отвечает, его ответ обрабатывается и отображается на экране; затем на сервер отправляется следующее сообщение.

Указанные действия продолжают в цикле до тех пор, пока не завершается соединение путем отправки пустого ответа из строки запроса. Вместо вызова метода `write()` объекта `transport` для передачи очередного сообщения серверу выполняется метод `loseConnection()`, который закрывает сокет. После того как это происходит, вызывается метод `clientConnectionLost()` фабрики и работа класса `reactor` останавливается, что приводит к завершению выполнения сценария. Работа класса `reactor` останавливается также, если метод `clientConnectionFailed()` вызывается по какой-то другой причине.

В заключительной части сценария для клиента создается фабрика, устанавливается соединение с сервером и вызываются на выполнение методы класса `reactor`. Обратите внимание на то, что в этом примере происходит порождение объекта фабрики клиента вместо передачи его классу `reactor`, как было сделано в коде сервера. Причина этого состоит в том, что не клиент, а сервер ожидает запросов от клиентов, которые требуют установить с ними соединение, поэтому именно фабрика сервера должна создавать новый объект протокола для каждого соединения. В данном случае рассматривается программа клиента, поэтому в ней создается единственный объект протокола, применяемый для подключения к серверу, фабрика которого создает соединение для работы с клиентом.

2.6.3. Эксплуатация сервера и клиентов TSP

Клиент Twisted отображает вывод, аналогичный тому, что показывают все прочие клиенты, которые рассматривались выше:

```
$ tsTclntTW.py
> Where is hope
...sending Where is hope...
[Tue Apr 18 23:53:09 2006] Where is hope
> When words fail
...sending When words fail...
[Tue Apr 18 23:53:14 2006] When words fail
>
$
```

Сервер снова и снова возвращается к единственному соединению. Инфраструктура Twisted поддерживает это соединение и не закрывает объект `transport` после каждого сообщения:

```
$ tsTservTW.py
waiting for connection...
...connected from: 127.0.0.1
```

В выводе "connection from" не отображается какая-либо прочая информация, поскольку в коде сервера запрашивается только хост и адрес из метода `getPeer()` объекта `transport` сервера.

Следует учитывать, что большинство приложений на основе Twisted намного сложнее по сравнению с теми, что показаны в примерах этого раздела. Библиотека Twisted обладает широким набором средств, но характеризуется таким уровнем сложности, что к работе с ней нельзя подходить неподготовленным.

2.7. Связанные модули

В табл. 2.4 перечислены некоторые другие модули Python, относящиеся к сетевому программированию и программированию сокетов. Модуль `select` обычно используется в сочетании с модулем `socket` для разработки приложений низкого уровня на основе сокетов. В этом модуле предусмотрена функция `select()`, которая управляет наборами объектов сокетов. Одно из наиболее полезных действий, выполняемых этой функцией, состоит в том, что она принимает набор сокетов и прослушивает его в целях определения в них наличия активных соединений. Функция `select()` блокирует набор сокетов до тех пор, пока не обнаруживается по крайней мере один сокет,

готовый для установления соединения, а когда это происходит, возвращает набор сокетов, готовых для чтения. (Эта функция может также определить, какие сокет-вызовы для записи, хотя ее применение для такой цели происходит гораздо менее часто по сравнению с указанной ранее.)

Таблица 2.4. Модули, связанные с сетевым программированием и программированием сокетов

Модуль	Описание
socket	Интерфейс организации сетей низкого уровня, который рассматривается в этом разделе
asyncore/asyncchat	Предоставляет инфраструктуру для создания сетевых приложений, которые выполняют операции с клиентами асинхронно
select	Управляет несколькими соединениями через сокет в однопоточном сетевом серверном приложении
SocketServer	Модуль высокого уровня, который предоставляет классы сервера для сетевых приложений, наряду со средствами организации многопоточности или ветвления

Модули `async*` и `SocketServer` предоставляют функциональные средства высокого уровня для создания серверов. Эти модули разработаны на основе модулей `socket` и (или) `select` и обеспечивают более быструю разработку систем “клиент–сервер”, поскольку в них уже предусмотрен весь необходимый код низкого уровня. Программисту достаточно лишь создать соответствующие базовые классы или их подклассы, после чего приступить к их использованию. Как уже было сказано выше, модуль `SocketServer` предоставляет даже функциональные возможности включения в программу сервера средств многопоточной обработки или порождения новых процессов, поэтому обеспечивается возможность большего распараллеливания обработки клиентских запросов.

Безусловно, модуль `async*` из стандартной библиотеки обеспечивает поддержку разработки только асинхронных приложений, но в предыдущем разделе представлен пакет `Twisted` сторонних разработчиков, обладающий гораздо более широкими возможностями, чем указанные выше модули, разработанные ранее. В этой главе приведен пример кода для `Twisted`, немного более сложный по сравнению с другими простейшими сценариями, но библиотека `Twisted` предоставляет гораздо более мощную и гибкую инфраструктуру и даже на текущем этапе ее развития обеспечивает реализацию большого числа протоколов. Дополнительные сведения о библиотеке `Twisted` можно узнать на посвященном ему веб-сайте (<http://twistedmatrix.com>).

Предусмотрена также еще более современная инфраструктура организации сетей, `Concurrence`, представляющая собой ядро голландской социальной сети `Hyves`. `Concurrence` — это высокопроизводительная система ввода-вывода, применяемая в сочетании с системой `libevent`, которая представляет собой систему планирования отправки обратных вызовов на основе событий низкого уровня. В системе `Concurrence` реализована асинхронная модель и используются упрощенные потоки (выполняющие обратные вызовы) по принципу управления событиями для выполнения всей обработки данных и передачи сообщений в составе межпоточковой связи. Дополнительные сведения о системе `Concurrence` можно найти по адресу:

<http://opensource.hyves.org/concurrence>

Современные сетевые инфраструктуры, как правило, создаются на основе использования одной из многих асинхронных моделей (гринлетов, генераторов и т.д.) для развертывания высокопроизводительных асинхронных серверов. Одной из целей таких инфраструктур является существенное снижение сложности асинхронного программирования, что позволяло бы пользователям разрабатывать код по принципу более знакомого, синхронного подхода.

В настоящей главе рассмотрены темы, касающиеся сетевого программирования с применением сокетов на языке Python, и было показано, как создаются заказные приложения с использованием наборов сетевых протоколов низкого уровня, таких как TCP/IP и UDP/IP. Тем читателям, которые желают разрабатывать приложения высокого уровня для веб и Интернета, настоятельно рекомендуется перейти к главе 3 или даже сразу приступить к изучению части II этой книги.

2.8. Упражнения

- 2.1. *Сокеты.* В чем состоит различие между сокетами без установления логических соединений и с установлением логических соединений?
- 2.2. *Архитектура “клиент–сервер”.* Расскажите своими словами, что означает этот термин, и приведите несколько примеров.
- 2.3. *Сокеты.* Между TCP и UDP, какой тип серверов принимает соединения и передает их, чтобы отделить сокеты для связи клиента?
- 2.4. *Клиенты.* Внесите изменения в клиентские программы для TCP (`tsTclnt.py`) и UDP (`tsUclnt.py`), чтобы имя сервера не было жестко задано в приложении. Предоставьте возможность пользователю указывать имя хоста и номер порта и используйте значения по умолчанию, только если не заданы один или оба параметра.
- 2.5. *Организация сетей и сокеты.* Реализуйте примеры программ клиентов и серверов TCP, приведенные в документации справочного руководства по библиотеке Python для модуля `socket`, и заставьте их работать. Установите сервер, а затем клиент. По следующему адресу можно также получить готовую версию этого исходного кода:

<http://docs.python.org/library/socket#example>

Вы нашли, что наш сервер слишком примитивен. Внесите изменения в программу сервера так, чтобы он мог выполнять гораздо больше действий, в частности, распознавал следующие команды:

- `date.` Сервер возвращает свою текущую отметку даты/времени, т.е. результат вызова `time.ctime()`.
- `os.` Получение информации об операционной системе (`os.name`).
- `ls.` Получение листинга текущего каталога. (Подсказка. Вызов `os.listdir()` обеспечивает получение листинга каталога, а `os.getcwd()` возвращает текущий каталог.) **Дополнительное задание.** Сервер должен принимать команду `ls dir` и возвращать листинг файлов, полученный с помощью `dir`.

Для выполнения этого задания не нужна сеть, поскольку компьютер может обмениваться данными сам с собой. Следует учитывать, что после выхода из программы сервера необходимо очищать созданную ранее привязку, прежде чем появится возможность снова вызвать сервер на выполнение. В противном случае могут появляться сообщения об ошибках со словами "port already bound" (привязка порта уже выполнена). Операционная система обычно удаляет привязку в течение 5 минут, поэтому приходится проявлять терпение.

- 2.6. *Служба дневного времени.* Используйте функцию `socket.getservbyname()`, чтобы определить номер порта для службы дневного времени, создаваемой с применением протокола UDP. Ознакомьтесь с документацией к функции `getservbyname()` для получения точных сведений о синтаксисе ее использования (т.е. выполните вызов `socket.getservbyname.__doc__`). После этого напишите приложение, которое отправляет фиктивное сообщение и ожидает ответа. Вслед за тем, как будет получен ответ от сервера, его необходимо отобразить на экране.
- 2.7. *Полудуплексная интерактивная переписка.* Создайте простую, полудуплексную программу интерактивной переписки. Под полудуплексной подразумевается то, что после создания соединения и начала работы службы только один из участников переписки может вводить свое сообщение. Другой участник должен ожидать поступления сообщения, поскольку лишь после этого перед ним появится приглашение к вводу сообщения. После отправки сообщения отправитель должен ожидать ответа для получения возможности отправить следующее сообщение. Один из участников будет находиться на стороне сервера, а второй — на стороне клиента.
- 2.8. *Дуплексная интерактивная переписка.* Доработайте свое решение предыдущего упражнения таким образом, чтобы служба интерактивной переписки стала дуплексной; это означает, что в этой службе оба абонента могут и передавать, и получать сообщения независимо друг от друга.
- 2.9. *Многопользовательская дуплексная интерактивная переписка.* Внесите дополнительные обновления в свое решение, чтобы служба интерактивной переписки стала многопользовательской.
- 2.10. *Многопользовательская, многоэкранная, дуплексная интерактивная переписка.* Теперь доработайте свою программу интерактивной переписки так, чтобы предоставляемая ею служба стала многопользовательской и многоэкранной.
- 2.11. *Веб-клиент.* Напишите программу клиента TCP, которая подключается к порту 80 вашего предпочтительного веб-сайта (удаляйте префикс "http://" и всю последующую информацию; используйте только имя хоста). После установления соединения отправляйте строку команды `GET /\n` протокола HTTP и записывайте все данные, возвращаемые сервером, в файл. (Команда `GET` осуществляет выборку веб-страницы, параметр `/file` указывает файл, который должен быть получен, а подстрока `\n` применяется для отправки команды на сервер.) Изучите содержимое полученного файла. Из чего оно состоит? Как осуществить проверку, чтобы убедиться в правильности полученных данных? (**Примечание.** Возможно, после строки команды придется вставить один или два символа перевода строки. Обычно достаточно одного.)

- 2.12.** *Сервер ждущего режима.* Создайте сервер ждущего режима. Клиент отправляет запрос для перевода его в состояние ожидания на указанное количество секунд. Сервер выдает команду от имени клиента, затем возвращает клиенту сообщения, указывающее на успешное завершение. Клиент должен переходить в состояние ожидания или простаивать в точном соответствии с указанным временем. Это простая реализация удаленного вызова процедур, при котором запрос клиента вызывает команды на другом компьютере по сети.
- 2.13.** *Сервер имен.* Спроектируйте и реализуйте сервер имен. Такой сервер отвечает за сопровождение базы данных с парами значений “имя хоста–номер порта”, к которым могут прилагаться строковые описания служб, предоставляемых соответствующими серверами. Возьмите за основу один или несколько существующих серверов и обеспечьте регистрацию их служб в подготовленном вами сервере имен. (Обратите внимание на то, что в данном случае эти серверы становятся клиентами сервера имен.)

Каждый запускаемый клиент не имеет сведений о том, где находится его искомым сервер. Эти клиенты, будучи также клиентами сервера имен, должны отправлять запрос на сервер имен с указанием того, какого рода обслуживание им требуется. Сервер имен в ответ возвращает клиенту пару значений “имя хоста–номер порта”, чтобы клиент мог затем подключиться к соответствующему серверу для обработки своего запроса.

Дополнительные задания

- 1) Предусмотрите на сервере имен кэширование для ускорения формирования ответов на часто встречающиеся запросы.
- 2) Обеспечьте возможность ведения журналов на сервере имен, чтобы можно было следить за тем, какие серверы зарегистрированы и какие службы запрашиваются клиентами.
- 3) Сервер имен должен периодически выполнять эхо-тестирование зарегистрированных хостов по их соответствующим номерам портов для проверки того, что данная служба действительно работает. В случае неоднократных неудачных проверок сервер должен быть исключен из списка служб.

Можно реализовать реальные службы на серверах, которые регистрируются в службе имен, или просто использовать фиктивные серверы (лишь подтверждающие получение запросов).

- 2.14.** *Проверка на наличие ошибок и корректный останов.* Во всех примерах кода клиентов и серверов, приведенных в этой главе, имеется общий недостаток — отсутствие проверки на наличие ошибок. Мы не рассматривали такое развитие событий, в котором пользователь вводит комбинацию клавиш <Ctrl+C> для завершения работы сервера, или <Ctrl+D> для завершения ввода данных в клиентской программе, а также не проверяли правильность данных, вводимых с помощью функции `raw_input()`, и не обрабатывали ошибки, возникающие в сети. Из-за этих недостатков слишком часто приходилось завершать работу приложений без закрытия сокетов, что может приводить к безвозвратной потере данных. Выберите одну из пар, состоящих из клиентской и серверной программ, которые рассматривались в

примерах данной главы, и введите достаточный объем средств проверки на наличие ошибок, которые позволяли бы завершать работу каждого приложения должным образом, иными словами, закрывать сетевые подключения.

- 2.15. *Асинхронная организация работы и модули SocketServer/socketserver.* Возьмите за основу один из примеров сервера TCP и примените любой из дополнительных классов для обеспечения создания асинхронного сервера. Для проверки этого сервера создайте и запустите на выполнение одновременно несколько клиентов и покажите, что сервер обслуживает запросы от всех этих клиентов, переходя от одного к другому.
- 2.16. **Расширение классов SocketServer.* Когда мы рассматривали пример с кодом сервера TCP на основе модуля SocketServer, нам пришлось внести изменения в код клиента, отличающие его от исходного базового клиента TCP, поскольку класс SocketServer не поддерживает соединение между запросами.
- а) Создайте подклассы классов TCPServer и StreamRequestHandler и перепроектируйте сервер так, чтобы он сопровождал и использовал отдельное соединение для каждого клиента (а не создавал соединение для каждого запроса).
 - б) Объедините свое решение предыдущего упражнения с решением для пункта (а) текущего упражнения, чтобы обеспечить параллельное обслуживание нескольких клиентов.
- 2.17. **Асинхронные системы.* Исследуйте по меньшей мере пять различных асинхронных систем на основе языка Python. Сделайте выбор среди Twisted, Greenlets, Tornado, Diesel, Concurrency, Eventlet, Gevent и т.д. Опишите, что представляет каждая из них, распределите их по категориям, найдите подобию и различия, а затем подготовьте несколько образцов кода для демонстрации.

ГЛАВА

3

Программирование интернет-клиентов

В этой главе...

- Что такое интернет-клиенты
- Передача файлов
- Сетевые новости
- Электронная почта
- Связанные модули

Что-либо удалить из Интернета невозможно. Это равносильно попытке извлечь мочу из воды в плавательном бассейне. После того как она туда попала, там и остается.
Джо Гаррелли (Joe Garrelli), март 1996 г. (словами своего персонажа Джо Рогана из телевизионной передачи NewsRadio).

В главе 2 рассматривались протоколы сетевого взаимодействия низкого уровня на основе сокетов. Организация сетей такого типа является основой большинства протоколов “клиент–сервер”, применяемых в наши дни в Интернете. В их число входят протоколы передачи файлов (FTP и т.д.), чтения групп новостей Usenet (Network News Transfer Protocol — NNTP), передачи электронной почты (SMTP), загрузки электронной почты с сервера (POP3, IMAP) и т.д. Эти протоколы действуют во многом подобно тому, что было показано на примерах взаимодействия “клиент–сервер” в главе 2. Единственное различие состоит в том, что теперь мы возьмем за основу протоколы низкого уровня, такие как TCP/IP, и создадим на базе их более новые, более специализированные протоколы для реализации служб высокого уровня, подобных перечисленным выше.

3.1. Что такое интернет-клиенты

Прежде чем приступить к рассмотрению этих протоколов, необходимо найти ответ на вопрос: что такое интернет-клиент? Чтобы ответить на этот вопрос, упрощенно представим Интернет как пространство, где происходит обмен данными, а этот обмен осуществляется между теми, кто предоставляет услуги, и пользователями этих услуг. Пара “производитель–потребитель” часто упоминается во многих сферах производства и экономики (хотя, когда дело касается программного обеспечения, эти понятия в основном применяются к участникам обмена данными в операционных системах). Производителями являются серверы, предоставляющие услуги, а потребителями становятся клиенты, которые потребляют необходимые им ресурсы. Когда речь идет о каких-то конкретных услугах, обычно говорят только об одном сервере (который может также именоваться *процессом*, *хостом* и т.д.) и нескольких клиентах-потребителях. Выше в данной книге уже рассматривалась модель “клиент–сервер”, и эта модель хорошо подходит для раскрытия темы настоящей главы, хотя нам вряд ли придется создавать интернет-клиенты на основе операций с сокетами низкого уровня.

В этой главе мы рассмотрим несколько из перечисленных выше протоколов Интернета и для каждого из них создадим клиенты. После изучения этой темы читатель может понять, насколько подобны друг другу интерфейсы прикладного программирования (API-интерфейсы) всех этих протоколов. Так было задумано с самого начала, поскольку единообразие интерфейсов предоставляет важные преимущества, в частности, позволяет создать практически применимые клиенты для этих и других протоколов Интернета. В этой главе мы достаточно кратко рассмотрим три определенных протокола, но после знакомства с ней читатель будет чувствовать себя достаточно уверенно для того, чтобы самостоятельно заняться написанием клиентских программ практически для *любого* протокола Интернета.

3.2. Передача файлов

3.2.1. Протоколы Интернета для передачи файлов

Одной из наиболее важных областей применения Интернета является обмен файлами. Передача и прием файлов происходят во всей сети, не прерываясь ни на секунду. Для доставки файлов из одной точки в другую по Интернету было разработано много протоколов, но наиболее широко применяемыми стали FTP (File Transfer Protocol), UUCP (Unix-to-Unix Copy Protocol), а также, безусловно, HTTP (Hypertext Transfer Protocol) в составе средств веб. Необходимо также отметить команду дистанционного копирования файлов `rcp` (Unix), а также более безопасные и гибкие современные аналоги этой команды, `scp` и `rsync`.

Протоколы HTTP, FTP и команды `scp/rsync` все еще весьма широко применяются для передачи файлов и в наши дни. Протокол HTTP в основном используется для загрузки файлов на основе веб и для доступа к веб-службам. При использовании этого протокола обычно не требуется, чтобы клиенты имели имя входа и (или) пароль на хосте сервера для получения документов или услуг. Большая часть всех запросов на передачу файлов HTTP состоит из запросов на поиск веб-страниц (загрузку файлов).

С другой стороны, при использовании команд `scp` и `rsync` требуется, чтобы пользователь регистрировался на хосте сервера. Клиенты должны проходить проверку подлинности до того, как появится возможность осуществлять передачу файлов, а с файлами могут производиться операции отправки (передачи) или получения (загрузки). Наконец, предусмотрен такой протокол, как FTP. Подобно `scp/rsync`, FTP может использоваться для передачи или загрузки файлов и, по аналогии с `scp/rsync`, в нем применяются концепции имен пользователей и паролей многопользовательской системы Unix. Клиенты FTP должны использовать имя входа/пароль существующих пользователей, но протокол FTP позволяет также применять анонимные учетные записи. Приступим к более подробному рассмотрению протокола FTP.

3.2.2. Протокол передачи файлов

Протокол передачи файлов (File Transfer Protocol — FTP) был разработан покойным ныне Джоном Постелом (Jon Postel) и Джойсом Рейнолдсом (Joyce Reynolds) и представлен в так называемом “Предложении к обсуждению в Интернете” (Request for Comment — RFC) под номером 959, который был опубликован в октябре 1985 года. Этот протокол в основном используется для загрузки общедоступных файлов от имени анонимного пользователя. Кроме того, протокол FTP может использоваться для передачи файлов между двумя компьютерами, особенно если для хранения или архивирования файлов применяется система на основе Unix, а для работы — настольный компьютер или ноутбук. До того как веб-доступ получил широкое распространение, протокол FTP служил в качестве одного из основных методов передачи файлов по Интернету, а также был практически единственным способом загрузки программного обеспечения и (или) исходного кода.

Как уже было указано выше, как правило, необходимо иметь имя входа/пароль для получения доступа к удаленному хосту, на котором работает FTP-сервер. Исключением являются анонимные учетные записи, которые предназначены для загрузок, осуществляемых пользователями с правами гостя. Это позволяет загружать файлы клиентам, не имеющим учетных записей. Для предоставления такой возможности

администратор сервера должен выполнить настройку FTP-сервера на работу с анонимными учетными записями. В таких случаях имя входа незарегистрированного пользователя обозначается как `anonymous`, а паролем, как правило, становится адрес электронной почты клиента. Такая организация работы применяется для обеспечения входа в систему широкого круга пользователей и доступа к каталогам, предназначенным для общего пользования, в отличие от такого функционирования, при котором применяются регистрация в системе и передача файлов для определенных пользователей. В этом случае также, как правило, поддерживается гораздо более узкий перечень команд протокола FTP по сравнению с теми командами, которые могут применять реальные пользователи.

1. Среда применения протокола схематически показана на рис. 3.1 и функционирует следующим образом.
2. Клиент посылает запрос на FTP-сервер, работающий на удаленном узле.
3. Клиент регистрируется, указывая имя пользователя и пароль (или задавая имя `anonymous` и адрес электронной почты).
4. Клиент осуществляет различные операции передачи файлов или отправляет информационные запросы.
5. Клиент завершает свою работу путем выхода из системы удаленного хоста и FTP-сервера.

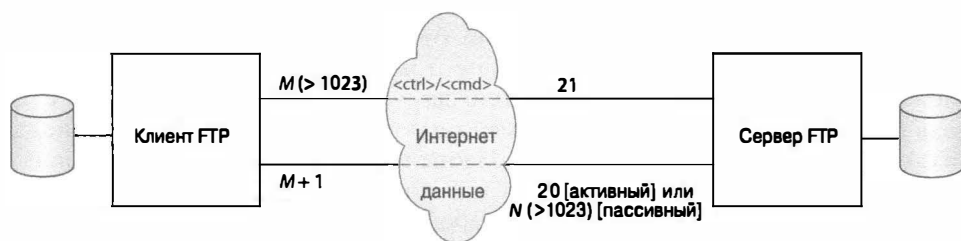


Рис. 3.1. Схема расположения клиента и сервера FTP в Интернете. Клиент и сервер взаимодействуют путем передачи команд протокола FTP через порт управления; данные передаются через порт данных

Разумеется, это — весьма упрощенное описание работы FTP. Иногда возникают такие обстоятельства, что приходится прерывать операцию передачи файлов до ее завершения. К ним относится отключение от сети вследствие отказа одного из двух хостов или возникновение каких-то других проблем, обусловленных состоянием сетевого соединения. Работа с клиентом может быть также прервана из-за его продолжительного пребывания в неактивном состоянии. Как правило, выход по тайм-ауту из FTP-соединения производится после простоя в течение 15 минут (900 секунд).

При повседневной работе это не имеет большого значения, но следует знать, что в протоколе FTP используется только протокол TCP (см. главу 2); в нем никоим образом не применяется протокол UDP. Кроме того, протокол FTP можно рассматривать как не совсем типичный пример сетевого программного обеспечения “клиент-сервер”, поскольку в нем клиенты и серверы используют для связи не один, а два сокета: один из них открывается в порту управления, или порту передачи команд (порт 21), а другой — в порту данных (в некоторых случаях, порт 20).

Выше было сказано “в некоторых случаях”, поскольку применение порта данных зависит от того, какой из двух возможных режимов FTP используется: активный или пассивный. Порт данных сервера с номером 20 применяется только для активного режима. После того как сервер устанавливает порт 20 в качестве порта данных, происходит “активное” инициирование сервером соединения с портом данных клиента. Если же применяется пассивный режим, то сервер открывает порт данных со случайно выбранным номером, после чего лишь сообщает клиенту этот номер порта. Инициировать подключение к порту данных сервера должен клиент. Вполне очевидно, что в этом режиме FTP-сервер выполняет более пассивную роль в процедуре установления подключения к данным. Наконец, отметим, что теперь организована поддержка для нового, расширенного пассивного режима, который предназначен для работы с адресами интернет-протокола версии 6 (IPv6); см. документ RFC 2428.

В языке Python предусмотрена поддержка большинства протоколов Интернета, включая FTP. Сведения о других поддерживаемых клиентских библиотеках можно найти по адресу <http://docs.python.org/library/internet>. Теперь перейдем к рассмотрению того, насколько просто решается задача создания интернет-клиента с помощью сценария на языке Python.

3.2.3. Язык Python и протокол FTP

Прежде всего займемся созданием клиента FTP с помощью языка Python. Сведения, приведенные в предыдущем разделе, позволяют приступить к этой задаче с полной уверенностью. Необходимо лишь дополнительно обеспечить импорт соответствующего модуля Python и предусмотреть в сценарии Python вызовы необходимых функций. Вначале кратко рассмотрим, как действует протокол.

1. Подключение к серверу.
2. Вход в систему.
3. Выполнение запросов к службе (и при благоприятном стечении обстоятельств получение ответов).
4. Завершение работы.

При использовании поддержки протокола FTP, предусмотренной в языке Python, достаточно лишь импортировать модуль `ftplib` и создать экземпляр класса `ftplib.FTP`. Все действия, предусмотренные протоколом FTP, включая регистрацию, передачу файлов и выход из системы, будут осуществляться с использованием заранее созданного объекта.

Ниже приведен псевдокод Python, который станет прообразом будущей программы.

```
from ftplib import FTP
f = FTP('some.ftp.server')
f.login('anonymous', 'your@email.address')
:
f.quit()
```

Вскоре после этого мы рассмотрим реальный пример, но вначале ознакомимся с методами класса `ftplib.FTP`, которые чаще всего используются в коде программы.

3.2.4. Методы класса `ftplib.FTP`

Наиболее широко применяемые методы класса `ftplib.FTP` приведены в табл. 3.1. Этот список не является исчерпывающим (для ознакомления со всеми методами необходимо изучить исходный код самого класса или обратиться к документации), но представленные здесь методы составляют основу API-интерфейса программирования клиента FTP на языке Python. Иными словами, в действительности работа программиста чаще всего сводится к использованию указанных методов, тогда как прочие предназначены в основном для выполнения вспомогательных или административных задач либо используются другими методами API-интерфейса.

Таблица 3.1. Методы объектов класса FTP

Метод	Описание
<code>login(user='anonymous', passwd='', acct='')</code>	Обеспечивает регистрацию на FTP-сервере; все параметры этого метода являются необязательными
<code>pwd()</code>	Показывает текущий рабочий каталог
<code>cwd(path)</code>	Позволяет перейти из текущего рабочего каталога в каталог <i>path</i>
<code>dir([path[,...[,cb]])</code>	Отображает листинг каталога <i>path</i> ; может быть передан необязательный обратный вызов <i>cb</i> для <code>retrlines()</code>
<code>nlst([path[,...]])</code>	Аналогичен <code>dir()</code> , но возвращает, а не выводит список имен файлов
<code>retrlines(cmd [, cb])</code>	Загружает текстовый файл по указанной команде <i>cmd</i> протокола FTP, например <code>RETR filename</code> . Может быть передан необязательный обратный вызов <i>cb</i> для обработки каждой строки файла
<code>retrbinary(cmd, cb [, bs=8192[, ra]])</code>	Аналогичен <code>retrlines()</code> , за исключением того, что применяется к двоичным файлам. Может быть передан необязательный обратный вызов <i>cb</i> для обработки каждого необходимого загруженного блока (размер <i>bs</i> по умолчанию принят равным 8 Кбайт)
<code>storlines(cmd, f)</code>	Передает текстовый файл по указанной команде <i>cmd</i> протокола FTP, например <code>STOR filename</code> . Требуется объект открытого файла <i>f</i>
<code>storbinary(cmd, f[, bs=8192])</code>	Аналогичен <code>storlines()</code> , но применяется к двоичным файлам. Требуется объект открытого файла <i>f</i> , размер блока передачи <i>bs</i> по умолчанию принят равным 8 Кбайт
<code>rename(old, new)</code>	Переименовывает файл <i>old</i> на удаленном хосте в файл <i>new</i>
<code>delete(path)</code>	Удаляет файл <i>file</i> на удаленном хосте, находящийся в каталоге <i>path</i>
<code>mkd(directory)</code>	Создает каталог <i>directory</i> на удаленном хосте
<code>rmd(directory)</code>	Удаляет каталог <i>directory</i> на удаленном хосте
<code>quit()</code>	Закрывает соединение и завершает работу

При выполнении обычной последовательности операций протокола FTP чаще всего применяются методы `login()`, `cwd()`, `dir()`, `pwd()`, `stor*` и `retr*` и `quit()`. На практике может возникнуть необходимость в использовании других методов объекта FTP, не приведенных в этой таблице. Для ознакомления с более подробными сведениями об объектах FTP обратитесь к документации Python, находящейся по адресу <http://docs.python.org/library/ftplib#ftp-objects>.

3.2.5. Пример программы для работы с протоколом FTP в интерактивном режиме

В определенных случаях с помощью языка Python можно настолько просто организовать работу с протоколом FTP, что для этого не потребуется даже писать сценарий. Достаточно лишь открыть приглашение интерактивного интерпретатора, после чего выполнять необходимые действия и получать результаты в режиме реального времени. Ниже приведен простой пример сеанса, проведенного несколько лет тому назад, когда еще по адресу `python.org` работал FTP-сервер. В наши дни этот сеанс больше не может быть повторен, поэтому является лишь иллюстрацией того, как проводится работа с действующим FTP-сервером.

```
>>> from ftplib import FTP
>>> f = FTP('ftp.python.org')
>>> f.login('anonymous', 'guido@python.org')
'230 Guest login ok, access restrictions apply.'
>>> f.dir()
total 38
drwxrwxr-x 10 1075      4127      512 May 17  2000 .
drwxrwxr-x 10 1075      4127      512 May 17  2000 ..
drwxr-xr-x  3 root      wheel      512 May 19  1998 bin
drwxr-sr-x  3 root      1400      512 Jun  9  1997 dev
drwxr-xr-x  3 root      wheel      512 May 19  1998 etc
lrwxrwxrwx  1 root      bin        7 Jun 29  1999 lib -> usr/lib
-r--r--r--  1 guido     4127        52 Mar 24  2000 motd
drwxrwsr-x  8 1122      4127      512 May 17  2000 pub
drwxr-xr-x  5 root      wheel      512 May 19  1998 usr
>>> f.retrlines('RETR motd')
Sun Microsystems Inc.      SunOS 5.6      Generic August 1997
'226 Transfer complete.
>>> f.quit()
'221 Goodbye.'
```

3.2.6. Пример клиентской программы FTP

Как уже было сказано, специально разрабатывать сценарий нет необходимости, поскольку можно выполнить всю работу в интерактивном режиме и не терять времени на написание кода. В действительности применение сценария может принести большую пользу. Например, предположим, что должен быть создан код, с помощью которого всегда можно загрузить новейшую копию программы Bugzilla с веб-сайта Mozilla. Предложенный нами вариант решения этой задачи приведен в примере 3.1. Здесь была предпринята попытка создать приложение, но и в этом случае все необходимые действия можно было бы выполнить в интерактивном режиме. В рассматриваемом приложении используется библиотека FTP для загрузки файла и включены некоторые средства контроля ошибок.

Пример 3.1. Программа загрузки по FTP (`getLatestFTP.py`)

Эта программа используется для загрузки новейшей версии файла с веб-сайта. Читатель может внести в нее изменения, чтобы она могла применяться для загрузки необходимых ему приложений.

```
1  #!/usr/bin/env python
2
3  import ftplib
4  import os
5  import socket
6
7  HOST = 'ftp.mozilla.org'
8  DIRN = 'pub/mozilla.org/webtools'
9  FILE = 'bugzilla-IATEST.tar.gz'
10
11 def main():
12     try:
13         f = ftplib.FTP(HOST)
14     except (socket.error, socket.gaierror) as e:
15         print 'ERROR: cannot reach "%s"' % HOST
16         return
17     print '*** Connected to host "%s"' % HOST
18
19     try:
20         f.login()
21     except ftplib.error_perm:
22         print 'ERROR: cannot login anonymously'
23         f.quit()
24         return
25     print '*** Logged in as "anonymous"'
26
27     try:
28         f.cwd(DIRN)
29     except ftplib.error_perm:
30         print 'ERROR: cannot CD to "%s"' % DIRN
31         f.quit()
32         return
33     print '*** Changed to "%s" folder' % DIRN
34
35     try:
36         f.retrbinary('RETR %s' % FILE,
37                     open(FILE, 'wb').write)
38     except ftplib.error_perm:
39         print 'ERROR: cannot read file "%s"' % FILE
40         os.unlink(FILE)
41     else:
42         print '*** Downloaded "%s" to CWD' % FILE
43     f.quit()
44     return
45
46 if __name__ == '__main__':
47     main()
```

Следует учитывать то, что этот сценарий не автоматизирован, поэтому пользователю самому придется взять на себя задачу вызова сценария каждый раз, когда требуется выполнение загрузки. Если же для работы применяется система на основе Unix, то можно установить задание cron для автоматизации вызова сценария. Еще один вопрос, требующий рассмотрения, состоит в том, что сценарий перестанет нормально работать после изменения на хосте имен файлов или каталогов.

Если во время выполнения сценария не происходят какие-то ошибки, то формируется следующий вывод:

```
$ getLatestFTP.py
*** Connected to host "ftp.mozilla.org"
*** Logged in as "anonymous"
*** Changed to "pub/mozilla.org/webtools" folder
*** Downloaded "bugzilla-LATEST.tar.gz" to CWD
$
```

Построчное объяснение

Строки 1–9

В начальных строках кода осуществляется импорт необходимых модулей (главным образом для перехвата объектов исключений) и задается несколько констант.

Строки 11–44

В функции `main()` выполняются следующие шаги, необходимые для решения поставленной задачи: создание объекта FTP и осуществление попытки подключения к серверу FTP (строки 12–17), а в случае любого отказа происходит возврат (и завершение работы). В этом сценарии предпринимается попытка зарегистрироваться в качестве пользователя `anonymous`, а в случае неудачи происходит аварийное завершение (строки 19–25). Следующий шаг состоит в переходе в каталог искомого дистрибутива (строки 27–33), после чего выполняется попытка загрузить файл (строки 35–44).

2.6

В строке 14 показано применение обработчика исключений. Аналогичный код в этой книге применяется везде, где требуется сохранение экземпляра исключения (в данном случае `e`). Если применяется Python 2.5 или более ранняя версия, то необходимо заменить ключевое слово `as` запятой, поскольку этот новый синтаксис был введен (но не стал обязательным) в версии 2.6 для упрощения перехода к версиям 3.x. В версии Python 3 распознается только новый синтаксис, который показан в строке 14.

В строках 35–36 происходит передача обратного вызова в функцию `retrbinary()`, которая должна быть выполнена применительно к каждому блоку загруженных двоичных данных. Для записи на диск локальной версии файла применяется метод `write()` предварительно созданного объекта файла. При этом мы рассчитываем на то, что интерпретатор Python корректно закроет наш файл после завершения передачи данных и не произойдет потеря каких-либо данных. При таком подходе программа становится проще, но такой стиль является нежелательным, поскольку программист должен сам брать на себя ответственность за освобождение непосредственно выделенных ресурсов, а не рассчитывать на то, что это будет сделано в коде, написанном другим программистом. В этом случае необходимо сохранить объект открытого файла в переменной, допустим, `loc`, а затем передать `loc.write` в вызове метода `ftp.retrbinary()`.

После завершения передачи необходимо вызвать метод `loc.close()`. Если по какой-то причине не удастся сохранить файл, то мы обязаны удалить пустой файл, чтобы предотвратить загромождение файловой системы (строка 40). Вызов метода `os.unlink(FILE)` должен сопровождаться применением определенных средств контроля ошибок на тот случай, если файл не существует. Наконец, для предотвращения

ненужного вызова еще одной пары строк (строки 43-44), в которых осуществляются закрытие FTP-соединения и возврат, используется конструкция `else` (строки 35-42).

Строки 46-47

Обычно аналогичные меры предусматриваются при выполнении любого автономного сценария.

3.2.7. Другие особенности протокола FTP

2.1

Язык Python поддерживает и активный, и пассивные режимы. Однако следует учитывать, что в версии Python 2.0 и предыдущих версиях пассивный режим был отключен по умолчанию; а начиная с версии Python 2.1 и во всех последующих версиях этот режим задан по умолчанию.

Ниже приведен список типичных клиентов FTP.

- **Пример клиентской программы командной строки.** В подобных случаях передача по FTP выполняется с помощью клиентской программы FTP, такой как `/bin/ftp` или `NcFTP`, которая позволяет пользователям выполнять операции протокола FTP в интерактивном режиме с помощью командной строки.
- **Пример клиентской программы графического интерфейса пользователя.** Это приложение аналогично по своему назначению клиентской программе командной строки, за исключением того, что оно представляет собой приложение с графическим интерфейсом пользователя, такое как `WS_FTP`, `Filezilla`, `CuteFTP`, `Fetch` или `SmartFTP`.
- **Веб-браузер.** Большинство веб-браузеров (также относящихся к категории клиентов) позволяют не только работать по протоколу HTTP, но и осуществлять обмен данными по протоколу FTP. Первое ключевое слово (директива) в указателе URL/URI обозначает протокол, например `"http://blahblah"`. В данном случае оно служит для браузера указанием использовать протокол HTTP в качестве средства передачи данных с указанного веб-сайта. Чтобы обеспечить использование браузера для передачи данных по протоколу FTP, достаточно указать соответствующий протокол, например `"ftp://blahblah"`. Остальная часть URL в значительной степени напоминает ту, которая применяется при работе по протоколу HTTP. (Разумеется, вместо `"blahblah"` в URL должно быть указано значение в формате `"хост/путь?атрибуты"`, которое следует за директивой с указанием протокола, `"ftp://"`.) Если требуется регистрация, то пользователь может указать учетную запись и пароль (в виде открытого текста) непосредственно в указателе URL, например `ftp://user:passwd@host/path?attr1=val1&attr2=val2...`
- **Определяемое пользователем приложение:** программа, написанная пользователем, в которой применяется протокол FTP для передачи файлов. Как правило, пользователю не предоставляется возможность непосредственно обращаться к серверу, поскольку должны быть созданы конкретные приложения для выполнения определенных задач.

Язык Python позволяет создавать клиенты всех четырех типов. В приведенном выше примере для создания определяемого пользователем приложения применялась библиотека `ftplib`, но можно также создать интерактивное приложение

командной строки. Кроме того, можно воспользоваться инструментарием создания графического интерфейса пользователя, таким как Tk, wxWidgets, GTK+, Q T, MFC и даже Swing (импортировав соответствующие модули интерфейса Python или Jython), и создать полноценное приложение с графическим интерфейсом пользователя после отладки разработанного самостоятельно кода клиента командной строки. Наконец, для интерпретации указателей URL FTP и осуществления передачи по FTP можно воспользоваться модулем `urllib` языка Python. При этом, по сути, модуль `urllib` импортирует и использует модуль `ftplib`, а это означает, что `urllib` выступает в роли клиента по отношению к `ftplib`.

Протокол FTP может не только применяться для загрузки кода клиентских приложений в целях построения и (или) использования этих приложений, но и служить основой для организации повседневной работы, которая предусматривает перемещение файлов между системами. Например, предположим, что в задачу инженера или системного администратора входит обеспечение передачи файлов. На первый взгляд, следует использовать команды `scp` или `rsync`, которые позволяют пересекать границу между локальной сетью и Интернетом или доставлять файлы на внешний сервер, достижимый с клиентского компьютера. Однако если приходится таким образом перемещать по защищенной сети чрезвычайно крупные журналы или файлы базы данных от одного компьютера локальной сети к другому, то возникает необходимость преодолевать существенные сложности: обеспечивать безопасность, шифрование, упаковку/распаковку и т.д. Вместо этого достаточно лишь создать простое приложение FTP, позволяющее быстро и безопасно перемещать файлы в часы наименьшей загрузки, а язык Python подходит для этого наилучшим образом!

С дополнительными сведениями о протоколе FTP можно ознакомиться в определении/спецификации протокола FTP (документ RFC 959) по адресу <http://tools.ietf.org/html/rfc959>, а также посетить веб-страницу www.networksorcery.com/enp/protocol/ftp.htm. Тематика, связанная с протоколом FTP, рассматривается также в документах RFC с номерами 2228, 2389, 2428, 2577, 2640 и 4217. Чтобы больше узнать о поддержке протокола FTP в языке Python, можно обратиться по адресу <http://docs.python.org/library/ftplib>.

3.3. Сетевые новости

3.3.1. Usenet и группы новостей

Система новостей Usenet — это глобальная архивная доска объявлений. Группы новостей имеются почти по любой теме, от поэзии до политики, от лингвистики до языков программирования, от программного обеспечения до аппаратных средств, от выращивания растений до кулинарии, от науки до осведомления о возможностях занятости, занятия музыкой и магией, расставания с теми, кого не любишь, или поиска тех, кого полюбишь. Группы новостей могут быть общими и международными или целенаправленными и рассчитанными на конкретный географический регион.

Вся система новостей представляет собой большую глобальную сеть компьютеров, которые участвуют в распространении поступлений Usenet. После того как пользователь передает сообщение на свой локальный компьютер Usenet, это сообщение распространяется по другим смежным компьютерам Usenet, затем по соседним по отношению к ним компьютерам и т.д. Это происходит до тех пор, пока новое поступление не обойдет весь мир и не окажется в распоряжении каждого желающего.

Новые поступления сохраняются в Usenet в течение конечного промежутка времени, который определяется системным администратором Usenet или задается в самом поступлении в виде даты и времени истечения срока действия.

В каждой системе имеется список групп новостей, на которые в ней оформлена подписка, и осуществляется прием только интересующих поступлений, а не всех групп новостей, которые могли быть архивированы на сервере. Характер услуг, предоставляемых служб Usenet, зависит от используемого поставщика групп новостей. Многие из этих служб открыты для широкой публики, а другие разрешают доступ только конкретным пользователям, таким как подписчики на платные рассылки, студенты определенного университета и т.д. При этом применение имени входа и пароля может быть необязательным, согласно конфигурации, установленной системным администратором Usenet. Администратор может настраивать также такой параметр, как возможность загружать только поступления.

Usenet утратила свое значение в качестве глобальной доски объявлений и была в значительной степени заменена оперативными форумами. Тем не менее для нас имеет смысл рассмотреть работу Usenet в этой главе в основном с точки зрения применяемого в этой сети протокола.

В старых версиях Usenet в качестве механизма сетевого транспорта использовался протокол UUCP, а начиная с середины 1980-х годов, после того как основная часть сетевого трафика стала переводиться на протоколы TCP/IP, был создан другой протокол. Этот новый протокол мы рассмотрим в следующую очередь.

3.3.2. Протокол передачи сетевых новостей

Протокол, с помощью которого пользователи могут загружать поступления (или статьи) групп новостей или сами публиковать новые статьи, именуется NNTP (Network News Transfer Protocol — протокол передачи сетевых новостей). Он был разработан Брайеном Кантором (Brian Kantor) из Калифорнийского университета, г. Сан-Диего, и Филом Лапли (Phil Lapsley) из Калифорнийского университета, г. Беркли, и опубликован в документе RFC 977 в феврале 1986 г. Обновленная версия протокола была опубликована в документе RFC 2980 в октябре 2000 г.

Протокол NNTP представляет собой еще один пример реализации архитектуры “клиент–сервер” и функционирует по принципу, аналогичному FTP, но является намного более простым. В протоколе NNTP вместо целого набора номеров портов для входа в систему, передачи данных и управления (как в протоколе FTP) используется для связи только один стандартный порт, 119. Серверу передается запрос, а сервер возвращает соответствующий ответ, как показано на рис. 3.2.

3.3.3. Язык Python и протокол NNTP

На основе своего знакомства с языком Python и поддержкой в нем протокола FTP в предыдущем разделе читатель может предположить, что в этом языке предусмотрена библиотека `nnplib` и имеется класс `nnplib.NNTP`, экземпляр которого необходимо создать в программе, и будет прав. Как и в случае с протоколом FTP, достаточно лишь импортировать необходимый модуль Python и выполнить соответствующие вызовы в программе на языке Python. Вначале кратко рассмотрим, как действует протокол.

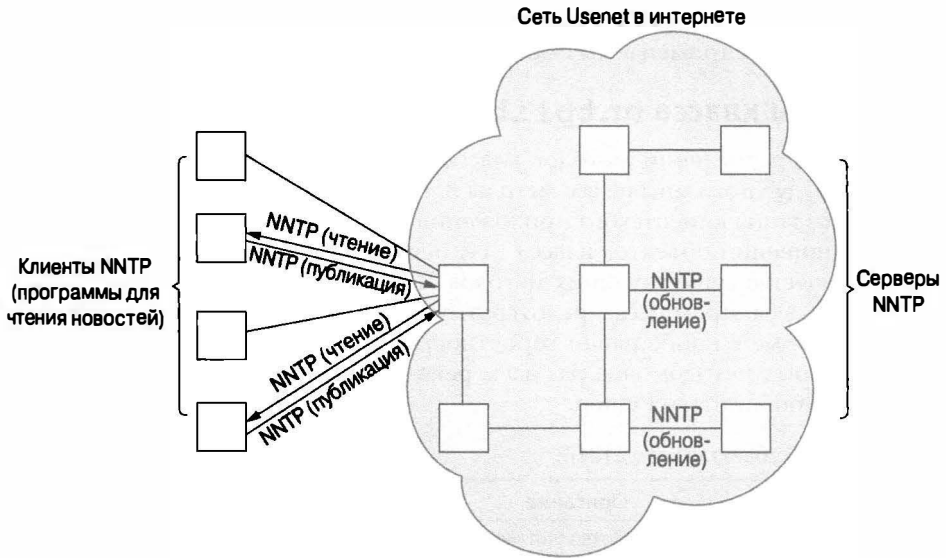


Рис. 3.2. Схема расположения клиентов и серверов NNTP в Интернете. Клиенты главным образом читают новости, но могут также их публиковать. Затем статьи распространяются в ходе того, как серверы обновляют содержимое друг друга

1. Подключение к серверу.
2. Вход в систему (если это предусмотрено).
3. Выполнение запросов на обслуживание.
4. Завершение работы.

По-видимому, такая последовательность действий становится привычной. Так и должно быть, поскольку работа с протоколом NNTP осуществляется полностью аналогично протоколу FTP. Единственное отличие состоит в том, что шаг регистрации является необязательным и его применение зависит от того, как настроен сервер NNTP.

Ниже приведен фрагмент псевдокода Python, с которого можно начать знакомство с этим протоколом.

```
from nntplib import NNTP
n = NNTP('your.nntp.server')
r, c, f, l, g = n.group('comp.lang.python')
...
n.quit()
```

Как правило, после входа в систему необходимо выбрать интересующую группу новостей и вызвать метод `group()`. Этот метод возвращает ответ от сервера с указанием количества статей с различными номерами, идентификаторов первой и последней статей, а также, дополнительно, снова показывает имя группы. После получения этой информации можно предпринять необходимые действия, например, выполнить прокрутку списка статей и просмотреть статьи, загрузить целые поступления (с заголовками и текстами статей) или, возможно, отправить в группу новостей собственную статью.

Прежде чем перейти к рассмотрению реального примера, рассмотрим некоторые из наиболее широко применяемых методов класса `nntplib.NNTP`.

3.3.4. Методы класса `nntplib.NNTP`

Как и при перечислении методов класса `ftplib.FTP` в предыдущем разделе, в этом разделе будут показаны не все методы `nntplib.NNTP`, а только те, которые необходимы для создания клиентского приложения NNTP.

Как и в отношении объектов класса `FTP`, перечисленных в табл. 3.1, можно отметить, что количество существующих методов объектов класса `NNTP` больше, чем описано в табл. 3.2. Мы стремимся предотвратить путаницу, поэтому приводим только те методы, которые с наибольшей вероятностью будет использовать читатель. Что касается остальных методов, еще раз даем рекомендацию обратиться к справочному руководству по библиотеке Python.

Таблица 3.2. Методы объектов класса `NNTP`

Метод	Описание
<code>group(name)</code>	Позволяет указать имя группы новостей и получить кортеж (<code>rsp, ct, fst, lst, group</code>): ответ сервера, количество статей, номера первой и последней статей и имя группы. Все эти параметры заданы в формате (<code>name == group</code>)
<code>xhdr(hdr, artrg[, ofile])</code>	Возвращает список заголовков <code>hdr</code> для диапазона статей <code>artrg</code> (в формате "первая-последняя") или выводит данные в файл <code>ofile</code>
<code>body(id [, ofile])</code>	Позволяет получить текст статьи по соответствующему значению <code>id</code> , которое представляет собой либо идентификатор сообщения (заключенный в угловые скобки <code><...></code>), либо номер статьи, представленный в виде строки. Возвращает кортеж (<code>rsp, anum, mid, data</code>): ответ сервера, номер статьи в виде строки, идентификатор сообщения (заключенный в угловые скобки <code><...></code>) и список строк статьи, или же выводит данные <code>data</code> в файл <code>ofile</code>
<code>head(id)</code>	Аналогичен методу <code>body()</code> . Возвращает такой же кортеж, с тем отличием, что строки содержат только заголовки статей
<code>article(id)</code>	Также аналогичен методу <code>body()</code> . Возвращается тот же кортеж, но с тем отличием, что строки содержат и заголовки, и текст статей
<code>stat(id)</code>	Задаёт "указатель" на статью, равный <code>id</code> (идентификатор сообщения или номер статьи, как указано выше). Возвращает кортеж, как в вызове <code>body(rsp, anum, mid)</code> , но не содержит никаких данных из статьи
<code>next()</code>	При использовании с методом <code>stat()</code> перемещает указатель статьи на следующую статью и возвращает аналогичный кортеж
<code>last()</code>	Также используется вместе с методом <code>stat()</code> . Перемещает указатель статьи на последнюю статью и возвращает аналогичный кортеж
<code>post(ufile)</code>	Передаёт на сервер данные из файлового объекта <code>ufile</code> (с использованием вызова <code>ufile.readline()</code>) и помещает статью в текущую группу новостей
<code>quit()</code>	Закрывает соединение и завершает работу

3.3.5. Пример использования протокола NNTP в интерактивном режиме

Ниже приведен интерактивный пример того, как следует использовать библиотеку NNTP языка Python. Очевидно, что этот пример должен выглядеть аналогично интерактивному примеру для FTP. (Адреса электронной почты изменены в целях обеспечения конфиденциальности.)

После подключения к группе метод `group()` возвращает пятиэлементный кортеж, как описано в табл. 3.2.

```
>>> from nntplib import NNTP
>>> n = NNTP('your.nntp.server')
>>> rsp, ct, fst, lst, grp = n.group('comp.lang.python')
>>> rsp, anum, mid, data = n.article('110457')
>>> for eachLine in data:
...     print eachLine
From: "Alex Martelli" <alex@...>
Subject: Re: Rounding Question
Date: Wed, 21 Feb 2001 17:05:36 +0100
"Remco Gerlich" <remco@...> wrote:
> Jacob Kaplan-Moss <jacob@...> wrote in comp.lang.python:
>> So I've got a number between 40 and 130 that I want to round up to
>> the nearest 10. That is:
>>
>> 40 --> 40, 41 --> 50, ..., 49 --> 50, 50 --> 50, 51 --> 60
> Rounding like this is the same as adding 5 to the number and then
> rounding down. Rounding down is subtracting the remainder if you were
> to divide by 10, for which we use the % operator in Python.
This will work if you use +9 in each case rather than +5 (note that he
doesn't really want rounding -- he wants 41 to 'round' to 50, for ex).
Alex
>>> n.quit()
'205 closing connection - goodbye!'
>>>
```

3.3.6. Пример клиентской программы NNTP

При создании программы клиента NNTP, приведенной в примере 3.2, попытаемся применить наиболее интересные возможности. Этот пример в целом будет аналогичным примеру с клиентом FTP, но в данном случае будет предусмотрена загрузка новейших поступлений, в частности, самой свежей статьи, опубликованной в группе новостей, которые касаются языка Python (`comp.lang.python`).

После получения этой статьи будет выведено вплоть до 20 первых строк в статье, а дополнительно к этому вплоть до 20 первых значимых строк статьи. Под значимыми строками подразумеваются строки с реальными данными, а не цитированный текст (который начинается со знака `>` или `!`) или даже цитированные текстовые вступления, как в примере: "In article <"..., soAndSo@some.domain wrote:.

Наконец, перейдем к осмысленной обработке пустых строк. Будет отображена одна пустая строка, если она присутствует в статье; если же имеется больше одной следующих друг за другом пустых строк, то будет показана только первая пустая строка из этого набора. В числе первых 20 строк учитываются только строки с реальными данными, поэтому может быть отображено максимальное количество строк вывода, равное 39, которое состоит из 20 строк с реальными данными, чередующихся с 19 пустыми строками.

Пример 3.2. Загрузка по протоколу NNTP (getFirstNNTP.py)

В этом сценарии осуществляются загрузка и отображение первых значимых строк (в количестве до 20) из последней по времени имеющийся статьи в группе новостей comp.lang.python, посвященной языку Python.

```

1  #!/usr/bin/env python
2
3  import nntplib
4  import socket
5
6  HOST = 'your.nntp.server'
7  GRNM = 'comp.lang.python'
8  USER = 'wesley'
9  PASS = 'youllNeverGuess'
10
11 def main():
12
13     try:
14         n = nntplib.NNTP(HOST)
15         #, user=USER, password=PASS)
16     except socket.gaierror as e:
17         print 'ERROR: cannot reach host "%s"' % HOST
18         print '      ("%s")' % eval(str(e))[1]
19         return
20     except nntplib.NNTPPermanentError as e:
21         print 'ERROR: access denied on "%s"' % HOST
22         print '      ("%s")' % str(e)
23         return
24     print '*** Connected to host "%s"' % HOST
25
26     try:
27         rsp, ct, fst, lst, grp = n.group(GRNM)
28     except nntplib.NNTPTemporaryError as ee:
29         print 'ERROR: cannot load group "%s"' % GRNM
30         print '      ("%s")' % str(e)
31         print '      Server may require authentication'
32         print '      Uncomment/edit login line above'
33         n.quit()
34         return
35     except nntplib.NNTPTemporaryError as ee:
36         print 'ERROR: group "%s" unavailable' % GRNM
37         print '      ("%s")' % str(e)
38         n.quit()
39         return
40     print '*** Found newsgroup "%s"' % GRNM
41
42     rng = '%s-%s' % (lst, lst)
43     rsp, frm = n.xhdr('from', rng)
44     rsp, sub = n.xhdr('subject', rng)
45     rsp, dat = n.xhdr('date', rng)
46     print '***** Found last article (#%s):
47
48     From: %s
49     Subject: %s
50     Date: %s
51     ''' % (lst, frm[0][1], sub[0][1], dat[0][1])

```



```

52
53     rsp, anum, mid, data = n.body(lst)
54     displayFirst20(data)
55     n.quit()
56
57 def displayFirst20(data):
58     print '*** First (<= 20) meaningful lines:\n'
59     count = 0
60     lines = (line.rstrip() for line in data)
61     lastBlank = True
62     for line in lines:
63         if line:
64             lower = line.lower()
65             if (lower.startswith('>') and not \
66                 lower.startswith('>>>')) or \
67                 lower.startswith('|') or \
68                 lower.startswith('in article') or \
69                 lower.endswith('writes:') or \
70                 lower.endswith('wrote:'):
71                 continue
72             if not lastBlank or (lastBlank and line):
73                 print '    %s' % line
74                 if line:
75                     count += 1
76                     lastBlank = False
77                 else:
78                     lastBlank = True
79             if count == 20:
80                 break
81
82 if __name__ == '__main__':
83     main()

```

Если при выполнении не произойдет никаких ошибок, то после вызова сценария на выполнение будут получены примерно такие результаты:

```

$ getLatestNNTP.py
*** Connected to host "your.nntp.server"
*** Found newsgroup "comp.lang.python"
*** Found last article (#471526):
    From: "Gerard Flanagan" <grflanagan@...>
    Subject: Re: Generate a sequence of random numbers that sum up to 1?
    Date: Sat Apr 22 10:48:20 CEST 2006
*** First (<= 20) meaningful lines:
def partition(N=5):
    vals = sorted( random.random() for _ in range(2*N) )
    vals = [0] + vals + [1]
    for j in range(2*N+1):
        yield vals[j:j+2]
deltas = [ x[1]-x[0] for x in partition() ]
print deltas
print sum(deltas)
[0.10271966686994982, 0.13826576491042208, 0.064146913555132801,
0.11906452454467387, 0.10501198456091299, 0.011732423830768779,
0.11785369256442912, 0.065927165520102249, 0.098351305878176198,
0.077786747076205365, 0.099139810689226726]
1.0
$

```

В этом выводе отображено действительное поступление в группу новостей, которое выглядит следующим образом:

```

From: "Gerard Flanagan" <grflanagan@...>
Subject: Re: Generate a sequence of random numbers that sum up to 1?
Date: Sat Apr 22 10:48:20 CEST 2006
Groups: comp.lang.python
Gerard Flanagan wrote:
> Anthony Liu wrote:
> > I am at my wit's end.
> > I want to generate a certain number of random numbers.
> > This is easy, I can repeatedly do uniform(0, 1) for
> > example.
> > But, I want the random numbers just generated sum up
> > to 1 .
> > I am not sure how to do this. Any idea? Thanks.
> -----
> import random
> def partition(start=0, stop=1, eps=5):
>     d = stop - start
>     vals = [ start + d * random.random() for _ in range(2*eps) ]
>     vals = [start] + vals + [stop]
>     vals.sort()
>     return vals
> P = partition()
> intervals = [ P[i:i+2] for i in range(len(P)-1) ]
> deltas = [ x[1] - x[0] for x in intervals ]
> print deltas

> print sum(deltas)
> -----
def partition(N=5):
    vals = sorted( random.random() for _ in range(2*N) )
    vals = [0] + vals + [1]
    for j in range(2*N+1):
        yield vals[j:j+2]
deltas = [ x[1]-x[0] for x in partition() ]
print deltas
print sum(deltas)
[0.10271966686994982, 0.13826576491042208, 0.064146913555132801,
0.11906452454467387, 0.10501198456091299, 0.011732423830768779,
0.11785369256442912, 0.065927165520102249, 0.098351305878176198,
0.077786747076205365, 0.099139810689226726]
1.0

```

Разумеется, при каждом последующем вызове сценария будет получен другой результат, поскольку публикация статей никогда не прекращается. Ни в коем случае не может случиться так, что два подряд вызова на выполнение приведут к получению одинакового вывода. Исключением может быть лишь то, что содержимое сервера телеконференций не было обновлено в связи с поступлением еще одной статьи после предыдущего выполнения сценария.

Построчное объяснение

Строки 1–9

Работа приложения начинается с выполнения нескольких инструкций `import` и определения некоторых констант, во многом аналогично тому, как в примере с клиентом FTP.

Строки 11–40

В первой части сценария предпринимается попытка подключиться к хост-серверу NNTP, и если эта попытка оказывается неудачной, происходит аварийное завершение (строки 13–24). Строка 15 закомментирована преднамеренно, поскольку она предусмотрена на тот случай, что сервер требует аутентификации (с указанием имени входа и пароля). Если это имеет место, раскомментируйте строку и внесите изменения в строке 14. Далее следует попытка загрузить конкретную группу новостей. Если требуемая группа новостей не существует, не заархивирована на данном сервере, а также если требуется аутентификация (строки 26–40), происходит аварийное завершение.

Строки 42–55

В следующей части сценария происходит поиск некоторых заголовков для их отображения (строки 42–51). Из всех заголовков наиболее значимыми являются те, что содержат данные об авторе, теме и дате. Происходит поиск этих данных и отображение для пользователя. Перед каждым вызовом метода `xhdr()` необходимо определить диапазон статей для поиска заголовков. Нас интересует только одно сообщение, поэтому диапазон задан в виде “X-X”, где X — номер последнего сообщения.

Метод `xhdr()` возвращает двухэлементный кортеж, состоящий из ответа сервера (`rsp`) и списка заголовков в указанном диапазоне. В данном случае мы запрашиваем информацию, касающуюся только одного сообщения (последнего), поэтому извлекаем лишь первый элемент из списка (`hdr[0]`). Этот элемент данных представляет собой двухэлементный кортеж, состоящий из номера статьи и строки данных. Номер статьи уже известен (и указан в запросе диапазона), поэтому нам требуется только второй элемент, строка данных (`hdr[0][1]`).

Последняя задача состоит в том, чтобы загрузить сам текст статьи (строки 53–55). Для решения этой задачи необходимо вызвать метод `body()`, вывести первые 20 или меньшее количество значимых строк (как определено в начале этого раздела), выйти из системы сервера и завершить работу сценария.

Строки 57–80

2.4

Основная часть обработки выполняется в функции `displayFirst20()` (строки 57–80). Эта функция получает набор строк, из которых состоит текст статьи, выполняет некоторую предварительную обработку, такую как установка нулевого значения счетчика, создание выражения-генератора, которое обеспечивает постепенную обработку в цикле набора строк текста (который может иметь значительный объем), составляющих текст, при условии, что лишние пустые строки должны быть удалены и не показаны (об этом будет подробнее сказано ниже; строки 59–61). Выражения-генераторы были впервые введены в версии Python 2.4, поэтому читатели, которые все еще используют версии 2.0–2.3, должны вместо этого задать расширенное выражение

списка. (В действительности читателям вообще не следует использовать какие-либо версии, предшествующие версии 2.4.) При очистке строк данных удаляются только заключительные пробельные символы (`rstrip()`), поскольку ведущие пробелы могут принадлежать к интересующим нас строкам кода на языке Python.

Как уже было сказано, мы должны учитывать требование не выводить какой-либо цитируемый текст или специально отмеченные текстовые вступления. Именно для этой цели предназначена охватывающая несколько операторов инструкция `if`, которая занимает строки 65–71 (включая также строку 64). Эта проверка выполняется, если текущая строка не является пустой (строка 63). Предварительно осуществляется приведение строки к нижнему регистру, чтобы в операциях сравнения не приходилось учитывать регистр (строка 64).

Если строка начинается со знака `>` или `|`, обычно это указывает на строку цитирования. Исключением являются строки, которые начинаются со знаков `>>>`, поскольку они могут указывать на приглашение интерактивного интерпретатора. Однако при этом возникает неоднозначность, связанная с тем, что так же обозначаются сообщения, которые цитировались трижды (повторялись три раза перед поступлением к четвертому респонденту). (Один из примеров в конце данной главы посвящен устранению этой неоднозначности в программе.) Строки, которые начинаются со слов `"in article..."`, и заканчиваются словом `"writes:"` или `"wrote:"`, за которыми следуют двоеточие (`:`), также представляют собой цитируемые текстовые вступления. Пропуск всех этих строк осуществляется с помощью инструкции `continue`.

Теперь перейдем к рассмотрению пустых строк. Постараемся предусмотреть как можно более осмысленную обработку этих строк в нашем приложении. Безусловно, все одинарные пустые строки из статьи должны сохраниться, но если подряд следует несколько пустых строк, то лишние строки должны быть удалены. Если одна за другой идут несколько пустых строк, то должна быть показана только первая из них, чтобы пользователю не приходилось самому пропускать ненужные пустые строки, выполняя прокрутку строк на экране, чтобы перейти к полезной информации. Кроме того, мы не должны засчитывать пустые строки, собирая обусловленный набор из 20 значимых строк. Все эти требования выполняются в строках 72–78.

Инструкция `if` в строке 72 указывает, что строка должна быть отображена, только если предыдущая строка не была пустой или если текущая строка содержит данные, пусть даже предшествующая ей строка была пустой. Иными словами, если при обработке принято решение вывести текущую строку, то эта строка содержит данные, а если является пустой, то предыдущая строка не была пустой. Теперь перейдем еще к одной сложной части: если рассматривается непустая строка, то нужно увеличить значение счетчика и установить флаг `lastBlank` равным `False`, поскольку эта строка не была пустой (строки 74–76). В противном случае, поскольку рассматриваемая строка оказалась пустой, флаг должен быть задан равным `True`.

Теперь вернемся к строке 61. Значение флага `lastBlank` было задано равным `True`, поэтому, если первая реальная (не относящаяся к вступлению или не цитируемая) строка текста является пустой, то ее не следует отображать; вначале должна быть показана первая строка с реальными данными!

Наконец, если первые 20 непустых строк уже выведены, можно завершить работу и отбросить оставшиеся строки (строки 79–80). В противном случае обработка всех строк закончена и можно завершить выполнение цикла `for` обычным образом.

3.3.7. Дополнительные сведения о протоколе NNTP

Дополнительные сведения о протоколе NNTP можно получить в определении/спецификации протокола NNTP (документ RFC 977) по адресу <http://tools.ietf.org/html/rfc977>, а также на веб-странице <http://www.networksorcery.com/enp/protocol/nntp.htm>. К другим связанным с этим протоколом документам RFC относятся документы 1036 и 2980. Чтобы больше узнать о поддержке протокола NNTP в языке Python, можно перейти по адресу <http://docs.python.org/library/nntplib>.

3.4. Электронная почта

Электронная почта одновременно является и архаичной, и современной. Особенно “старой” электронная почта кажется тем, кто использовал Интернет с первых дней работы этой сети, тем более что они могут ее сравнить с более новыми и непосредственными механизмами связи, такими как интерактивная переписка на основе веб-интерфейса, средства мгновенного обмена сообщениями (instant messaging — IM) и цифровая телефония в виде таких приложений, как передача голоса по протоколу Интернета (Voice over Internet Protocol — VoIP). В следующем разделе приведено общее описание того, как работает электронная почта. Читатели, знакомые с этой тематикой и желающие непосредственно приступить к разработке клиентов для электронной почты на языке Python, могут перейти к следующим разделам.

Прежде чем приступать к рассмотрению электронной почты, необходимо понять, в чем состоит точное определение сообщений электронной почты. Итак, согласно документу RFC 2822, “[a] сообщение состоит из полей заголовка (именуемых в целом заголовком сообщения), за которыми может следовать текст сообщения”. Нас как пользователей электронной почты непосредственно касается ее содержимое, будь то реальное сообщение или так называемое незапрашиваемое коммерческое объявление (или спам). Тем не менее еще раз отметим: как следует из документа RFC, текст может быть не приведен, а обязательными являются только заголовки. На первых порах трудно понять, для чего нужно электронное письмо без текста!

3.4.1. Компоненты и протоколы почтовой системы

Многие этого не знают, но в действительности электронная почта существовала еще до появления Интернета в его современном виде. Развитие электронной почты фактически началось с организации простого обмена сообщениями между пользователями мэйнфреймов; для этого не требовалась даже сеть как таковая, поскольку все отправители и получатели работали на терминалах, хотя и за одним большим компьютером. Затем получили свое развитие сети, позволяющие связать между собой несколько компьютеров, поэтому возникла необходимость обеспечить обмен сообщениями между пользователями, работающими на разных сетевых узлах. Очевидно, что при этом задача усложнилась, поскольку связь нужно было организовать между людьми, использующими разные компьютеры, на которых, в свою очередь, могли применяться разные сетевые протоколы. Поэтому стандарт передачи электронной почты по Интернету, который стал основой единой системы обмена сообщениями, сложился только в начале 1980-х годов.

Прежде чем углубляться в подробности, необходимо рассмотреть, как в целом работает электронная почта. Первым делом нужно понять, как сообщение, переданное отправителем, достигает получателя, пройдя через многие из технических устройств,

образующих Интернет. Упрощая задачу, можно свести рассмотрение этой темы к исходному компьютеру отправителя (из которого передается сообщение отправителя) и целевому компьютеру получателя (почтовому серверу получателя). Оптимальное решение может быть найдено, если на компьютере отправителя имеются точные сведения о том, как достичь узла получателя, поскольку в этом случае обеспечивается создание непосредственного соединения для доставки сообщения. Однако обычно обстоятельства не складываются таким образом.

Компьютер отправителя рассылает запросы, чтобы найти какой-то промежуточный узел, способный обеспечить прохождение сообщения в нужном направлении на пути к узлу конечного получателя. После этого такой промежуточный узел ищет следующий узел, который находится еще на один шаг ближе к месту назначения. Таким образом, сообщение, передаваемое от начального узла до конечного узла назначения, проходит через целый ряд компьютеров. Этапы прохождения сообщения принято называть участками маршрута. В любом полностью развернутом сообщении электронной почты, полученном пользователем, в составе заголовков можно видеть своего рода “заграничный паспорт” с отметками всех “границ”, или промежуточных узлов, которые пройдены сообщением на пути к месту назначения.

Теперь рассмотрим задачу обмена сообщениями электронной почтой немного с другой стороны, с точки зрения компонентов почтовой системы. На переднем плане находится компонент, именуемый *транспортный агент электронной почты* (mail transport agent — MTA). Это серверный процесс, работающий на узле почтового обмена, который отвечает за маршрутизацию, постановку в очередь и отправку электронной почты. Агенты MTA работают на всех узлах, через которые передается сообщение электронной почты, начиная от исходного узла и заканчивая конечным целевым узлом, не говоря уже о том, что эти агенты обеспечивают прохождение почты через все участки маршрута, связывающие эти узлы. Таким образом, агенты MTA выполняют роль агентов системы сетевого транспорта сообщений.

При выполнении своей работы агенты MTA должны иметь возможность определить следующее: во-первых, местонахождение следующего агента MTA, которому необходимо перенаправить сообщение, и, во-вторых, способ взаимодействия с этим агентом передачи почты. Для решения первой задачи используется *служба доменных имен* (domain name service — DNS), позволяющая найти по специальным таблицам запись с обозначением MX (Mail eXchange — почтовый обмен), которая относится к целевому домену. Эта запись не обязательно должна указывать на конечного получателя; она может просто предоставлять сведения о следующем получателе, который может способствовать доставке в конечном итоге данного сообщения в его заключительное место назначения. Теперь мы должны рассмотреть, как агенты MTA перенаправляют сообщения другим агентам MTA.

3.4.2. Отправка электронной почты

Для отправки электронной почты почтовый клиент пользователя должен установить соединение с агентом MTA и обмениваться с ним сведениями, касающимися передачи сообщения, на языке, понятном им обоим, для чего служит протокол связи. Средой, в которой агенты MTA обмениваются сообщениями друг с другом, является *система передачи сообщений* (message transport system — MTS). Последовательность действий при обмене сообщениями, называемая протоколом, должна быть согласованной заранее и соблюдаться обоими агентами MTA, поскольку в противном случае ни о каком взаимодействии не может идти речь. Как было указано в начале этого

раздела, в первые дни становления компьютерных сетей подобная связь была ненадежной и непредсказуемой, поскольку разные изготовители выпускали компьютерные системы многих различных типов, на которых применялось сетевое программное обеспечение, разработанное исключительно для той или иной системы. Кроме того, для связи между компьютерами использовались не только сетевые средства передачи данных, но и коммутируемые модемы, поэтому невозможно было прогнозировать сроки доставки сообщений. Действительно, в свое время автор столкнулся с таким фактом, что некоторое сообщение куда-то исчезло, но затем появилось у получателя почти через девять месяцев после его отправки! Разве это возможно при тех скоростях, на которых работает современный Интернет? На пути к преодолению всех этих недостатков постепенно сложился *простой протокол электронной почты* (Simple Mail Transfer Protocol — SMTP), который стал основой современной электронной почты.

Протоколы SMTP, ESMTP, LMTP

Протокол SMTP был первоначально разработан ныне покойным Джонатаном Постелом (Jonathan Postel) из Института научной информации (Institute for Scientific Information — ISI) и опубликован в документе RFC 821 в августе 1982 г. После этого было выпущено несколько версий этого протокола. В дальнейшем был создан ряд так называемых служебных расширений протокола SMTP и на их основе разработан протокол ESMTP, опубликованный в документе RFC 1869 в ноябре 1995 г. После этого оба протокола, SMTP и ESMTP, были описаны в действующем на данный момент документе RFC 5321, опубликованном в октябре 2008 г. Далее в этой книге для ссылки и на SMTP, и на ESMTP будет использоваться только термин SMTP. Для создания приложений общего назначения обычно достаточно лишь иметь возможность зарегистрироваться на сервере, отправить сообщение и выйти из системы. Все остальные функции, поддерживаемые протоколом SMTP, являются дополнительными.

Следует также упомянуть еще один вариант протокола отправки почты, известный как LMTP (Local Mail Transfer Protocol). Он был создан на основе протоколов SMTP и ESMTP и определен в документе RFC 2033, опубликованном в октябре 1996 г. Одним из обязательных требований к реализации протокола SMTP является организация почтовых очередей, но для этого требуются дополнительные системы хранения и администрирования, поэтому был разработан протокол LMTP, в котором предусмотрена более простая система передачи почты, исключающая необходимость применения почтовых очередей. С другой стороны, этот протокол требует, чтобы доставка сообщений осуществлялась немедленно (поскольку лишь при этом условии можно обойтись без очередей). Доступ извне к серверам LMTP не предоставляется. Эти серверы работают непосредственно с шлюзом электронной почты, сопряженным с Интернетом, который позволяет определить, является ли то или иное сообщение принятым или отвергнутым. Шлюз в определенной степени заменяет в протоколе LMTP очередь.

Агенты MTA

Некоторые известные агенты MTA, в которых реализован протокол SMTP, приведены ниже.

Агенты MTA с открытым исходным кодом

- Sendmail
- Postfix

- Exim
- qmail

Коммерческие агенты МТА

- Microsoft Exchange
- Почтовый сервер Lotus Notes Domino

Следует учитывать, что во всех этих приложениях реализованы минимальные требования протокола SMTP, но в большинстве из них, особенно в коммерческих агентах МТА, добавлена поддержка серверами функций, дополнительных по отношению к стандартному определению протокола.

Сервер SMTP — это приложение MTS, которое используется большинством агентов МТА в Интернете для обмена сообщениями. Сам протокол SMTP используется агентами передачи почты для отправки электронной почты от одного хоста (МТА) к другому хосту (МТА). При отправке электронной почты необходимо подключиться к серверу SMTP для исходящей почты, который выполняет для почтового приложения роль клиента SMTP. Таким образом, сервер SMTP находится в конце первого участка маршрута для сообщения.

3.4.3. Язык Python и протокол SMTP

Как и в отношении поддержки многих других протоколов, предусмотрены библиотека `smtpplib` и класс `smtpplib.SMTP`, экземпляр которого должен быть создан в программе. Рассмотрим, как в данном случае реализуются уже известные нам принципы работы с протоколами в языке Python.

1. Подключение к серверу.
2. Вход в систему (если это предусмотрено).
3. Выполнение запросов на обслуживание.
4. Завершение работы.

Как и в случае с протоколом NNTP, этап регистрации является необязательным и может потребоваться, только если на сервере предусмотрена аутентификация по протоколу SMTP (SMTP-AUTH). Режим SMTP-AUTH определен в документе RFC 2554. Аналогичным протоколу NNTP является также то, что для работы по протоколу SMTP достаточно обеспечить взаимодействие только с одним портом на сервере; на этот раз речь идет о порте 25.

Ниже приведен фрагмент псевдокода Python, с которого можно начать знакомство с этим протоколом.

```
from smtpplib import SMTP
n = SMTP('smtp.yourdomain.com')
...
n.quit()
```

Прежде чем перейти к рассмотрению реального примера, необходимо ознакомиться с некоторыми из наиболее широко применяемых методов класса `smtpplib.SMTP`.

3.4.4. Методы класса `smtplib.SMTP`

2.6

В дополнение к классу `smtplib.SMTP` в версии Python 2.6 были введены еще два класса: `SMTP_SSL` и `LMTP`. В последнем классе реализован протокол LMTP, как было описано выше в разделе 3.4.2, а первый из указанных действует точно так же, как и обычный класс `SMTP`, за исключением того, что в нем обмен данных осуществляется по сокету, поддерживающему шифрование, и сам этот класс является альтернативным по отношению к реализациям протокола SMTP с использованием TLS. Если порт для протокола `SMTP_SSL` не указан, то по умолчанию применяется порт 465.

Как и в предыдущих разделах, в настоящем разделе будут показаны не все методы, принадлежащие классу `SMTP`, а только те, которые необходимы для создания клиентского приложения SMTP. Для большинства приложений, которые должны обеспечивать отправку электронной почты, требуются только следующие два метода: `sendmail()` и `quit()`.

Все параметры вызова метода `sendmail()` должны соответствовать документу RFC 2822; это означает, что адреса электронной почты должны быть отформатированы в соответствии с требованиями, а текст сообщения должен сопровождаться предшествующими ему заголовками и содержать строки, разделенные парами символов возврата каретки и перевода строки (`\r\n`).

Необходимо отметить, что сам текст сообщения не должен обязательно присутствовать. Согласно документу RFC 2822, "...обязательными полями заголовка являются только поле даты составления письма и поля адреса составителя сообщения, например "Дата:" и "От:" (MAIL FROM, RCPT TO, DATA)".

Некоторые наиболее широко применяемые методы объекта `SMTP` представлены в табл. 3.3. Имеется также довольно значительное количество других методов, которые здесь не описаны, но они обычно не требуются для отправки сообщения электронной почты. Дополнительные сведения обо всех методах объекта `SMTP` приведены в документации по языку Python.

Таблица 3.3. Широко распространенные методы объектов класса `SMTP`

Метод	Описание
<code>sendmail(from, to, msg [, mopts, ropts])</code>	Передаёт сообщение <i>msg</i> от отправителя <i>from</i> к получателю <i>to</i> (список/кортеж) с необязательными параметрами ESMTP, которые относятся к почте (<i>mopts</i>) и получателю (<i>ropts</i>)
<code>ehlo()</code> или <code>helo()</code>	Иницирует сеанс обмена данными с сервером SMTP или ESMTP с помощью команды EHLO или HELO соответственно. Этот метод не должен быть обязательным, поскольку <code>sendmail()</code> вызывает его по мере необходимости
<code>starttls(keyfile=None, certfile=None)</code>	Передаёт указание серверу о переходе в режим TLS (Transport Layer Security). Если задан либо параметр <i>keyfile</i> , либо <i>certfile</i> , то соответствующий параметр используется для создания защищенного сокета
<code>set_debuglevel(level)</code>	Задаёт уровень отладки для обмена данными с сервером
<code>quit()</code>	Закрывает соединение и завершает работу
<code>login(user, passwd)^a</code>	Позволяет зарегистрироваться на сервере SMTP с указанием имени пользователя <i>user</i> и пароля <i>passwd</i>

^a Только для режима SMTP-AUTH.

3.4.5. Пример сеанса интерактивного взаимодействия с использованием протокола SMTP

В этом разделе мы еще раз рассмотрим интерактивный пример:

```
>>> from smtplib import SMTP as smtp
>>> s = smtp('smtp.python.is.cool')
>>> s.set_debuglevel(1)
>>> s.sendmail('wesley@python.is.cool', ('wesley@python.is.cool', 'chun@python.
is.cool'), '' From: wesley@python.is.cool\r\nTo: wesley@python.is.cool, chun@python.
is.cool\r\nSubject: test msg\r\n\r\nxxx\r\n.'')
send: 'ehlo myMac.local\r\n'
reply: '250-python.is.cool\r\n'
reply: '250-7BIT\r\n'
reply: '250-8BITMIME\r\n'
reply: '250-AUTH CRAM-MD5 LOGIN PLAIN\r\n'
reply: '250-DSN\r\n'
reply: '250-EXPN\r\n'
reply: '250-HELP\r\n'
reply: '250-NOOP\r\n'
reply: '250-PIPELINING\r\n'
reply: '250-SIZE 15728640\r\n'
reply: '250-STARTTLS\r\n'
reply: '250-VERS V05.00c++\r\n'
reply: '250 XMVP 2\r\n'
reply: retcode (250); Msg: python.is.cool
7BIT
8BITMIME
AUTH CRAM-MD5 LOGIN PLAIN
DSN
EXPN
HELP
NOOP
PIPELINING
SIZE 15728640
STARTTLS
VERS V05.00c++
XMVP 2
send: 'mail FROM:<wesley@python.is.cool> size=108\r\n'
reply: '250 ok\r\n'
reply: retcode (250); Msg: ok
send: 'rcpt TO:<wesley@python.is.cool>\r\n'
reply: '250 ok\r\n'
reply: retcode (250); Msg: ok
send: 'data\r\n'
reply: '354 ok\r\n'
reply: retcode (354); Msg: ok
data: (354, 'ok')
send: 'From: wesley@python.is.cool\r\nTo: wesley@python.is.cool\r\nSubject: test msg\r\
n\r\nxxx\r\n.\r\n.\r\n'
reply: '250 ok ; id=2005122623583701300or7hhe\r\n'
reply: retcode (250); Msg: ok ; id=2005122623583701300or7hhe
data: (250, 'ok ; id=2005122623583701300or7hhe')
{}
>>> s.quit()
send: 'quit\r\n'
reply: '221 python.is.cool\r\n'
reply: retcode (221); Msg: python.is.cool
```

3.4.6. Дополнительные сведения о протоколе SMTP

С дополнительными сведениями о протоколе SMTP можно ознакомиться в определении/спецификации протокола SMTP в документе RFC 5321, который находится по адресу <http://tools.ietf.org/html/rfc2821>. Чтобы больше узнать о поддержке протокола SMTP в языке Python, перейдите по адресу <http://docs.python.org/library/smtplib>.

Одним из наиболее важных аспектов применения электронной почты, который мы еще не обсуждали, является форматирование должным образом адресов Интернета, а также самих сообщений электронной почты. Эта информация приведена в новейшей спецификации формата сообщений Интернета в документе RFC 5322, который доступен по адресу <http://tools.ietf.org/html/rfc5322>.

3.4.7. Получение электронной почты

В первое время обмен электронной почтой в Интернете был прерогативой студентов университетов, исследователей, а также сотрудников частных промышленных предприятий и коммерческих корпораций. В качестве настольных компьютеров главным образом использовались рабочие станции на основе Unix. Домашние пользователи в основном ограничивались применением коммутируемого доступа к Интернету со своих персональных компьютеров и почти не использовали электронную почту. После того как началось стремительное развитие Интернета в середине 1990-х годов, электронная почта вошла практически в каждый дом.

Для домашних пользователей не реально иметь у себя рабочие станции, на которых работает сервер SMTP, поэтому возникла необходимость в разработке системы нового типа, которая предусматривает сохранение электронной почты на узле входящей почты, обеспечивая периодическую загрузку почты для чтения в автономном режиме. Для создания такой системы потребовалось разработать не только новое приложение, но и новый протокол, предназначенный для обмена данными с почтовым сервером.

Приложение, работающее на домашнем компьютере, получило название почтового агента пользователя (mail user agent — MUA). Агент MUA загружает почту с сервера и в зависимости от настройки автоматически удаляет загруженную почту или оставляет ее на сервере для последующего удаления вручную пользователем. Кроме того, агент MUA должен обладать способностью отправлять электронные письма, иными словами, эта программа должна поддерживать протокол SMTP для непосредственного обмена с агентом MTA при отправке электронного письма. Клиентская программа аналогичного типа уже рассматривалась в предыдущем разделе при описании протокола SMTP. Рассмотрим, как при этом происходит загрузка электронной почты.

3.4.8. Протоколы POP и IMAP

Первым протоколом, разработанным для загрузки электронной почты, был протокол POP (Post Office Protocol). Как указано в оригинальном документе RFC 918, опубликованном в октябре 1984 года, — “протокол POP должен обеспечить для рабочей станции пользователя получение доступа к почте с сервера почтового ящика”. Предполагалось, что почта будет отправляться с рабочей станции на сервер почтового ящика с помощью протокола SMTP (Simple Mail Transfer Protocol). Новейшая версия протокола POP обозначена номером 3, поэтому для этого протокола часто

используется наименование POP3. Протокол POP3, определенный в документе RFC 1939, до сих пор продолжает широко использоваться.

Через несколько лет после создания протокола POP появился конкурирующий протокол, получивший название IMAP (Internet Message Access Protocol), или протокол интерактивного доступа к электронной почте. (Иногда приходится слышать, что аббревиатура IMAP должна расшифровываться следующим образом: Internet Mail Access Protocol, Interactive Mail Access Protocol или Interim Mail Access Protocol.) Первая версия IMAP была экспериментальной и только после появления версии 2 был опубликован документ RFC, относящийся к этому протоколу (документ RFC 1064, опубликованный в июле 1988 г.). В документе RFC 1064 указано, что стимулом к разработке протокола IMAP2 стало появление второй версии POP, т.е. POP2.

Назначение протокола IMAP состоит в том, чтобы предоставить более полное решение по сравнению с протоколом POP; однако этот протокол сложнее, чем POP. Преимущество IMAP состоит в том, что этот протокол является чрезвычайно удобным для работы в современных условиях, когда пользователи обращаются к своим сообщениям электронной почты с нескольких разных устройств, например, с настольных, переносных и планшетных компьютеров, мобильных телефонов, графических игровых систем и т.д. Протокол POP не слишком приспособлен для работы с несколькими почтовыми клиентами и в целом рассматривается как устаревший, хотя все еще широко используется. Следует учитывать, что многие провайдеры Интернета в настоящее время предоставляют только протокол POP для получения электронной почты (и SMTP для ее отправки). Можно предположить, что в дальнейшем протокол IMAP будет находить все более широкое распространение.

Текущей версией протокола IMAP, которая в основном применяется в наши дни, является IMAP4rev1. В действительности в Microsoft Exchange, одном из наиболее распространенных в мире почтовых серверов, в качестве основного механизма загрузки используется протокол IMAP. Ко времени написания этой книги последнее черновое определение протокола IMAP4rev1 приведено в документе RFC 3501, опубликованном в марте 2003 г. Мы используем термин IMAP4 для обозначения обеих версий протокола, IMAP4 и IMAP4rev1.

Рекомендуется в качестве дополнительной литературы просмотреть вышеупомянутые документы RFC. Диаграмма на рис. 3.3 показывает всю эту сложную систему, которую мы привыкли называть просто электронной почтой.

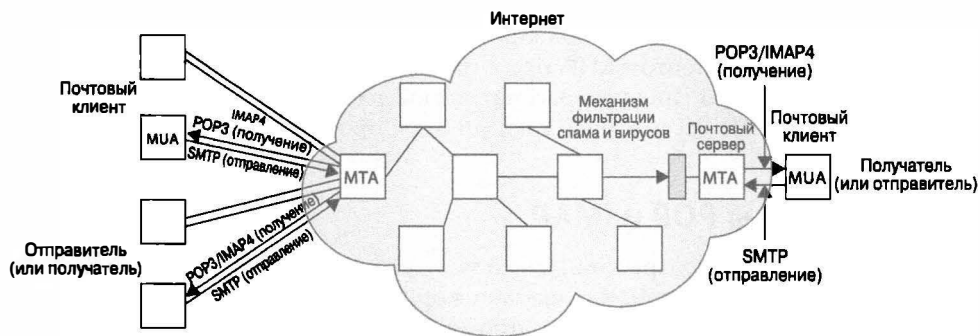


Рис. 3.3. Схема расположения отправителей и получателей электронной почты в Интернете. Клиенты загружают и отправляют почту через агентов MUA, которые взаимодействуют с соответствующими агентами MTA. Электронная почта проходит по транзитным переходам от одного MTA к другому, пока не достигнет конечного назначения

Перейдем к более подробному рассмотрению средств поддержки протоколов POP3 и IMAP4 в языке Python.

3.4.9. Язык Python и протокол POP3

Снова воспользуемся знакомой последовательностью действий: импортируем библиотеку `poplib` и создадим экземпляр класса `poplib.POP3`; стандартный диалог проходит, как и следовало ожидать.

1. Подключение к серверу.
2. Вход в систему.
3. Выполнение запросов на обслуживание.
4. Завершение работы.

Ниже приведен необходимый псевдокод Python.

```
from poplib import POP3
p = POP3('pop.python.is.cool')
p.user(...)
p.pass(...)
...
p.quit()
```

2.4

Прежде чем перейти к рассмотрению реального примера, необходимо подчеркнуть, что имеется также класс `poplib.POP3_SSL` (впервые введенный в версии 2.4), который обеспечивает передачу почты по зашифрованному соединению, если заданы соответствующие учетные данные. Рассмотрим интерактивный пример и ознакомимся с основными методами класса `poplib.POP3`.

3.4.10. Интерактивный пример работы по протоколу POP3

Ниже приведен интерактивный пример, в котором используется библиотека `poplib` языка Python. В ходе выполнения этого примера возникло исключение, которое, очевидно, вызвано умышленным вводом неверного пароля исключительно для демонстрации того, с чем можно столкнуться на практике при работе с сервером. Результаты работы в интерактивном режиме приведены ниже.

```
>>> from poplib import POP3
>>> p = POP3('pop.python.is.cool')
>>> p.user('wesley')
'+OK'
>>> p.pass_("you'llNeverGuess")
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
    File "/usr/local/lib/python2.4/poplib.py", line 202,
    in pass_
        return self._shortcmd('PASS %s' % pswd)
    File "/usr/local/lib/python2.4/poplib.py", line 165,
    in _shortcmd
        return self._getresp()
    File "/usr/local/lib/python2.4/poplib.py", line 141,
    in _getresp
```

```

    raise error_proto(resp)
poplib.error_proto: -ERR directory status: BAD PASSWORD
>>> p.user('wesley')
'+OK'
>>> p.pass_('youllNeverGuess')
'+OK ready'
>>> p.stat()
(102, 2023455)
>>> rsp, msg, siz = p.retr(102)
>>> rsp, siz
('+OK', 480)
>>> for eachLine in msg:
...     print eachLine
...
Date: Mon, 26 Dec 2005 23:58:38 +0000 (GMT)
Received: from c-42-32-25-43.smtp.python.is.cool
        by python.is.cool (scnrch31) with ESMTP
        id <2005122623583701300or7hhe>; Mon, 26 Dec 2005 23:58:37 +0000
From: wesley@python.is.cool
To: wesley@python.is.cool
Subject: test msg

xxx
.
>>> p.quit()
'+OK python.is.cool'
```

3.4.11. Методы класса poplib.POP3

Класс POP3 предоставляет многочисленные методы, с помощью которых можно легко загружать почту и управлять ящиком для входящей почты в автономном режиме. Наиболее широко используемые методы перечислены в табл. 3.4.

Таблица 3.4. Наиболее широко применяемые методы объектов класса POP3

Метод	Описание
<code>user(login)</code>	Отправляет серверу имя входа <code>login</code> ; ожидает ответ, указывающий, что сервер запрашивает пароль пользователя
<code>pass_(passwd)</code>	Передаёт пароль <code>passwd</code> (после того как пользователь регистрируется с помощью метода <code>user()</code>); если сочетание "имя входа/пароль" отвергается сервером, возникает исключение
<code>stat()</code>	Возвращает информацию о состоянии почтового ящика, двухэлементный кортеж (<code>msg_ct</code> , <code>mbox_siz</code>): общее количество сообщений и общий размер сообщения в октетах (байтах)
<code>list([msgnum])</code>	Метод, превосходящий по своим возможностям метод <code>stat()</code> . Возвращает весь список сообщений с сервера в виде трехэлементного кортежа (<code>rsp</code> , <code>msg_list</code> , <code>rsp_siz</code>): ответ сервера, список сообщений, размер сообщения ответа. Если задан параметр <code>msgnum</code> , возвращает данные только для сообщения, указанного этим параметром
<code>retr(msgnum)</code>	Осуществляет выборку сообщения <code>msgnum</code> с сервера и устанавливает для него флаг <code>seen</code> ; возвращает трехэлементный кортеж (<code>rsp</code> , <code>msglines</code> , <code>msgsiz</code>): ответ сервера, все строки сообщения <code>msgnum</code> и размер сообщения в байтах/октетах

Окончание табл. 3.4

Метод	Описание
<code>delete(msgnum)</code>	Обозначает сообщение с номером <i>msgnum</i> как намеченное для удаления. На большинстве серверов запросы на удаление обрабатываются после выполнения метода <code>quit()</code>
<code>quit()</code>	Обеспечивает выход из системы и фиксацию изменений (например, обработку флагов <code>seen</code> , <code>delete</code> и т.д.), разблокирует почтовый ящик, закрывает соединение, а затем завершает работу

При входе в систему метод `user()` не только отправляет имя входа на сервер, но и ожидает ответ, указывающий, что серверу должен быть предоставлен пароль пользователя. Если выполнение метода `pass_()` оканчивается неудачей из-за проблем аутентификации, активизируется исключение `poplib.error_proto`. В случае успеха этот метод возвращает положительный ответ, например, “+OK ready”, и почтовый ящик на сервере блокируется до тех пор, пока не произойдет вызов метода `quit()`.

Что касается метода `list()`, то список сообщений `msg_list` имеет форму `[msgnum msgsiz,...]`, где *msgnum* и *msgsiz* — соответственно номер и размер для каждого сообщения.

Предусмотрено также несколько других методов, которые не приведены в данной таблице. Для ознакомления со всеми доступными сведениями обратитесь к документации по `poplib` в справочном руководстве по библиотеке Python.

3.4.12. Пример применения протоколов SMTP и POP3

В примере 3.3 показано, как использовать протоколы SMTP и POP3 для создания клиента, который позволяет не только получать и загружать электронную почту, но и отправлять ее и выгружать на сервер. Перед этой программой поставлена задача отправить сообщение электронной почты самому отправителю (или какой-то учетной записи, предназначенной для проверки) с помощью протокола SMTP, перейти в состояние ожидания на некоторое время (в данном случае произвольно выбрано время десять секунд), а затем воспользоваться протоколом POP3 для загрузки сообщения и подтверждения идентичности отправленного и принятого писем. Как признак успеха будет рассматриваться завершение программы без сообщений; иными словами, после выхода из программы не должен формироваться какой-либо вывод и не должны появляться сообщения об ошибках.

Пример 3.3. Протоколы SMTP и POP3 (`myMail.py`)

В этом сценарии происходит отправка проверочного сообщения электронной почты по адресу получателя (через исходящий почтовый сервер/почтовый сервер SMTP), а после этого получение письма (с сервера входящей почты/POP). Для подготовки этого сценария к использованию в других условиях достаточно откорректировать имена серверов и адреса электронной почты.

```

1  #!/usr/bin/env python
2
3  from smtplib import SMTP
4  from poplib import POP3
5  from time import sleep
6
7  SMTPSVR = 'smtp.python.is.cool'
```

```
8 POP3SVR = 'pop.python.is.cool'
9
10 who = 'wesley@python.is.cool'
11 body = ''\
12 From: %(who)s
13 To: %(who)s
14 Subject: test msg
15
16 Hello World!
17 ''' % {'who': who}
18
19 sendSvr = SMTP(SMTPSVR)
20 errs = sendSvr.sendmail(who, [who], origMsg)
21 sendSvr.quit()
22 assert len(errs) == 0, errs
23 sleep(10) # ожидать доставки почты
24
25 recvSvr = POP3(POP3SVR)
26 recvSvr.user('wesley')
27 recvSvr.pass_('youllNeverGuess')
28 rsp, msg, siz = recvSvr.retr(recvSvr.stat()[0])
29 # удалить заголовки и сравнить с исходным сообщением
30 sep = msg.index('\n')
31 recvBody = msg[sep+1:]
32 assert origBody == recvBody # подтвердить идентичность
```

Построчное объяснение

Строки 1–8

Код приложения начинается с нескольких инструкций **import** и объявлений некоторых констант, во многом аналогично тому, как и в других примерах в этом разделе. В данном случае константы содержат имена почтовых серверов исходящей (SMTP) и входящей (POP3) почты.

Строки 10–17

В этих строках находятся операторы, обеспечивающие подготовку содержимого сообщения. Для этого тестового сообщения в качестве отправителя и получателя выступает один и тот же пользователь. Необходимо соблюдать требование документа RFC 2822, касающиеся применения разделителей строк определенного формата и пустой строки, отделяющей эти два раздела.

Строки 19–23

Вначале производится подключение к исходящему серверу (SMTP) и отправка сообщения. В данном случае задана другая пара адресов “От:” и “Кому:”. Это — действительные адреса электронной почты, или адреса отправителя и получателей конверта. Поле получателя должно допускать возможность указания не только одного, но и нескольких адресов. Если передана одна строка, то она преобразуется в список, состоящий из одного элемента. Признаком нежелательной электронной почты, или спама, обычно является несоответствие между заголовками сообщения и конверта.

Третьим параметром метода `sendmail()` является само сообщение электронной почты. После того как происходит возврат этого сообщения, осуществляется выход из системы сервера SMTP и проверяется, что не возникли какие-либо ошибки. Затем в распоряжении серверов предоставляется определенное время на отправку и получение этого сообщения.

Строки 25–32

В заключительной части приложения происходит загрузка только что отправленного сообщения и подтверждение того, что отправленное и полученное сообщения идентичны. Соединение с сервером POP3 устанавливается с указанием имени пользователя и пароля. После успешного входа в систему выполняется вызов метода `stat()` для получения списка имеющихся сообщений. Выбирается первое сообщение (`[0]`) и происходит вызов метода `retr()` для его загрузки.

Выполняется поиск пустой строки, отделяющей заголовки и сообщение, заголовки отбрасываются, а затем текст первоначального сообщения сравнивается с текстом полученного сообщения. Если сообщения идентичны, какой-либо вывод не формируется и происходит успешный выход из программы. В противном случае формируется некоторое утверждение.

Следует учитывать, что в ходе работы этой программы могут возникать весьма разнообразные ошибки, но в данный пример не был включен весь код контроля ошибок, чтобы сценарий оставался удобным для восприятия. (В одном из упражнений в конце данной главы предлагается добавить код контроля ошибок.)

После изучения данной главы читатель должен иметь достаточно полное представление о том, как происходят передача и прием электронной почты в современной вычислительной среде. Чтобы продолжить изучение этой сферы применения экспертных знаний в области программирования, перейдите к следующему разделу, представляющему другие относящиеся к электронной почте модули Python, которые могут оказаться полезными при разработке приложений.

3.4.13. Язык Python и протокол IMAP4

В языке Python для поддержки протокола IMAP4 предусмотрен модуль `imaplib`. По своему назначению он весьма напоминает модули поддержки других протоколов Интернета, описанные в этой главе. Прежде всего необходимо импортировать библиотеку `imaplib` и создать экземпляр одного из классов `imaplib.IMAP4*`; для этого, как и следует ожидать, применяется уже знакомая нам последовательность операторов.

1. Подключение к серверу.
2. Вход в систему.
3. Выполнение запросов на обслуживание.
4. Завершение работы.

Псевдокод Python также аналогичен тому, что был приведен ранее в этой главе:

```
from imaplib import IMAP4
s= IMAP4('imap.python.is.cool')
s.login(...)
...
s.close()
s.logout()
```

2.3

Этот модуль определяет три класса, IMAP4, IMAP4_SSL и IMAP4_stream, которые могут использоваться для подключения к серверу, совместимому с IMAP4. Как и класс POP3_SSL для POP, класс IMAP4_SSL позволяет подключаться к серверам IMAP4 с использованием сокета, обеспечивающего шифрование по протоколу SSL. Еще одним классом, обеспечивающим поддержку протокола IMAP, является класс IMAP4_stream, который позволяет создать интерфейс доступа к серверу IMAP4 с помощью объекта, подобного файлу. Последние два из указанных классов были добавлены в версии Python 2.3.

Теперь рассмотрим интерактивный пример, позволяющий ознакомиться с основными методами класса `imaplib.IMAP4`.

3.4.14. Интерактивный пример применения протокола IMAP4

Ниже приведен интерактивный пример, в котором используется модуль `imaplib` языка Python.

```
>>> s = IMAP4('imap.python.is.cool') # порт, заданный по умолчанию: 143
>>> s.login('wesley', 'youllneverguess')
('OK', ['LOGIN completed'])
>>> rsp, msgs = s.select('INBOX', True)
>>> rsp
'OK'

>>> msgs
['98']
>>> rsp, data = s.fetch(msgs[0], '(RFC822)')
>>> rsp
'OK'
>>> for line in data[0][1].splitlines()[5]:
...     print line
...
Received: from mail.google.com
    by mx.python.is.cool (Internet Inbound) with ESMTP id 316ED380000ED
    for <wesley@python.is.cool>; Fri, 11 Mar 2011 10:49:06 -0500 (EST)
Received: by gyb11 with SMTP id 11sol25539gyb.10
    for <wesley@python.is.cool>; Fri, 11 Mar 2011 07:49:03 -0800 (PST)
>>> s.close()
('OK', ['CLOSE completed'])
>>> s.logout()
('BYE', ['IMAP4rev1 Server logging out'])
```

3.4.15. Общие методы класса `imaplib.IMAP4`

Как уже было сказано выше, протокол IMAP сложнее, чем POP, поэтому для работы с ним предусмотрено гораздо больше методов. В данной главе, как обычно, представлены далеко не все методы. В частности, в табл. 3.5 приведено описание только основных методов, которые с наибольшей вероятностью могут потребоваться для создания простого приложения электронной почты.

Таблица 3.5. Наиболее широко применяемые методы объектов класса IMAP4

Метод	Описание
<code>close()</code>	Закрывает текущий почтовый ящик. Если доступ не задан как допускающий только чтение, то все сообщения, отмеченные как предназначенные для удаления, будут уничтожены
<code>fetch(message_set, message_parts)</code>	Обеспечивает получение сообщений электронной почты (или отдельных частей сообщений, если задан параметр <code>message_parts</code>), которые указаны с помощью параметра <code>message_set</code>
<code>login(user, password)</code>	Обеспечивает вход в систему пользователя <code>user</code> с применением заданного пароля <code>password</code>
<code>logout()</code>	Выполняет выход из системы сервера.
<code>noop()</code>	Производит эхо-тестирование сервера, но без выполнения каких-либо действий ("пустая операция")
<code>search(charset, *criteria)</code>	Выполняет поиск в почтовом ящике таких сообщений, которые согласуются по крайней мере с одной частью строки поиска <code>criteria</code> . Если параметр <code>charset</code> имеет значение <code>False</code> , то по умолчанию применяется кодировка US-ASCII
<code>select(mailbox='INBOX', read-only=False)</code>	Выбирает почтовый ящик <code>mailbox</code> (значение по умолчанию — <code>INBOX</code>). Пользователю не разрешается изменять его содержимое, если задан параметр <code>readonly</code> (только чтение)

Ниже приведено несколько примеров использования некоторых из этих методов.

- NOP, NOOP или пустая операция. Следующий оператор предназначен для поддержки соединения с сервером:

```
>>> s.noop()
('OK', ['NOOP completed'])
```

- Получение информации о конкретном сообщении:

```
>>> rsp, data = s.fetch('98', '(BODY)')
>>> data[0]
'98 (BODY ("TEXT" "PLAIN" ("CHARSET" "ISO-8859-1" "FORMAT" "flowed" "DELSP" "yes")
NIL NIL "7BIT" 1267 33))'
```

- Получение только заголовков сообщения:

```
>>> rsp, data = s.fetch('98', '(BODY[HEADER])')
>>> data[0][1][:45]
'Received: from mail-gy.google.com (mail-gy.go'
```

- Получение идентификаторов просмотренных сообщений (попытайтесь также воспользоваться ключевыми словами 'ALL', 'NEW' и т.д.):

```
>>> s.search(None, 'SEEN')
('OK', ['1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 59 60 61 62 63 64 97'])
```

- Получение более чем одного сообщения (в качестве разделителя используется двоеточие (:); обратите внимание на то, что для разметки результатов применяется "):

```
>>> rsp, data = s.fetch('98:100', '(BODY[TEXT])')
>>> data[0][1][:45]
'Welcome to Google Accounts. To activate your'
```

```
>>> data[2][1][:45]
'\r\n-bl_aeblac91493d87ea4f2aa7209f56f909\r\nCont'
>>> data[4][1][:45]
'This is a multi-part message in MIME format.'
>>> data[1], data[3], data[5]
(')', ')', ')')
```

3.4.16. Практический пример

Компоновка электронной почты

Выше в данном разделе довольно подробно рассматривались различные способы, которые могут использоваться в языке Python для загрузки сообщений электронной почты. Кроме того, вкратце был затронут вопрос о том, как создавать простые текстовые сообщения электронной почты, а затем подключаться к серверам SMTP для их отправки. Тем не менее не была затронута важная тема, касающаяся того, как создавать немного более сложные сообщения с помощью языка Python. Как и можно было бы предположить, речь идет о сообщениях электронной почты, содержащих информацию в более сложном виде по сравнению с текстовыми файлами, с вложениями, альтернативными форматами и т.д. Теперь мы можем перейти к краткому рассмотрению этой темы.

Более длинные сообщения, которые будут описаны в этом разделе, обычно состоят из нескольких частей, в том числе включают часть сообщения в виде простого текста, которая может быть представлена в формате HTML для отображения в веб-браузерах, поддерживаемых почтовыми клиентами, а также содержат одно или несколько вложений. Во всем мире для идентификации и проведения различий между частями сообщений применяется формат MIME (Mail Interchange Message Extension).

В языке Python предусмотрен пакет `email`, который идеально подходит для обработки и управления частями MIME каждого сообщения электронной почты, вместе взятого, поэтому в настоящем разделе этот пакет будет использоваться наряду, разумеется, с библиотекой `smtpplib`. В пакет `email` входят отдельные компоненты, позволяющие проводить синтаксический анализ входящей электронной почты, а также создавать исходящую. Начнем с решения последней задачи, а в заключение кратко рассмотрим, как осуществляются синтаксический анализ и сквозной контроль сообщений.

В примере 3.4 показаны два способа создания сообщений электронной почты, с помощью методов `make_mpa_msg()` и `make_img_msg()`. В обоих вариантах создается отдельное сообщение электронной почты с одним вложением. В первом сценарии создается одно многокомпонентное альтернативное сообщение и происходит его отправка, а во втором формируется сообщение электронной почты, содержащее одно изображение, после чего происходит его отправка. Вслед за примером приведено его построчное описание.

Пример 3.4. Создание электронной почты (`email-examples.py`)

В этом сценарии Python 2 создаются и отправляются два разных типа сообщений электронной почты.

```
1  #!/usr/bin/env python
2  'email-examples.py - demo creation of email messages'
3
```

```
4  from email.mime.image import MIMEImage
5  from email.mime.multipart import MIMEMultipart
6  from email.mime.text import MIMEText
7  from smtplib import SMTP
8
9  # многокомпонентное сообщение: текст и код HTML
10 def make_mpa_msg():
11     email = MIMEMultipart('alternative')
12     text = MIMEText('Hello World!\r\n', 'plain')
13     email.attach(text)
14     html = MIMEText(
15         '<html><body><h4>Hello World!</h4>'
16         '</body></html>', 'html')
17     email.attach(html)
18     return email
19
20 # многокомпонентное сообщение: изображения
21 def make_img_msg(fn):
22     f = open(fn, 'r')
23     data = f.read()
24     f.close()
25     email = MIMEImage(data, name=fn)
26     email.add_header('Content-Disposition',
27         'attachment; filename="%s"' % fn)
28     return email
29
30 def sendMsg(fr, to, msg):
31     s = SMTP('localhost')
32     errs = s.sendmail(fr, to, msg)
33     s.quit()
34
35 if __name__ == '__main__':
36     print 'Sending multipart alternative msg...'
37     msg = make_mpa_msg()
38     msg['From'] = SENDER
39     msg['To'] = ', '.join(RECIPS)
40     msg['Subject'] = 'multipart alternative test'
41     sendMsg(SENDER, RECIPS, msg.as_string())
42
43     print 'Sending image msg...'
44     msg = make_img_msg(SOME_IMG_FILE)
45     msg['From'] = SENDER
46     msg['To'] = ', '.join(RECIPS)
47     msg['Subject'] = 'image file test'
48     sendMsg(SENDER, RECIPS, msg.as_string())
```

Построчное объяснение

Строки 1–7

В этом сценарии, кроме стандартных начальных строк и объявления строк документации, выполняется импорт классов `MIMEImage`, `MIMEMultipart`, `MIMEText` и `SMTP`.

Строки 9–18

Многокомпонентные альтернативные сообщения обычно состоят из следующих двух частей: текста сообщения электронной почты в виде простого текста и его

эквивалента в коде HTML. Задача выбора для отображения одного из альтернативных вариантов сообщения возлагается на почтовый клиент. Например, в почтовой системе с веб-интерфейсом, как правило, отображается версия в коде HTML, тогда как программа чтения почты с интерфейсом командной строки обычно показывает только версию с текстовым файлом.

Для создания сообщения этого типа необходимо использовать класс `email.mime.multiple.MIMEMultipart` и создавать экземпляр этого класса, передавая ключевое слово `'alternative'` в качестве единственного параметра. Если же указанное ключевое слово не будет задано, то каждый из альтернативных вариантов будет представлен в качестве отдельного вложения. Таким образом, в некоторых почтовых системах могут быть показаны обе части.

В данном примере класс `email.mime.text.MIMEText` использовался для обеих частей (поскольку в действительности обе эти части представляют собой текст сообщения в виде открытого текста). Затем каждая часть присоединяется ко всему сообщению электронной почты, поскольку эти части создаются перед возвратом итогового сообщения.

Строки 20–28

Функция `make_img_msg()` принимает единственный параметр — имя файла. Содержимое этого файла считывается, а затем передается непосредственно в новый экземпляр `email.mime.image.MIMEImage`. Добавляется заголовок `Content-Disposition`, после чего сообщение возвращается пользователю.

Строки 30–33

Назначение метода `sendMsg()` состоит исключительно в том, чтобы получить основные данные, необходимые для отправки электронной почты (имя отправителя, имена получателей, текст сообщения), передать сообщение, а затем вернуть полученные результаты в точку вызова.

Иногда может потребоваться получение более подробного вывода. В таком случае можно попытаться воспользоваться следующим расширением: `s.set_debuglevel(True)`, где `s` — сервер `smtplib.SMTP`. Наконец, еще раз отметим, что на многих серверах SMTP предусмотрено использование учетных записей, поэтому здесь проводится соответствующая подготовка (непосредственно после входа в систему, но перед отправкой сообщения электронной почты).

Строки 35–48

В основной части этого сценария выполняется лишь проверка каждой из двух указанных функций. С помощью функций создается сообщение, добавляются поля `"From"`, `"To"` и `"Sender"`, а затем сообщение передается указанным получателям. Безусловно, приложение будет работать лишь при том условии, что следующие параметры получат определенные значения: `SENDER`, `RECIPS` и `SOME_IMG_FILE`.

Синтаксический анализ электронной почты

Задача синтаксического анализа немного проще по сравнению с формированием сообщения с нуля. Для этого, как правило, используются несколько инструментов из пакета `email`: функция `email.message_from_string()`, а также методы `message.walk()` и `message.get_payload()`. Типичная последовательность действий иллюстрируется в следующем коде:

```
def processMsg(entire_msg):
    body = ''
    msg = email.message_from_string(entire_msg)
    if msg.is_multipart():
        for part in msg.walk():
            if part.get_content_type() == 'text/plain':
                body = part.get_payload()
                break
    else:
        body = msg.get_payload(decode=True)
    else:
        body = msg.get_payload(decode=True)
    return body
```

Этот фрагмент кода является достаточно простым для изучения. Ниже показаны основные используемые методы.

- `email.message_from_string()`. Применяется для синтаксического анализа сообщения.
- `msg.walk()`. Обеспечивает рассмотрение древовидной иерархии вложений электронной почты сверху вниз.
- `part.get_content_type()`. Позволяет однозначно установить применяемый тип MIME.
- `msg.get_payload()`. Обеспечивает получение определенной части из текста сообщения. Как правило, флаг декодирования устанавливается равным `True`, чтобы в процессе синтаксического анализа письма осуществлялось декодирование его текста в соответствии с заголовком `Content-Transfer-Encoding`.

Службы электронной почты с веб-интерфейсом на основе облака

Работа с помощью протоколов, которые были описаны выше в данной главе, как правило, приводит к получению идеальных результатов: нам не приходится слишком заботиться о проблемах безопасности или преодолевать сложности, связанные с применением средств защиты. Разумеется, выше в этой главе уже было сказано, что для некоторых серверов требуется применение учетных записей.

Тем не менее, приступая к решению задач программирования на практике, приходится сталкиваться с тем, что службы сопровождения активно поддерживаемых серверов стремятся всеми силами предотвратить ситуации, в которых сервер мог бы стать для злоумышленников инструментом, применяемым для рассылки спама, ретрансляции фишинговой электронной почты или других злонамеренных действий. В таких системах, большую часть которых составляют почтовые системы, применяются соответствующие средства защиты. Примеры работы с электронной почтой, приведенные выше в данном разделе, относятся к службам электронной почты общего назначения, доступ к которым предоставляет любой провайдер служб Интернета. Абоненты вносят ежемесячную оплату за пользование услугами Интернета, а что касается возможности передачи (отправки) или загрузки (получения) электронной почты, то принято считать, что они предоставляются “бесплатно”.

Рассмотрим некоторые общедоступные службы электронной почты с веб-интерфейсом, такие как Yahoo! Mail и Gmail компании Google. Это облачные службы, доступ к которым предоставляется по принципу SaaS (software as a service — программное обеспечение как услуга), и при этом не предъявляются требования по

ежемесячной оплате, поэтому пользователям кажется, что эти услуги — полностью бесплатные. Тем не менее пользователи “платят” тем, что подвергаются воздействию рекламы и не могут от нее избавиться. Безусловно, провайдеры служб стремятся повысить релевантность рекламы (т.е. ее соответствие аудитории), и чем лучше это удастся, тем больше вероятность успешного возмещения провайдерами затрат на бесплатное предоставление некоторых услуг пользователям.

В службе Gmail предусмотрены алгоритмы, с помощью которых осуществляется просмотр сообщений электронной почты для выявления их смысла, чтобы с применением качественных алгоритмов машинного обучения выработать рекомендации по демонстрации рекламы, которая с большей вероятностью заденет чувства пользователя, чем случайно выбранные объявления из общего массива. Рекламные объявления, как правило, бывают оформленными в виде простого текста и располагаются вдоль правой стороны окна с сообщением электронной почты. Эти алгоритмы Google оказались очень эффективными, поэтому данная компания не только предоставляет бесплатный доступ к своей службе Gmail с веб-интерфейсом, но и позволяет использовать клиентские программы для получения сообщений электронной почты из служб других провайдеров по протоколам POP3 и IMAP4, а также отправлять электронную почту с помощью протокола SMTP.

Компания Yahoo!, с другой стороны, демонстрирует рекламные объявления общего характера в формате изображения, встраивая их в различные части своего веб-приложения. Реклама этой компании не является столь целенаправленной, как реклама Google, поэтому не приносит сравнимый доход. Это может явиться причиной того, что Yahoo! предлагает службу с платной подпиской (называемую Yahoo! Mail Plus) для загрузки электронной почты. Еще одна причина предоставления только платной подписки заключается в том, что после оформления такой подписки пользователи не желают терять потраченные деньги и становятся менее склонными к тому, чтобы переходить к использованию какой-то другой почтовой службы. Ко времени написания данной книги компания Yahoo! предоставляла услуги по отправке электронной почты с помощью протокола SMTP бесплатно. Примеры кода для того и другого варианта приведены в конце данного раздела.

Рекомендации по обеспечению безопасности и рефакторингу

Мы должны уделить определенное время рассмотрению рекомендаций по обеспечению безопасности и рефакторингу. Иногда даже самые тщательно продуманные планы срываются при столкновении с реальностью. В программировании это может быть обусловлено тем, что появляются другие версии языка программирования с усовершенствованиями и исправлениями, которых не было в предыдущих версиях, поэтому приходится выполнять немного больший объем работы по сравнению с первоначально запланированным.

Прежде чем приступить к изучению двух служб электронной почты, предоставляемых компаниями Google и Yahoo!, рассмотрим некоторый шаблонный код, который будет использоваться для каждого ряда примеров:

```
from imaplib import IMAP4_SSL
from poplib import POP3_SSL
from smtplib import SMTP_SSL

from secret import * # получаем MAILBOX и PASSWD

who = . . . # xxx@yahoo/gmail.com, где MAILBOX = xxx
```



```
from_ = who
to = [who]

headers = [
    'From: %s' % from_,
    'To: %s' % ', '.join(to),
    'Subject: test SMTP send via 465/SSL',
]

body = [
    'Hello',
    'World!',
]

msg = '\r\n\r\n'.join((''\r\n'.join(headers), '\r\n'.join(body)))
```

Прежде всего заслуживает внимания то, что разрабатываемые сценарии больше не рассчитаны на гипотетическую ситуацию, а также то, что мы теперь с помощью Интернета решаем и личные, и производственные, а иногда даже такие жизненно важные задачи, которые требуют применения безопасных соединений. Поэтому во всех случаях используются варианты этих трех протоколов с поддержкой SSL. Это выражается в том, что к концу каждого из исходных имен классов присоединяется префикс `_SSL`.

Кроме того, в этих практических примерах, в отличие от примеров программ, приведенных в предыдущих разделах, нельзя больше использовать имена почтовых ящиков (регистрационные имена) и пароли в виде открытого текста. В действительности то, что мы позволяли себе — помещать учетные имена и пароли в текстовый файл и включать их в исходный код, — это просто ужасно, если не сказать больше. В реальном коде следует обеспечить извлечение этих регистрационных данных из защищенной базы данных, импортировать их из откомпилированного в шестнадцатеричный код файла `.рус` или `.руо` или получать в оперативном режиме от сервера или брокера, размещенного непосредственно во внутренней сети компании. Что касается примеров, приведенных в этой главе, то будем предполагать, что регистрационные данные находятся в файле `secret.рус`, содержащем атрибуты `MAILBOX` и `PASSWD`, связанные с эквивалентной информацией о правах доступа.

Последний набор переменных представляет лишь фактическое сообщение электронной почты, а также имена отправителя и получателя (в целях упрощения работы было предусмотрено, что все эти данные относятся к одним и тем же людям). Кроме того, будет применяться немного более сложный способ структуризации самого сообщения электронной почты по сравнению с принятым ранее, когда текст состоял из одной строки и требовалось ввести в поля необходимые данные:

```
body = '''\
From: %(who)s
To: %(who)s
Subject: test msg

Hello World!
''' % {'who': who}
```

Вместо этого мы решили перейти к использованию списков, поскольку более велика вероятность того, что на практике текст сообщения электронной почты будет формироваться или каким-то образом контролироваться с помощью приложения, а

не оставаться жестко закодированной строкой. То же касается и заголовков электронной почты. После перехода к использованию списков становится проще решать задачи добавления (или даже удаления) строк в ходе работы с сообщением электронной почты. Затем, когда все будет готово для передачи, достаточно лишь выполнить несколько вызовов метода `str.join()`, задавая в качестве разделителя строк пары символов `\r\n`. (Напомним, что в предыдущем разделе этой главы уже было сказано, что этот разделитель официально утвержден документом RFC 5322 для использования в серверах SMTP, хотя некоторые серверы принимают в качестве разделителя лишь единственный символ обозначения новой строки, `\n`.)

В этот сценарий будет внесено еще одно небольшое дополнение, касающееся обработки данных текста сообщения: количество получателей может быть больше одного, поэтому переменная `to` также должна быть преобразована в список. Затем с помощью вызова `str.join()` происходит соединение элементов списка в одну строку и создается набор заголовков электронной почты в окончательном виде. Наконец, рассмотрим определенную вспомогательную функцию, которая нам потребуется в приведенных ниже примерах работы со службами Yahoo! Mail и Gmail. С помощью этой функции создается небольшой фрагмент кода, назначение которого состоит лишь в извлечении строки темы из входящих сообщений электронной почты.

```
def getSubject(msg, default='(no Subject line)'):
    '''
    getSubject(msg) - 'msg' is an iterable, not a
    delimited single string; this function iterates
    over 'msg' look for Subject: line and returns
    if found, else the default is returned if one isn't
    found in the headers
    '''
    for line in msg:
        if line.startswith('Subject:'):
            return line.rstrip()
    if not line:
        return default
```

Функция `getSubject()` является не очень сложной; с ее помощью осуществляется поиск строки темы только в заголовках. Сразу после обнаружения искомой информации происходит возврат из функции. Признаком конца заголовков является пустая строка, поэтому, если к этому моменту строка темы не обнаруживается, функция возвращает значение по умолчанию, заданное с помощью локальной переменной с заданным по умолчанию параметром; это позволяет пользователю при желании передавать какую-то стандартную строку темы. Мой опыт показывает, что программисты, стремящиеся во всем добиваться повышения производительности, в этом случае предпочли бы использовать оператор `line[:8] == 'Subject:'`, избегая вызова метода `str.startswith()`, но является ли выигрыш столь значительным? Не следует забывать, что применение переменной `line[:8]` приводит к вызову `str.__getslice__()`. Разумеется, несмотря на это, в данном варианте выполнение оператора происходит приблизительно на 40% быстрее по сравнению с вызовом `str.startswith()`, как показывают проверки с помощью функции `timeit`:

```
>>> t = timeit.Timer('s[:8] == "Subject:"', 's="Subject: xxx"')
>>> t.timeit()
0.14157199859619141
>>> t.timeit()
```

```
0.1387479305267334
>>> t.timeit()
0.13623881340026855
>>>
>>> t = timeit.Timer('s.startswith("Subject:");', 's="Subject: xxx"')
>>> t.timeit()
0.23016810417175293
>>> t.timeit()
0.23104190826416016
>>> t.timeit()
0.24139499664306641
```

Очевидно, что функция `timeit` заслуживает того, чтобы взять ее на вооружение, и приведенный выше пример характеризует один из наиболее часто применяемых вариантов ее использования: имеется два или несколько фрагментов кода, предназначенных для решения одной и той же задачи, поэтому необходимо определить, какой из этих фрагментов является более эффективным. Теперь перейдем к рассмотрению того, как применить часть полученных нами знаний в реальном коде.

Служба Yahoo! Mail

Примем предположение, что весь приведенный выше стандартный код успешно выполняется, и приступим к работе со службой Yahoo! Mail. На данном этапе будет рассматриваться код, представляющий собой расширение примера 3.3. Кроме того, отметим, что электронная почта будет передаваться с помощью протокола SMTP, а для получения сообщений вначале воспользуемся протоколом POP, а затем IMAP. Ниже приведен сценарий, который будет принят в качестве прототипа.

```
s = SMTP_SSL('smtp.mail.yahoo.com', 465)
s.login(MAILBOX, PASSWD)
s.sendmail(from_, to, msg)
s.quit()
print 'SSL: mail sent!'

s = POP3_SSL('pop.mail.yahoo.com', 995)
s.user(MAILBOX)
s.pass_(PASSWD)
rv, msg, sz = s.retr(s.stat()[0])
s.quit()

line = getSubject(msg)
print 'POP:', line

s = IMAP4_SSL('imap.n.mail.yahoo.com', 993)
s.login(MAILBOX, PASSWD)
rsp, msgs = s.select('INBOX', True)
rsp, data = s.fetch(msgs[0], '(RFC822)')
line = getSubject(StringIO(data[0][1]))
s.close()
s.logout()
print 'IMAP:', line
```

При условии, что весь необходимый код собран в файле `ymail.py`, результаты выполнения сценария могут выглядеть примерно так:

```
$ python ymail.py
SSL mail sent!
POP: Subject:Meet singles for dating, romance and more.
IMAP: Subject: test SMTP send via 465/SSL
```

2.6

В данном случае предусмотрено использование учетной записи Yahoo! Mail Plus, которая позволяет загружать электронную почту. (Возможность передачи почты через службу Yahoo! Mail Plus предоставляется бесплатно всем подписчикам, независимо от того, оформлена ли ими платная или бесплатная подписка.) Однако в ходе выполнения этого примера обнаружился определенный ошибки. Прежде всего, сообщение, полученное с помощью протокола POP, не соответствовало нашему отправленному сообщению, тогда как протокол IMAP оказался способным найти правильный вариант сообщения. Вообще говоря, протокол IMAP на практике показывает более высокую надежность. Кроме того, в предыдущем примере предполагалось, что отправку письма осуществляет клиент с платной подпиской, который использует текущую версию Python (версию не ниже 2.6.3); если эти условия не соблюдаются, то результаты становятся еще хуже.

Клиенты, которые не платят за Yahoo! Mail Plus, не имеют право загружать электронную почту. Ниже приведен типичный пример обратной трассировки, которая формируется при попытке сделать именно это.

```
Traceback (most recent call last):
  File "ymail.py", line 101, in <module>
    s.pass_(PASSWD)
  File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/poplib.py",
line 189, in pass_
    return self._shortcmd('PASS %s' % pswd)
  File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/poplib.py",
line 152, in _shortcmd
    return self._getresp()
  File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/poplib.py",
line 128, in _getresp
    raise error_proto(resp)
poplib.error_proto: -ERR [SYS/PERM] pop not allowed for user.
```

Кроме того, класс SMTP_SSL был добавлен только в версии 2.6, и, вдобавок ко всему, его реализация содержала программные ошибки, не исправленные до выхода версии 2.6.3, поэтому лишь с помощью последней версии (а также следующих за ней) можно написать код, в котором используется протокол SMTP с поддержкой SSL. Если используется версия, предшествующая версии 2.6, то не удастся даже создать экземпляр необходимого класса, а если работа проводится с версиями 2.6(0)–2.6.2, то обнаруживается ошибка, которая выглядит примерно так:

```
Traceback (most recent call last):
  File "ymail.py", line 61, in <module>
    s.login(MAILBOX, PASSWD)
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/smtplib.py",
line 549, in login
    self.ehlo_or_helo_if_needed()
  File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/smtplib.py",
line 509, in ehlo_or_helo_if_needed
    if not (200 <= self.ehlo()[0] <= 299):
```

```

File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/smtplib.py", line 382, in ehlo
    self.putcmd(self.ehlo_msg, name or self.local_hostname)
File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/smtplib.py", line 318, in putcmd
    self.send(str)
File "/System/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/smtplib.py", line 310, in send
    raise SMTPServerDisconnected('please run connect() first')
smtplib.SMTPServerDisconnected: please run connect() first

```

Это лишь часть проблем, с которыми приходится сталкиваться на практике; а самое главное — то, что в действительности условия работы никогда не бывают столь идеальными, какими принято их представлять в учебнике. Всегда возникают какие-то непонятные, непредвиденные нарушения, которые буквально ставят вас в тупик. В этой книге мы постоянно стремимся ознакомить читателя с возможными затруднениями, в надежде на то, что это поможет легче преодолеть их на практике.

Попытаемся представить полученный вывод так, чтобы он был немного проще для восприятия. Что еще более важно, введем в сценарий необходимые проверки (в том числе номеров версий). Учитывая то, что, как оказалось, невозможно обеспечить успешное применение сценария, не зная, в какой версии он эксплуатируется, настала пора осваивать профессиональную привычку дополнять этим свои программы. Окончательная версия сценария `yumail.py` приведена в примере 3.5.

Пример 3.5. Применение протоколов SMTP, POP, IMAP для работы со службой Yahoo! Mail (`yumail.py`)

Этот сценарий показывает, как работать с помощью протоколов SMTP, POP и IMAP со службой Yahoo! Mail.

```

1  #!/usr/bin/env python
2  'yumail.py - демо Yahoo!Mail SMTP/SSL, POP, IMAP'
3
4  from cStringIO import StringIO
5  from imaplib import IMAP4_SSL
6  from platform import python_version
7  from poplib import POP3_SSL, error_proto
8  from socket import error
9
10 # SMTP SSL добавлен в 2.6, исправлен в 2.6.3
11 release = python_version()
12 if release > '2.6.2':
13     from smtplib import SMTP_SSL, SMTPServerDisconnected
14 else:
15     SMTP_SSL = None
16
17 from secret import * # задаем MAILBOX и PASSWD
18
19 who = '%s@yahoo.com' % MAILBOX
20 from_ = who
21 to = [who]
22
23 headers = [
24     'From: %s' % from_,
25     'To: %s' % ', '.join(to),
26     'Subject: test SMTP send via 465/SSL',

```

```

27 ]
28 body = [
29     'Hello',
30     'World!',
31 ]
32 msg = '\r\n\r\n'.join(('r\n'.join(headers), '\r\n'.join(body)))
33
34 def getSubject(msg, default='(no Subject line)'):
35     '''\
36     getSubject(msg) - iterate over 'msg' looking for
37     Subject line; return if found otherwise 'default'
38     '''
39     for line in msg:
40         if line.startswith('Subject:'):
41             return line.rstrip()
42         if not line:
43             return default
44
45 # SMTP/SSL
46 print '*** Doing SMTP send via SSL...'
47 if SMTP_SSL:
48     try:
49         s = SMTP_SSL('smtp.mail.yahoo.com', 465)
50         s.login(MAILBOX, PASSWD)
51         s.sendmail(from_, to, msg)
52         s.quit()
53         print '    SSL mail sent!'
54     except SMTPServerDisconnected:
55         print '    error: server unexpectedly disconnected... try again'
56 else:
57     print '    error: SMTP_SSL requires 2.6.3+'
58
59 # POP
60 print '*** Doing POP recv...'
61 try:
62     s = POP3_SSL('pop.mail.yahoo.com', 995)
63     s.user(MAILBOX)
64     s.pass_(PASSWD)
65     rv, msg, sz = s.retr(s.stat()[0])
66     s.quit()
67     line = getSubject(msg)
68     print '    Received msg via POP: %r' % line
69 except error_proto:
70     print '    error: POP for Yahoo!Mail Plus subscribers only'
71
72 # IMAP
73 print '*** Doing IMAP recv...'
74 try:
75     s = IMAP4_SSL('imap.n.mail.yahoo.com', 993)
76     s.login(MAILBOX, PASSWD)
77     rsp, msgs = s.select('INBOX', True)
78     rsp, data = s.fetch(msgs[0], '(RFC822)')
79     line = getSubject(StringIO(data[0][1]))
80     s.close()
81     s.logout()
82     print '    Received msg via IMAP: %r' % line
83 except error:
84     print '    error: IMAP for Yahoo!Mail Plus subscribers only'

```

Построчное объяснение

Строки 1–8

В этой части сценария, как обычно, находятся строки его заголовка и импорта.

Строки 10–15

В этом фрагменте кода запрашивается номер версии языка Python в виде строки, для получения которой используется вызов `platform.python_version()`. Импорт атрибутов `smtplib` выполняется только при использовании версии 2.6.3 и последующих; в противном случае в качестве `SMTP_SSL` задается значение `None`.

Строки 17–21

Как уже было сказано выше, информация, которая служит для получения прав доступа, такая как имя входа и пароль, не задается жестко в коде, а сохраняется в каком-то другом месте, например, в откомпилированном в шестнадцатеричном коде файле `secret.рус`, не позволяющем извлечь данные `PASSWD` и `MAILBOX` рядовому пользователю. Это приложение применяется исключительно в качестве тестового, поэтому после получения указанной информации (строка 17) устанавливаются значения переменных с данными об отправителе и получателе конверта, которые указывают на одно и то же лицо (строки 19–21). Кстати, почему для переменной с обозначением отправителя мы применили имя `from_`, а не `from`?

Строки 23–32

Этот приведенный далее набор строк применяется для составления текста сообщения электронной почты. Строки 23–27 представляют заголовки (которые могут быть легко сформированы с помощью некоторого кода), строки 28–31 соответствуют фактическому тексту сообщения (который также может быть сформирован или взят в готовом виде). В конце (строка 32) находится строка кода, с помощью которой осуществляется объединение всей подготовленной ранее информации (включая заголовки и текст) и создается весь текст сообщения электронной почты с использованием требуемых разделителей.

Строки 34–43

Выше в данной главе уже шла речь о функции `getSubject()`, единственным назначением которой является поиск строки темы в заголовках входящего сообщения электронной почты и подстановка строки, заданной по умолчанию, если строка темы в письме не определена. Все необходимые варианты выбора подготовлены, поскольку значение по умолчанию для переменной `default` задано.

Строки 45–57

Это код `SMTP`. Ранее, в строках 10–15, проводилась проверка для определения того, следует ли использовать модуль `SMTP_SSL` или присвоить соответствующей переменной значение `None`. В данном фрагменте, если необходимый класс получен (строка 7), предпринимается попытка подключиться к серверу, войти в систему, отправить электронную почту, а затем завершить работу (строки 48–53). В противном случае следует предупредить пользователя, что требуется версия 2.6.3 или более новая (строки 56–57). Иногда может происходить разрыв соединения с сервером по ряду причин, таких как некачественное соединение и т.д. В подобных случаях обычно

удается достичь цели с помощью повторных попыток, поэтому пользователь получает сообщение о том, что такая попытка осуществляется (строки 54-55).

Строки 59–70

Это код для работы с протоколом POP3, который выше в данной главе уже рассматривался достаточно подробно (строки 62–68). Единственное различие состоит в том, что была добавлена проверка на тот случай, что доступ по протоколу POP клиентом не был оплачен, но он все равно пытается загрузить свою почту. Поэтому необходимо предусмотреть перехват исключения `poplib.error_proto` (строки 69-70), как было показано ранее.

Строки 72–84

Те же пояснения относятся к коду работы с протоколом IMAP4. Эти основные функциональные средства заключены в блок `try` (строки 74-82), и предусмотрен перехват исключения `socket.error` (строки 83-84). Внимательный читатель мог заметить, насколько удачно здесь используется объект `cStringIO.StringIO` (строка 79). Это обусловлено тем, что протокол IMAP возвращает все сообщение электронной почты в виде одной большой строки. С другой стороны, в функции `getSubject()` предусмотрена обработка в цикле нескольких строк, поэтому необходимо предоставить данные этой функции в таком виде, чтобы она могла с ними работать. Этой цели позволяет добиться объект `StringIO`, который получает длинную строку и создает интерфейс к этой строке, подобный применяемому для работы с файлом.

Отметим также, что на практике все действия, осуществляемые при работе со службой Yahoo! Mail, являются вполне применимыми для работы со службой Gmail, не считая того, что эта служба предоставляет все возможности доступа бесплатно. Кроме того, отметим, что в службе Gmail разрешается также применять стандартный протокол SMTP (с использованием TLS).

Служба Gmail

Сценарий для работы со службой Gmail компании Google рассматривается в примере 3.6. В службе Gmail не только обеспечивается работа по протоколу SMTP с поддержкой SSL, но и предоставляется возможность работать по протоколу SMTP с использованием средств TLS (Transport Layer Security), поэтому в сценарии дополнительно осуществляется импорт класса `smtpplib.SMTP`, с которым связан отдельный раздел кода. Как и все остальное (включая SMTP с поддержкой SSL, POP и IMAP), эти фрагменты кода весьма напоминают эквивалентный код для Yahoo! Mail. Возможность загрузки электронной почты предоставляется полностью бесплатно, поэтому нет необходимости предусматривать обработку исключений для возобновления работы после возникновения ошибок доступа, которые активизируются при обращении к службе тех, кто не является подписчиком.

Пример 3.6. Применение протоколов SMTPx2, POP, IMAP для доступа к службе Gmail (`gmail.py`)

В этом сценарии показано, как использовать протоколы SMTP, POP и IMAP для интерактивного доступа к службе электронной почты Gmail компании Google.


```

1  #!/usr/bin/env python
2  'gmail.py - demo GMail SMTP/TLS, SMTP/SSL, POP, IMAP'
3
4  from cStringIO import StringIO
5  from imaplib import IMAP4_SSL
6  from platform import python_version
7  from poplib import POP3_SSL
8  from smtplib import SMTP
9
10 # поддержка SMTP_SSL добавлена в версии 2.6
11 release = python_version()
12 if release > '2.6.2':
13     from smtplib import SMTP_SSL # исправлено в 2.6.3
14 else:
15     SMTP_SSL = None
16
17 from secret import * # задаем MAILBOX и PASSWD
18
19 who = '%s@gmail.com' % MAILBOX
20 from_ = who
21 to = [who]
22
23 headers = [
24     'From: %s' % from_,
25     'To: %s' % ', '.join(to),
26     'Subject: test SMTP send via 587/TLS',
27 ]
28 body = [
29     'Hello',
30     'World!',
31 ]
32 msg = '\r\n\r\n'.join((''.join(headers), '\r\n'.join(body)))
33
34 def getSubject(msg, default='(no Subject line)'):
35     '''
36     getSubject(msg) - iterate over 'msg' looking for
37     Subject line; return if found otherwise 'default'
38     '''
39     for line in msg:
40         if line.startswith('Subject:'):
41             return line.rstrip()
42     if not line:
43         return default
44
45 # SMTP/TLS
46 print '*** Doing SMTP send via TLS...'
47 s = SMTP('smtp.gmail.com', 587)
48 if release < '2.6':
49     s.ehlo() # в старых версиях
50 s.starttls()
51 if release < '2.5':
52     s.ehlo() # в старых версиях
53 s.login(MAILBOX, PASSWD)
54 s.sendmail(from_, to, msg)
55 s.quit()
56 print '    TLS mail sent!'
57
58 # POP

```

```

59 print '*** Doing POP recv...'
60 s = POP3_SSL('pop.gmail.com', 995)
61 s.user(MAILBOX)
62 s.pass_(PASSWD)
63 rv, msg, sz = s.retr(s.stat()[0])
64 s.quit()
65 line = getSubject(msg)
66 print '    Received msg via POP: %r' % line
67
68 body = body.replace('587/TLS', '465/SSL')
69
70 # SMTP/SSL
71 if SMTP_SSL:
72     print '*** Doing SMTP send via SSL...'
73     s = SMTP_SSL('smtp.gmail.com', 465)
74     s.login(MAILBOX, PASSWD)
75     s.sendmail(from_, to, msg)
76     s.quit()
77     print '    SSL mail sent!'
78
79 # IMAP
80 print '*** Doing IMAP recv...'
81 s = IMAP4_SSL('imap.gmail.com', 993)
82 s.login(MAILBOX, PASSWD)
83 rsp, msgs = s.select('INBOX', True)
84 rsp, data = s.fetch(msgs[0], '(RFC822)')
85 line = getSubject(StringIO(data[0][1]))
86 s.close()
87 s.logout()
88 print '    Received msg via IMAP: %r' % line

```

Построчное объяснение

Строки 1–8

В этом коде предусмотрены обычные строки заголовка и импорта с одним дополнением: импорт библиотеки `smtplib.SMTP`. Этот класс будет использоваться со средствами TLS для отправки сообщений электронной почты.

Строки 10–43

В целом по своему содержанию этот сценарий весьма напоминает сценарий `umail.py`. Одно из различий состоит в том, что переменная `who` содержит, как и следовало ожидать, адрес электронной почты в виде `@gmail.com` (строка 19). Еще одно изменение связано с переходом на использование SMTP/TLS, что отражается в строке темы. Кроме того, не производится импорт средств обработки исключения `smtplib.SMTPServerDisconnected`, поскольку мы ни разу не обнаруживали это исключение на протяжении всех проверок.

Строки 45–56

Это код поддержки протокола SMTP, который обеспечивает подключение к серверу с использованием средств TLS. Вполне очевидно, что по мере появления все новых и новых версия языка Python (строки 48–52) уменьшается необходимость в

использовании вспомогательного кода для обеспечения обмена данными с сервером. Кроме того, применяется другой номер порта по сравнению с SMTP/SSL (строка 47).

Строки 58–88

Остальная часть сценария почти идентична своему эквиваленту для Yahoo! Mail. Как уже было отмечено выше, в этом сценарии уменьшен объем кода проверки ошибок, поскольку при работе со службой Gmail такие, как раньше, ошибки не возникают или не обнаруживаются. Наконец, еще одно небольшое отличие состоит в том, что в связи с необходимостью отправки и сообщений SMTP/TLS, и сообщений SMTP/SSL приходится немного корректировать строку темы (строка 68).

Мы надеемся, что читатели в результате изучения последних двух приложений смогут ознакомиться с практическим применением концепций, изложенных ранее в этой главе, узнают, как использовать эти знания в своей повседневной работе по разработке приложений и обеспечить на практике реализацию средств защиты. Кроме того, эти приложения показывают, в чем состоят небольшие различия между версиями языка Python. Безусловно, хотелось бы, чтобы программная реализация была более универсальной, но, вполне очевидно, что это не осуществимо, и на практике в любом проекте разработки приходится учитывать множество ограничений и решать целый ряд проблем, лишь часть которых мы смогли затронуть в данной главе.

3.5. Связанные модули

Одно из самых значительных преимуществ языка Python заключается в том, что в его стандартной библиотеке предусмотрены мощные средства поддержки сетевого взаимодействия, которые являются особенно удобными для работы с протоколами Интернета и для создания клиентских программ. В следующих разделах представлены некоторые модули, относящиеся к средствам сетевой поддержки. В первую очередь рассматриваются модули для работы с электронной почтой, а за ними следует описание средств поддержки протоколов Интернета.

3.5.1. Электронная почта

Отличительной особенностью языка Python является то, что в этом языке предусмотрено много модулей и пакетов, обеспечивающих работу электронной почты, с помощью которых можно легко создать необходимое приложение. Некоторые из них перечислены в табл. 3.6.

Таблица 3.6. Модули, предназначенные для работы с электронной почтой

Модуль/пакет	Описание
email	Пакет, который может применяться для обработки электронной почты (поддерживает также MIME)
smtpd	Сервер SMTP
base64	Средства кодирования данных по основанию 16, 32 и 64 (документ RFC 3548)
mhlib	Классы для обработки папок и сообщений MH
mailbox	Классы для обеспечения синтаксического анализа форматов файлов почтового ящика
mailcap	Поддержка средств обработки файлов mailcap

Модуль/пакет	Описание
mimetools	(Устаревшие.) Средства синтаксического анализа сообщений в формате MIME (используют email)
mimetypes	Средства преобразования имен файлов/URL и соответствующих типов MIME
MimeWriter	(Устаревшие.) Средства обработки сообщений в формате MIME (используют email)
mimify	(Устаревшие.) Инструменты, с помощью которых могут обрабатываться сообщения в формате MIME (используют email)
quopri	Средства кодирования и декодирования данных типа quoted-printable стандарта MIME
binascii	Средства преобразования двоичного кода и кода ASCII
binhex	Средства поддержки кодирования и декодирования текста в формате Binhex4

3.5.2. Другие клиентские средства поддержки протоколов Интернета

В табл. 3.7 представлены другие клиентские модули поддержки протоколов Интернета.

Таблица 3.7. Клиентские модули поддержки протоколов Интернета

Модуль	Описание
ftplib	Клиент протокола FTP
xmlrpclib	Клиент протокола XML-RPC
httplib	Клиент протокола HTTP и HTTPS
imaplib	Клиент протокола IMAP4
nntplib	Клиент протокола NNTP
poplib	Клиент протокола POP3
smtplib	Клиент протокола SMTP

3.6. Упражнения

FTP

- 3.1. *Пример простой клиентской программы для работы по протоколу FTP.* Используя примеры для протокола FTP, приведенные в настоящей главе, напишите небольшую клиентскую программу для протокола FTP, с помощью которой вы смогли бы переходить на свои предпочтительные веб-сайты и загружать необходимые вам новейшие версии приложений. Эта программа может быть представлена в виде сценария, который можно время от времени вызывать на выполнение для обеспечения того, чтобы на компьютере использовались новейшие и наилучшие версии установленного программного обеспечения. Может потребоваться обеспечить сопровождение своего рода таблицы с указанием адреса сервера FTP, учетной записи и пароля для повышения удобства работы с программой.

- 3.2. *Пример простой клиентской программы для работы по протоколу FTP и сопоставления с шаблонами.* Используйте свое решение упражнения 3.1 в качестве исходной точки для создания еще одного простого клиента FTP, который позволяет выгружать или загружать ряд файлов с удаленного узла с использованием шаблонов. Например, если возникает необходимость перемещать наборы файлов, таких как файлы исходного кода Python или документы PDF, с одного узла на другой, то можно предусмотреть такую возможность, чтобы пользователи вводили шаблоны *.py или doc*.pdf и передавали только файлы, имена которых согласуются с шаблонами.
- 3.3. *Пример обладающей дополнительными функциями клиентской программы с командной строкой для работы по протоколу FTP (Smart FTP).* Создайте приложение FTP с командной строкой, аналогичное стандартной программе /bin/ftp системы Unix, но добейтесь того, чтобы этот клиент FTP был лучше. Под этим подразумевается наличие в программе дополнительных полезных средств. Для ознакомления с подобными возможностями рассмотрите приложение ncFTP. Его можно найти по адресу <http://ncftp.com>. Например, в этом приложении предусмотрены следующие дополнительные средства: ведение журнала, закладки (позволяющие сохранять адреса серверов FTP, а также имена входа и пароли), индикаторы хода выполнения работы и т.д. При этом, в частности, может потребоваться реализовать функциональные возможности библиотеки readline для ведения журнала и библиотеки curses для управления экраном.
- 3.4. *FTP и многопоточная организация работы.* Создайте клиент FTP, в котором используются потоки Python для загрузки файлов. При этом можно либо доработать существующий клиент Smart FTP, как в упражнении 3.3, либо написать более простой клиент для загрузки файлов. Это может быть программа с интерфейсом командной строки, который позволяет указывать несколько файлов в качестве параметров вызова, или программа с графическим интерфейсом пользователя, в котором пользователю предоставляется возможность выбрать не менее одного файла для передачи. **Дополнительное задание.** Предусмотрите возможность применения шаблонов, таких как *.exe. Используйте отдельные потоки для загрузки каждого файла.
- 3.5. *FTP и графический интерфейс пользователя.* Возвратитесь к обладающему дополнительными возможностями клиенту FTP, который был разработан ранее, и добавьте к нему графический интерфейс пользователя для формирования полноценного приложения FTP. Для этого можно выбрать любой из современных инструментариев создания графического интерфейса пользователя Python.
- 3.6. *Создание подклассов.* Создайте экземпляр ftplib.FTP и реализуйте новый класс FTP2, который не требует каждый раз выдавать команды STOR filename и RETR filename при работе с любым из четырех методов retr*() и stor*(); должно быть достаточно лишь передать имя файла. Вам предоставляется возможность переопределить существующие методы или создать новые с суффиксом 2, например retrlines2().

Файл Tools/scripts/ftpmirror.py в исходном дистрибутиве Python представляет собой сценарий, позволяющий создавать зеркальные отображения узлов FTP или их частей с использованием модуля ftplib. Это задание

может рассматриваться как расширенный пример применения указанного модуля. Следующие пять упражнений отличаются тем, что в них должны быть реализованы решения на основе кода, подобного коду приложения `ftpmirror.py`. Можно либо непосредственно использовать код сценария `ftpmirror.py`, либо реализовать собственное решение на основе использования этого кода в качестве образца.

- 3.7. *Рекурсия.* В сценарии `ftpmirror.py` предусмотрена возможность рекурсивно копировать каталоги на удаленном узле. Создайте более простой клиент FTP на основе сценария `ftpmirror.py`, но с тем условием, что рекурсия применяется лишь в случае необходимости. Предусмотрите возможность использовать опцию `-r`, которая служит для приложения указания, что каталоги и подкаталоги должны копироваться в локальную файловую систему рекурсивно.
- 3.8. *Сопоставление с шаблонами.* Для сценария `ftpmirror.py` предусмотрена опция `-s`, позволяющая пользователям пропускать файлы, которые согласуются с указанным шаблоном, таким как `.exe`. Создайте собственный более простой клиент FTP или обновите свое решение упражнения 3.7, чтобы дать возможность пользователю задавать шаблон и копировать только те файлы, которые согласуются с указанным шаблоном. Используйте в качестве отправной точки подготовленное вами решение одного из предыдущих упражнений.
- 3.9. *Рекурсия и сопоставление с шаблонами.* Создайте клиент FTP, в котором объединялись бы возможности, предусмотренные в упражнениях 3.7 и 3.8.
- 3.10. *Рекурсия и файлы ZIP.* Это упражнение подобно упражнению 3.7, но в нем вместо копирования файлов с удаленного узла в локальную файловую систему предлагается обеспечить либо загрузку удаленных файлов, либо сжатие их в архивы ZIP, TGZ или BZ2. Для этого обновите свой существующий клиент FTP или создайте новый. Опция `-z` позволяет пользователям автоматически создавать резервную копию узла FTP.
- 3.11. *Универсальное приложение.* Реализуйте единственное, окончательное, всеобъемлющее приложение FTP, в котором воплощались бы все решения упражнений 3.7–3.10, включая поддержку опций `-r`, `-s` и `-z`.

NNTP

- 3.12. *Вводные сведения о протоколе NNTP.* Внесите изменения в код сценария в упражнении 3.2 (`getLatestNNTP.py`), чтобы вместо последней по времени статьи отображалась первая доступная статья по интересующей теме.
- 3.13. *Усовершенствование кода.* Исправьте недостаток в сценарии `getLatestNNTP.py`, из-за которого в выводе отображаются строки, заключенные в тройные кавычки. Дело в том, что для нас желательно видеть в выводе строки интерактивного интерпретатора Python, а не текст в тройных кавычках. Для решения этой проблемы предусмотрите проверку того, представляет ли собой текст, который следует за знаками `>>>`, действительный код Python. В случае положительного ответа он должен отображаться как строка данных; в противном случае код в кавычках, о котором идет речь, не должен быть показан. **Дополнительное задание.** Используйте это решение для

поиска решения еще одной небольшой проблемы: ведущие пробелы не удаляются из текста, поскольку могут относиться к коду Python, обозначенному отступами. Если это действительно код, то он должен быть показан; в противном случае это — текст, поэтому к нему необходимо применить функцию `rstrip()` перед выводом.

- 3.14. *Поиск статей.* Создайте клиентское приложение NNTP, которое позволяет пользователям входить в систему и выбирать группу новостей, представляющую интерес. После этого для пользователя должно быть выведено приглашение, в котором можно указать ключевые слова для поиска в строках темы статей. Сформируйте список статей, которые соответствуют требованиям, и отобразите его для пользователя. После этого пользователю должна быть предоставлена возможность выбрать для прочтения статью из списка. Отобразите эту статью и предоставьте простейшие средства навигации, такие как разбивка на страницы и т.д. Если пользователь ничего не введет в строке поиска, включите в список все текущие статьи.
- 3.15. *Поиск в тексте.* Усовершенствуйте свое решение упражнения 3.14 так, чтобы поиск мог осуществляться и в строке темы, и в тексте статьи. Предусмотрите возможность применять операции AND и OR при поиске с помощью ключевых слов. Кроме того, предоставьте возможность с помощью операций AND и OR производить поиск в строках темы и (или) в текстах статей; иными словами, предусмотрите применение ключевых слов для поиска только в строках темы, только в текстах статей, в том или в другом, в том и в другом.
- 3.16. *Программа чтения новостей по потокам.* В данном случае речь идет не о создании многопоточной программы чтения групп новостей, а об организации поиска в поступлениях, которые увязаны в потоки статей. Иными словами, под потоками подразумеваются группы взаимосвязанных статей, в которых даты публикации отдельных статей играют второстепенную роль. Все статьи, принадлежащие к отдельным потокам, должны быть представлены в списке в хронологической последовательности. Предоставьте пользователю возможность делать следующее.
 - а) Выбирать для просмотра отдельные статьи (тексты), оставляя за собой возможность возвращаться к просмотру списка, переходить к предыдущей или следующей статье, последовательно или с учетом текущего потока.
 - б) Отправлять ответы в потоки, цитировать и копировать предыдущую статью, а также давать ответы, относящиеся ко всей группе новостей, формируя отдельное сообщение. **Дополнительное задание.** При этом участник обсуждения должен иметь возможность отвечать отдельным лицам по электронной почте.
 - в) Безвозвратно удалять потоки так, чтобы в списке статей этого потока больше не могли появляться относящиеся к нему статьи. Для этого необходимо постоянно вести список удаленных потоков, в котором в течение некоторого времени хранятся сведения об этих потоках для того, чтобы в данный период времени их нельзя было снова возобновить. Кроме того, должно быть учтено, что поток может прекратить свое существование, если в течение нескольких месяцев не появлялось ни одной статьи с относящейся к нему строкой темы.

- 3.17.** *Программа чтения групп новостей с графическим интерфейсом пользователя.* По аналогии с приведенным выше упражнением, касающимся FTP, создайте полностью отдельное приложение программы чтения групп новостей с графическим интерфейсом пользователя, выбрав для этого один из инструментов формирования графического интерфейса пользователя Python.
- 3.18.** *Применение рефакторинга.* По аналогии со сценарием `ftpmirror.py`, для FTP предусмотрен демонстрационный сценарий для NNTP: `Demo/scripts/newslst.py`. Вызовите его на выполнение. Этот сценарий написан очень давно и может потребовать модернизации. Настоящее упражнение состоит в том, что читателем должен быть проведен рефакторинг (перевод на другую основу) этой программы с использованием средств новейших версий Python, а также собственных наработок в рамках возможностей Python, что позволило бы выполнять те же задачи, но с меньшими затратами времени на эксплуатацию и выполнение. Для этого можно применить усовершенствованные конструкции списков или выражения генератора, осуществлять сцепление строк с помощью более мощных средств, исключить вызов ненужных функций и т.д.
- 3.19.** *Кеширование.* Еще один недостаток сценария `newslst.py` отмечен его автором: “Мне действительно приходится вести список игнорируемых пустых групп и повторно проверять такие группы на наличие статей при каждом прогоне, и я до сих пор не избавился от необходимости этим заниматься”. Постарайтесь провести это усовершенствование. В качестве точки отсчета можно взять указанную версию, применяемую по умолчанию, или воспользоваться усовершенствованной версией, полученной по результатам упражнения 3.18.

Электронная почта

- 3.20.** *Идентификаторы.* Метод `pass_()` протокола POP3 используется для отправки пароля на сервер после передачи регистрационного имени пользователя с помощью функции `login()`. По каким причинам, по вашему мнению, этот метод был обозначен именем, которое содержит заключительный знак подчеркивания (`pass_()`), когда можно было бы просто заменить старый метод `pass()`?
- 3.21.** *Протоколы POP и IMAP.* Напишите приложение для загрузки электронной почты с использованием одного из классов `poplib` (POP3 или POP3_SSL), затем разработайте аналогичное приложение с помощью `imaplib`. Для этого можно заимствовать часть кода, приведенного ранее в этой главе. Почему желательно исключить информацию об имени входа и пароле из исходного кода?
- Следующий ряд упражнений посвящен приложению `myMail.py`, представленному в упражнении 3.3.
- 3.22.** *Заголовки электронной почты.* В сценарии `myMail.py` последние несколько строк применяются для сравнения текста первоначально отправленного письма с текстом полученной электронной почты. Разработайте аналогичный код для сверки заголовков с отправленными ранее. **Подсказка.** Не рассматривайте вновь добавленные заголовки.

- 3.23.** *Проверка на наличие ошибок.* Добавьте средства проверки ошибок, основанные на использовании протоколов POP и SMTP.
- 3.24.** *Протоколы SMTP и IMAP.* Добавьте поддержку протокола IMAP. **Дополнительное задание.** Предусмотрите поддержку обоих протоколов загрузки почты и предоставьте пользователю возможность указать, какой из них он собирается использовать.
- 3.25.** *Составление письма для отправки по электронной почте.* Осуществите дополнительную доработку своего решения упражнения 3.24, предоставив пользователям приложения возможность составлять и отправлять электронную почту.
- 3.26.** *Приложение электронной почты.* Проведите дополнительную доработку своего приложения электронной почты путем введения средств управления почтовым ящиком, чтобы оно стало еще более удобным. Приложение должно предоставлять возможность считывать текущий набор сообщений электронной почты, указанных пользователем, и отображать для них строки темы. Пользователи должны иметь возможность выбирать сообщения для просмотра. **Дополнительное задание.** Предусмотрите поддержку просмотра вложений с помощью внешних приложений.
- 3.27.** *Применение графического интерфейса пользователя.* Дополните свое решение предыдущей задачи с применением средств графического интерфейса пользователя, чтобы получить фактически полноценное приложение электронной почты.
- 3.28.** *Первые шаги в борьбе со спамом.* В наши дни борьба с нежелательной электронной почтой, или спамом, превратилась в весьма актуальную и существенную проблему. Разработано много качественных решений этой проблемы, которые доказали свою применимость в данной области. Мы не собираемся требовать от читателя, чтобы он непременно повторил то, что уже сделано. В нашу задачу входит лишь предоставление возможности понять, в чем состоят некоторые приемы борьбы со спамом.

а) Формат mbox. Прежде чем приступить к решению этой задачи, необходимо обеспечить преобразование любых сообщений электронной почты, предназначенных для работы с ними, в единый формат, такой как формат mbox. (По желанию можно также воспользоваться другими форматами.) После преобразования нескольких (или всех) обрабатываемых сообщений в формат mbox их необходимо объединить в отдельный файл. **Подсказка.** См. описание модуля mailbox и пакета email.

б) Заголовки. Чаще всего признаком спама является определенное содержимое заголовков электронной почты. (Для начала можно попытаться воспользоваться пакетом для работы с электронной почтой или провести синтаксический анализ заголовков почты вручную.) Напишите код, который отвечает на приведенные ниже вопросы.

- Какой почтовый клиент применялся для первоначальной отправки этого сообщения? (Проверьте заголовок X-Mailer.)
- Является ли допустимым формат идентификатора сообщения (заголовок Message-ID)?
- Имеют ли место рассогласование доменных имен между заголовками

From, Received и, возможно, Return-Path? А как обстоят дела в отношении рассогласований доменного имени и IP-адреса? Имеется ли заголовок X-Authentication-Warning? Если да, то что он сообщает?

- в) *Информационные серверы.* Помощь в определении местонахождения, являющегося одним из источников массового формирования нежелательной электронной почты, могут оказать такие серверы, как WHOIS, SenderBase.org и т.д., для чего им необходимо предоставить IP-адрес или доменное имя. Найдите одну или несколько из этих служб и подготовьте код, позволяющий найти страну происхождения и, возможно, город, имя владельца сети, контактные данные и т.д.
- г) *Ключевые слова.* В спаме неизменно появляются определенные слова. С признаками спама, безусловно, встречались все пользователи электронной почты, в том числе со всеми их вариантами. К ним относится использование цифр, напоминающих буквы, преобразование в прописные случайно выбранных букв и т.д. Составьте список часто встречавшихся вам слов, которые определено связаны со спамом, и помещайте содержащие их письма в карантин. **Дополнительное задание.** Разработайте алгоритм или способ добавления вариантов ключевых слов, чтобы было проще вылавливать всевозможные сообщения, в которых обнаруживаются подобные признаки.
- д) *Фишинг.* Определенная категория спама, которая относится к фишингу, представляет собой попытку выдать поступающую электронную почту за отправленную крупными банковскими учреждениями или известными веб-сайтами в Интернете. Эти письма содержат ссылки, которые побуждают читателей отправляться на специально подготовленные веб-сайты и оставлять там свои приватные и чрезвычайно конфиденциальные данные, такие как регистрационные имена, пароли и номера кредитных карточек. За этими попытками стоят мошенники, которым удастся весьма успешно придать своим фиктивным сообщениям такой внешний вид, что почти невозможно отличить их от подлинных. Эти сообщения не могут скрыть тот факт, что фактическая ссылка, по которой они направляют пользователей, не принадлежит компании, под которую они маскируются. Чаще всего эти ссылки сразу же выдают себя; например, в них могут совершенно непривычно выглядеть доменные имена, применяться в непосредственном виде IP-адреса и даже IP-адреса в 32-разрядном целочисленном формате, а не в виде октетов. Разработайте код, который позволяет определить, является ли электронная почта, похожая на официальное сообщение, действительной или фиктивной.

Компоновка электронной почты

Следующий ряд упражнений касается составления сообщений электронной почты с использованием пакета электронной почты, в частности, относится к коду, который рассматривался в сценарии `email-examples.py`.

- 3.29. *Альтернативные варианты формирования многокомпонентных сообщений.* Прежде всего рассмотрим, что подразумевается под альтернативными вариантами формирования многокомпонентных сообщений. Эта тема кратко затрагивалась выше в данной главе при описании функции `make_mpa_msg()`,

а здесь будут приведены некоторые дополнительные сведения. Рассмотрим, как изменится поведение функции `make_mpa_msg()` после исключения параметра `'alternative'` при создании экземпляра класса `MIMEMultipart`, т.е. при использовании вызова `email = MIMEMultipart()`.

- 3.30. *Python 3.* Перенесите сценарий `email-examples.py` в версию Python 3 (или создайте гибридный вариант, который может применяться без модификации в обеих версиях, 2.x и 3.x).
- 3.31. *Несколько вложений.* В разделе, посвященном созданию электронной почты, рассматривалась функция `make_img_msg()`, с помощью которой создавалось одно сообщение электронной почты, содержащее отдельное изображение. Для начала освоения темы это вполне подходит, но на практике приходится решать более сложные задачи. Создайте функцию более общего назначения, которую пользователи могут применять для передачи нескольких файлов с изображениями, и присвойте ей имя `attachImgs()`, `attach_images()` или другое удобное имя. Задайте все необходимые файлы, затем преобразуйте их в отдельные вложения для сообщения электронной почты, имеющего общий текст, после чего возвратите единый многокомпонентный объект сообщения.
- 3.32. *Обеспечение надежности.* Внесите усовершенствования в решение упражнения 3.31, касающееся функции `attachImgs()`, обеспечив, чтобы пользователи могли передавать только файлы изображений (в противном случае необходимо активизировать исключения). Для этого необходимо предусмотреть проверку имен файлов, которая позволяла бы убедиться, что расширения этих файлов имеют вид `.png`, `.jpg`, `.gif`, `.tif` и т.д. **Дополнительное задание.** Предусмотрите применение средств анализа содержимого файла, которые позволяют проверять файлы с другим расширением, с неправильным расширением или без расширения и определять, к какому типу они действительно относятся. На первых порах может помочь страница Wikipedia, находящаяся по адресу http://en.wikipedia.org/wiki/File_format.
- 3.33. *Обеспечение надежности. Организация сетей.* Внесите дополнительные усовершенствования в функцию `attachImgs()`, чтобы пользователи могли не только указывать локальные файлы, но и передавать URL изображений, находящихся в сети, такие как http://docs.python.org/_static/py.png.
- 3.34. *Электронные таблицы.* Создайте функцию `attachSheets()`, которая позволяет вложить один или несколько файлов электронных таблиц в многокомпонентное сообщение электронной почты. Предусмотрите поддержку наиболее распространенных форматов, таких как `.csv`, `.xls`, `.xlsx`, `.ods`, `.uof`, `.uos` и т.д.). В качестве образца можно использовать функцию `attachImgs()`, но вместо метода `email.mime.image.MIMEImage` необходимо применить метод `email.mime.base.MIMEBase`, поскольку должен быть указан соответствующий тип MIME (например, `'application/vnd.ms-excel'`). Не забывайте также задать заголовок `Content-Disposition`.
- 3.35. *Документы.* По аналогии с упражнением 3.34 создайте функцию `attachDocs()`, которая позволяет подключать файлы документов к многокомпонентному сообщению электронной почты. Предусмотрите поддержку таких распространенных форматов, как `.doc`, `.docx`, `.odt`, `.rtf`, `.pdf`, `.txt`, `.uof`, `.uot` и т.д.

- 3.36. *Применение вложений нескольких типов.* Расширим область применения решений, полученных при выполнении упражнения 3.35. Создайте новую, более общую функцию `attachFiles()`, которая позволяет выбирать вложения любых типов. Для этого рекомендуется объединить весь код решений, который относится к каждому из рассматриваемых упражнений.

Разное

Список различных протоколов Интернета, включая три протокола, описанные более подробно в этой главе, можно найти по адресу <http://networksorcery.com/enp/topic/ipsuite.htm>. Список протоколов Интернета, поддерживаемых в языке Python, приведен по адресу <http://docs.python.org/library/internet>.

- 3.37. *Разработка альтернативных интернет-клиентов.* Теперь, после рассмотрения четырех примеров того, как с помощью языка Python можно разрабатывать интернет-клиенты, выберите еще один протокол с клиентской поддержкой в одном из библиотечных модулей стандартной библиотеки Python и напишите для него клиентское приложение.
- 3.38. **Разработка новых интернет-клиентов.* Намного более сложное упражнение: найдите менее распространенный или намеченный к созданию протокол без поддержки в языке Python и реализуйте его. Не сомневайтесь в том, что вы сможете написать и отправить на рассмотрение документ PEP, чтобы ваш модуль был включен в стандартный библиотечный дистрибутив будущей версии Python.

Многопоточное программирование

В этой главе...

- Введение/общее назначение
- Потоки и процессы
- Поддержка потоков в языке Python
- Модуль `thread`
- Модуль `threading`
- Сравнение однопоточного и многопоточного выполнения
- Практическое применение многопоточной обработки
- Проблема “производитель–потребитель” и модуль `Queue/queue`
- Дополнительные сведения об использовании потоков
- Связанные модули

- > С помощью Python можно запустить поток, но нельзя его остановить.
- > Вернее, приходится ожидать, пока он не достигнет конца выполнения.
- > Это означает, что все происходит, как в группе новостей [comp.lang.python]?
Обмен сообщениями между Клиффом Уэлсом (Cliff Wells)
и Стивом Холденом (Steve Holden) с участием Тимоти Делани
(Timothy Delaney), февраль 2002 г.

В настоящей главе рассматриваются различные способы обеспечения параллельного выполнения в коде. В первых нескольких разделах этой главы показано, в чем состоят различия между процессами и потоками. После этого будет дано определение понятия многопоточного программирования и представлены некоторые средства многопоточного программирования, предусмотренные в языке Python. (Читатели, уже знакомые с многопоточным программированием, могут перейти непосредственно к разделу 4.3.5.) В заключительных разделах этой главы приведены некоторые примеры того, как использовать модули `threading` и `Queue` для реализации многопоточного программирования с помощью языка Python.

4.1. Введение/общее назначение

До появления средств многопоточного (multithreaded — МТ) программирования выполнение компьютерных программ состояло из единой последовательности шагов, которые выполнялись процессором компьютера от начала до конца, т.е. *синхронно*. Такая организация выполнения применялась независимо от того, требовала ли сама задача последовательного упорядочения шагов или допускала разбиение на подзадачи и отдельное их выполнение в программе. В последнем случае подзадачи вообще могли быть независимыми, не связанными никакими причинно-следственными отношениями (а это означает, что результаты одних подзадач не влияют на выполнение других подзадач). Из этого следует вывод, что такие независимые задачи могут выполняться не последовательно, а одновременно. Подобная параллельная организация работы позволяет существенно повысить эффективность решения всей задачи. Изложенные выше соображения лежат в основе многопоточного программирования.

Многопоточное программирование идеально подходит для решения задач, *асинхронных* по своему характеру (т.е. допускающих прерывание работы), требующих выполнения нескольких параллельных действий, в которых реализация каждого действия может быть *недетерминированной*, иными словами, происходящей в случайные и непредсказуемые моменты времени. Такие задачи программирования могут быть организованы в виде нескольких потоков выполнения или разделены на несколько потоков, в каждом из которых осуществляется конкретная подзадача. В зависимости от приложения в этих подзадачах могут вычисляться промежуточные результаты для последующего слияния и формирования заключительной части вывода.

Задачи, выполнение которых ограничено пропускной способностью процессора, довольно легко разделить на подзадачи, выполняемые в последовательном или многопоточном режиме. С другой стороны, организовать выполнение однопоточного процесса с несколькими внешними источниками ввода не столь просто. Для решения этой задачи без применения многопоточного режима в последовательной программе необходимо предусмотреть один или несколько таймеров и реализовать схему мультиплексирования.

В последовательной программе потребуется опрашивать каждый терминальный канал ввода-вывода для проверки наличия данных, введенных пользователем. При этом необходимо добиться того, чтобы в программе не происходило блокирование при чтении из терминального канала ввода-вывода, поскольку сам характер поступления введенных пользователем данных является недетерминированным, а блокировка привела бы к нарушению обработки данных из других каналов ввода-вывода. В такой последовательной программе приходится использовать неблокирующий ввод-вывод или блокирующий ввод-вывод с таймером (чтобы блокировка устанавливалась лишь во время).

Последовательная программа представляет собой единый поток выполнения, поэтому ей приходится манипулировать отдельными подзадачами, чтобы на любую отдельно взятую подзадачу не затрачивалось слишком много времени, а также следить за тем, чтобы длительность формирования ответов пользователям соответствовала установленным критериям. Применение последовательной программы для решения задачи такого типа часто требует организации сложной системы передачи управления, которую трудно понять и сопровождать.

Если же для решения подобной задачи программирования применяется многопоточная программа с общей структурой данных, такой как `Queue` (многопоточная структура данных очереди, рассматриваемая ниже в этой главе), то весь ход работы можно организовать с помощью нескольких потоков, каждый из которых выполняет конкретные функции, например, как показано ниже.

- `UserRequestThread`. Обеспечивает чтение данных, введенных пользователем, возможно, из канала ввода-вывода. В программе может быть создан целый ряд потоков, по одному для каждого из одновременно работающих клиентов, запросы которых могут быть поставлены в очередь.
- `RequestProcessor`. Поток, который отвечает за выборку запросов из очереди и их обработку с предоставлением полученных выходных данных для еще одного потока.
- `ReplyThread`. Поток, обеспечивающий получение выходных данных, предназначенных для пользователя, и их отправку в ответ на запрос (если приложение является сетевым) или запись данных в локальной файловой системе или базе данных.

Если для решения подобной задачи программирования применяется несколько потоков, то сложность программы сокращается и появляется возможность обеспечить простую, эффективную и хорошо организованную реализацию. Программная реализация каждого потока, как правило, становится проще, поскольку поток предназначен для выполнения не всего задания, а лишь его части. Например, поток `UserRequestThread` просто считывает данные, введенные пользователем, и помещает их в очередь для дальнейшей обработки другим потоком и т.д. Каждый поток решает собственную подзадачу; программисту остается лишь тщательно спроектировать потоки каждого из применяемых типов, чтобы они выполняли то, что от них требуется, наилучшим образом. Принцип использования потоков для решения конкретных задач мало чем отличается от предложенной Генри Фордом модели сборочной линии для производства автомобилей.

4.2. Потоки и процессы

4.2.1. Общее определение понятия процесса

Компьютерные программы — это просто исполняемые объекты в двоичной (или другой) форме, которые находятся на диске. Программы начинают действовать лишь после их загрузки в память и вызова операционной системой. Процесс — это программа в ходе ее выполнения (в такой форме процессы иногда называют *тяжеловесными процессами*). Каждый процесс имеет собственное адресное пространство, память и стек данных, а также может использовать другие вспомогательные данные для контроля над выполнением. Операционная система управляет выполнением всех процессов в системе, выделяя каждому процессу процессорное время по определенному принципу. В ходе выполнения процесса может также происходить ветвление или порождение новых процессов для осуществления других задач, но каждый новый процесс имеет собственную память, стек данных и т.д. Вообще говоря, отдельные процессы не могут иметь доступ к общей информации, если не реализовано *межпроцессное взаимодействие* (interprocess communication — IPC) в той или иной форме.

4.2.2. Общее определение понятия потока

Потоки (иногда называемые *легковесными процессами*) подобны процессам, за исключением того, что все они выполняются в пределах одного и того же процесса, следовательно, используют один и тот же контекст. Потоки можно рассматривать как своего рода “мини-процессы”, работающие параллельно в рамках основного процесса или основного потока.

Поток запускается, проходит определенную последовательность выполнения и завершается. В потоке ведется указатель команд, позволяющий следить за тем, где в настоящее время происходит его выполнение в текущем контексте. Поток может быть прерван и переведен на время в состояние ожидания (это состояние принято также называть *приостановкой* (sleeping)), в то время как другие потоки продолжают работать. Такая операция называется *возвратом управления* (yielding).

Все потоки, организованные в одном процессе, используют общее пространство данных с основным потоком, поэтому могут обмениваться информацией или взаимодействовать друг с другом с меньшими сложностями по сравнению с отдельными процессами. Потоки, как правило, выполняются параллельно. Именно распараллеливание и совместное использование данных становятся предпосылками обеспечения координации выполнения нескольких задач. Вполне естественно, что в системе с одним процессором невозможно в полном смысле слова организовать параллельное выполнение, поэтому планирование потоков происходит таким образом, чтобы каждый из них выполнялся в течение какого-то короткого промежутка времени, а затем возвращал управление другим потокам (образно говоря, снова становился в очередь на получение следующей порции процессорного времени). В ходе выполнения всего процесса каждый поток осуществляет свои собственные, отдельные задачи и передает полученные результаты другим потокам по мере необходимости.

Разумеется, переход от последовательной организации работы к параллельной связан с возникновением дополнительных сложностей. В частности, если два или несколько потоков получают доступ к одному и тому же фрагменту данных, то в зависимости от того, в какой последовательности происходит доступ, могут возникать несогласованные результаты. Неопределенность в отношении последовательности

доступа принято называть *состоянием состязания* (race condition). К счастью, в большинстве библиотек поддержки потоков предусмотрены примитивы синхронизации того или иного типа, которые позволяют диспетчеру потоков управлять выполнением и доступом.

Еще одна сложность обусловлена тем, что невозможно предоставлять всем потокам равную и справедливую долю времени выполнения. Это связано с тем, что некоторые функции устанавливают блокировки и снимают их только после завершения своего выполнения. Если функция не разработана специально для использования в потоке, то ее применение может привести к перераспределению процессорного времени в ее пользу. Такие функции принято называть *жадными* (greedy).

4.3. Поддержка потоков в языке Python

В разделе описывается использование потоков в программе на языке Python. В частности, рассматриваются ограничения потоков, обусловленные применением глобальной блокировки интерпретатора, и приводится небольшой демонстрационный сценарий.

4.3.1. Глобальная блокировка интерпретатора

Выполнением кода Python управляет *виртуальная машина Python* (называемая также *главным циклом интерпретатора*). Язык Python разработан таким способом, чтобы в этом главном цикле мог выполняться только один поток управления по аналогии с тем, как организовано совместное использование одного процессора несколькими процессами в системе. В памяти может находиться много программ, но в любой конкретный момент времени процессор занимает только одна из них. Аналогичным образом, притом что в интерпретаторе Python могут эксплуатироваться несколько потоков, в любой момент времени интерпретатором выполняется только один поток.

Для управления доступом к виртуальной машине Python применяется *глобальная блокировка интерпретатора* (global interpreter lock — GIL). Именно эта блокировка обеспечивает то, что выполняется один и только один поток. Виртуальная машина Python функционирует в многопоточной среде следующим образом.

1. Задание глобальной блокировки интерпретатора.
2. Переключение на поток для его выполнения.
3. Должно быть выполнено одно из следующего:
 - а) заданное количество команд в байт-коде;
 - б) проверка способности потока самостоятельно возвращать управление (для чего может служить вызов функции `time.sleep(0)`).
4. Перевести поток назад в приостановленное состояние (выйти из потока).
5. Разблокировать глобальную блокировку интерпретатора.
6. Снова проделать все эти действия (lather, rinse, repeat).

Если сделан вызов внешнего кода (допустим, любой встроенной функции расширения C/C++), то глобальная блокировка интерпретатора будет заблокирована до завершения этого вызова (поскольку в языке Python невозможно задать интервал с помощью байт-кода). Тем не менее при программировании расширений не исключена

возможность разблокирования глобальной блокировки интерпретатора, что позволяет избавить разработчика Python от необходимости брать на себя управление блокировками в коде Python в подобных ситуациях.

Например, в любых процедурах Python, основанных на использовании ввода-вывода (в которых вызывается встроенный код С операционной системы), предусмотрено освобождение глобальной блокировки интерпретатора до вызова функции ввода-вывода, что позволяет продолжить выполнение других потоков, в то время как происходит ввод-вывод. Если же в коде не осуществляется большой объем ввода-вывода, то, как правило, процессор (и глобальная блокировка интерпретатора) блокируется на полный интервал времени, предоставленный потоку, пока он не вернет управление. Иными словами, больше шансов воспользоваться преимуществами многопоточной среды имеют программы Python, ограничиваемые пропускной способностью ввода-вывода, чем программы, ограничиваемые пропускной способностью процессора.

Читатели, желающие ознакомиться с исходным кодом, а также изучить организацию главного цикла интерпретатора и глобальной блокировки интерпретатора, могут просмотреть файл `Python/ceval.c`.

4.3.2. Выход из потока

После того как поток завершает выполнение задачи, для которой он был создан, происходит выход из потока. Выход из потока может осуществляться путем вызова одной из функций выхода, такой как `thread.exit()`, с применением любого из стандартных способов выхода из процесса Python, например `sys.exit()`, или с помощью генерирования исключения `SystemExit`. Однако возможность непосредственно уничтожить поток отсутствует.

В следующем разделе будут рассматриваться два модуля Python, применяемых для работы с потоками, но один из них, модуль `thread`, не рекомендуется для использования. Для этого есть много причин, но наиболее важной из них является то, что применение этого модуля приводит к завершению работы всех прочих потоков после выхода из основного потока, и при этом очистка памяти не осуществляется должным образом. Второй модуль, `threading`, гарантирует, что весь процесс будет оставаться действующим до тех пор, пока не произойдет выход из всех важных дочерних потоков. (Чтобы ознакомиться со сведениями о том, почему это так важно, прочитайте врезку “Избегайте использования модуля `thread`”.)

Тем не менее в основные потоки всегда следует закладывать такие алгоритмы, чтобы они качественно выполняли функции диспетчера и при осуществлении этой задачи учитывали, какое назначение имеют отдельные потоки, какие данные или параметры требуются для каждого из порожденных потоков, когда эти потоки завершат выполнение и какие результаты предоставят. В ходе выполнения этой работы основные потоки могут дополнительно формировать отдельные результаты в виде окончательного, значимого вывода.

4.3.3. Доступ к потокам из программы Python

Язык Python поддерживает многопоточное программирование с учетом особенностей операционной системы, под управлением которой он функционирует. Он поддерживается на большинстве платформ на основе Unix, таких как Linux, Solaris, Mac OS X, *BSD, а также на персональных компьютерах под управлением Windows. В языке Python используются потоки, совместимые со стандартом POSIX, которые иногда называют *пи-потоками* (pthreads).

По умолчанию поддержка потоков включается при построении интерпретатора Python из исходного кода (начиная с версии Python 2.0) или при установке исполняемой программы интерпретатора в среде Win32. Чтобы определить, предусмотрено ли применение потоков на конкретном установленном интерпретаторе, достаточно просто попытаться импортировать модуль `thread` из интерактивного интерпретатора, как показано ниже (если потоки доступны, то не появится сообщение об ошибке).

```
>>> import thread
>>>
```

Если интерпретатор Python не был откомпилирован с включенными потоками, то попытка импорта модуля оканчивается неудачей:

```
>>> import thread
Traceback (innermost last):
  File "<stdin>", line 1, in ?
ImportError: No module named thread
```

В таких случаях может потребоваться повторно откомпилировать интерпретатор Python, чтобы получить доступ к потокам. Для этого обычно достаточно вызвать сценарий `configure` с опцией `--with-thread`. Прочитайте файл `README` для применяемого вами дистрибутива, чтобы ознакомиться с инструкциями, касающимися того, как откомпилировать исполняемую программу интерпретатора Python с поддержкой потоков для своей системы.

4.3.4. Организация программы без применения потоков

В первом ряде примеров для демонстрации работы потоков воспользуемся функцией `time.sleep()`. Функция `time.sleep()` принимает параметр в формате с плавающей запятой и приостанавливается ("засыпает") на указанное количество секунд; иными словами, выполнение программы временно прекращается на заданное время.

Создадим два цикла во времени: приостанавливающийся на 4 секунды (функция `loop0()`) и на 2 секунды (функция `loop1()`) соответственно. (В данной программе имена `loop0` и `loop1` используются в качестве указания на то, что в конечном итоге будет создана последовательность циклов.) Если бы задача состояла в том, чтобы функции `loop0()` и `loop1()` выполнялись последовательно в однопроцессной или однопоточной программе, по аналогии со сценарием `onethr.py` в примере 4.1, то общее время выполнения составляло бы по меньшей мере 6 секунд. Между завершением работы `loop0()` и запуском `loop1()` может быть предусмотрен промежуток в 1 секунду, кроме того, в ходе выполнения могут возникнуть другие задержки, поэтому общая продолжительность работы программы может достичь 7 секунд.

Пример 4.1. Выполнение циклов в одном потоке (`onethr.py`)

В этом сценарии два цикла выполняются последовательно в однопоточной программе. Вначале должен быть завершен один цикл, чтобы мог начаться другой. Общее истекшее время представляет собой сумму значений времени, затраченных в каждом цикле.

```
1  #!/usr/bin/env python
2
3  from time import sleep, ctime
4
```

```

5  def loop0():
6      print 'start loop 0 at:', ctime()
7      sleep(4)
8      print 'loop 0 done at:', ctime()
9
10 def loop1():
11     print 'start loop 1 at:', ctime()
12     sleep(2)
13     print 'loop 1 done at:', ctime()
14
15 def main():
16     print 'starting at:', ctime()
17     loop0()
18     loop1()
19     print 'all DONE at:', ctime()
20
21 if __name__ == '__main__':
22     main()

```

В этом можно убедиться, выполнив сценарий `onethr.py` и ознакомившись со следующим выводом:

```

$ onethr.py
starting at: Sun Aug 13 05:03:34 2006
start loop 0 at: Sun Aug 13 05:03:34 2006
loop 0 done at: Sun Aug 13 05:03:38 2006
start loop 1 at: Sun Aug 13 05:03:38 2006
loop 1 done at: Sun Aug 13 05:03:40 2006
all DONE at: Sun Aug 13 05:03:40 2006

```

Теперь предположим, что работа функций `loop0()` и `loop1()` не организована по принципу приостановки, а предусматривает выполнение отдельных и независимых вычислений, предназначенных для выработки общего решения. При этом не исключена такая возможность, что выполнение этих функций можно осуществлять параллельно в целях сокращения общей продолжительности работы программы. В этом состоит идея, лежащая в основе многопоточного программирования, к рассмотрению которого мы теперь приступим.

4.3.5. Многопоточные модули Python

В языке Python предусмотрено несколько модулей, позволяющих упростить задачу многопоточного программирования, включая модули `thread`, `threading` и `Queue`. Для создания потоков и управления ими программисты могут использовать модули `thread` и `threading`. В модуле `thread` предусмотрены простые средства управления потоками и блокировками, а модуль `threading` обеспечивает высокоуровневое, полноценное управление потоками. С помощью модуля `Queue` пользователи могут создать структуру данных очереди, совместно используемую несколькими потоками. Рассмотрим эти модули отдельно и представим примеры и более крупные приложения.



Избегайте использования модуля `thread`

Мы рекомендуем использовать высокоуровневый модуль `threading` вместо модуля `thread` по многим причинам. Модуль `threading` имеет более широкий набор

функций по сравнению с модулем `thread`, обеспечивает лучшую поддержку потоков, и в нем исключены некоторые конфликты атрибутов, обнаруживаемые в модуле `thread`. Еще одна причина отказаться от использования модуля `thread` состоит в том, что `thread` — это модуль более низкого уровня и имеет мало примитивов синхронизации (фактически только один), в то время как модуль `threading` обеспечивает более широкую поддержку синхронизации.

Тем не менее мы представим некоторые примеры кода, в которых используется модуль `thread`, поскольку это будет способствовать изучению языка Python и многопоточной организации программ в целом. Но эти примеры представлены исключительно в учебных целях, в надежде на то, что они позволят гораздо лучше понять обоснованность рекомендации, касающейся отказа от использования модуля `thread`. Мы также покажем, как использовать более удобные инструменты, предусмотренные в модулях `Queue` и `threading`.

Еще одна причина отказа от работы с модулем `thread` состоит в том, что этот модуль не позволяет взять под свое управление выход из процесса. После завершения основного потока происходит также уничтожение всех прочих потоков без предупреждения или надлежащей очистки памяти. Как было указано выше, модуль `threading` позволяет по меньшей мере дожидаться завершения работы важных дочерних потоков и только после этого выйти из программы.

3.x

Использование модуля `thread` рекомендуется только для экспертов, которым требуется получить доступ к потоку на более низком уровне. Для того чтобы эта особенность модуля стала более очевидной, в Python 3 он был переименован в `_thread`. В любом создаваемом многопоточном приложении следует использовать `threading`, а также, возможно, другие высокоуровневые модули.

4.4. Модуль thread

Вначале рассмотрим, какие задачи возлагались на модуль `thread`. От модуля `thread` требовалось не только порождать потоки, но и обеспечивать работу с основной структурой синхронизации данных, называемой *объектом блокировки* (такowymi являются примитивная блокировка, простая блокировка, блокировка со взаимным исключением, мьютекс и двоичный семафор). Как было указано выше, без подобных примитивов синхронизации сложно обойтись при управлении потоками.

В табл. 4.1 приведен список наиболее широко используемых функций потока и методов объекта блокировки `LockType`.

Таблица 4.1. Модуль `thread` и объекты блокировки

Функция/метод	Описание
Функции модуля <code>thread</code>	
<code>start_new_thread(function, args, kwargs=None)</code>	Порождает новый поток и вызывает на выполнение функцию <code>function</code> с заданными параметрами <code>args</code> и необязательными параметрами <code>kwargs</code>
<code>allocate_lock()</code>	Распределяет объект блокировки <code>LockType</code>
<code>exit()</code>	Дает указание о выходе из потока
Методы объекта <code>LockType</code> <code>Lock</code>	
<code>acquire(wait=None)</code>	Предпринимает попытки захватить объект блокировки

Функция/метод	Описание
<code>locked()</code>	Возвращает <code>True</code> , если блокировка захвачена, в противном случае возвращает <code>False</code>
<code>release()</code>	Освобождает блокировку

Ключевой функцией модуля `thread` является `start_new_thread()`. Эта функция получает предназначенную для вызова функцию (объект) с позиционными параметрами и (необязательно) с ключевыми параметрами. Специально для вызова функции создается новый поток.

Возвратимся к примеру `onethr.py`, чтобы встроить в него многопоточную поддержку. В примере 4.2 представлен сценарий `mtsleepA.py`, в котором внесены некоторые изменения в функции `loop*()`:

Пример 4.2. Использование модуля `thread` (`mtsleepA.py`)

Выполняются те же циклы, что и в сценарии `onethr.py`, но на этот раз с использованием простого многопоточного механизма, предоставленного модулем `thread`. Эти два цикла выполняются одновременно (разумеется, не считая того, что менее продолжительный цикл завершается раньше), поэтому общие затраты времени определяются продолжительностью работы самого длительного потока, а не представляют собой сумму значений времени выполнения отдельно каждого цикла.

```

1  #!/usr/bin/env python
2
3  import thread
4  from time import sleep, ctime
5
6  def loop0():
7      print 'start loop 0 at:', ctime()
8      sleep(4)
9      print 'loop 0 done at:', ctime()
10
11  def loop1():
12      print 'start loop 1 at:', ctime()
13      sleep(2)
14      print 'loop 1 done at:', ctime()
15
16  def main():
17      print 'starting at:', ctime()
18      thread.start_new_thread(loop0, ())
19      thread.start_new_thread(loop1, ())
20      sleep(6)
21      print 'all DONE at:', ctime()
22
23  if __name__ == '__main__':
24      main()

```

Для функции `start_new_thread()` должны быть представлены по крайней мере первые два параметра, поэтому при ее вызове задан пустой кортеж, несмотря на то, что вызываемая на выполнение функция не требует параметров.

Выполнение этой программы показывает, что данные на выходе существенно изменились. Вместо полных затрат времени, составлявших 6 или 7 секунд, новый сценарий завершается в течение 4 секунд, что представляет собой продолжительность самого длинного цикла с добавлением небольших издержек.

```
$ mtsleepA.py
starting at: Sun Aug 13 05:04:50 2006
start loop 0 at: Sun Aug 13 05:04:50 2006
start loop 1 at: Sun Aug 13 05:04:50 2006
loop 1 done at: Sun Aug 13 05:04:52 2006
loop 0 done at: Sun Aug 13 05:04:54 2006
all DONE at: Sun Aug 13 05:04:56 2006
```

Фрагменты кода, в которых происходит приостановка на 4 с и на 2 секунды, теперь начинают выполняться одновременно, внося свой вклад в отсчет минимального значения полного времени прогона. Можно даже наблюдать за тем, как цикл 1 завершается перед циклом 0.

Еще одним важным изменением в приложении является добавление вызова `sleep(6)`. С чем связана необходимость такого добавления? Причина этого состоит в том, что если не будет установлен запрет на продолжение основного потока, то в нем произойдет переход к следующей инструкции, появится сообщение “all done” (работа закончена) и работа программы завершится после уничтожения обоих потоков, в которых выполняются функции `loop0()` и `loop1()`.

В сценарии отсутствует какой-либо код, который бы указывал основному потоку, что следует ожидать завершения дочерних потоков, прежде чем продолжить выполнение инструкций. Это — одна из ситуаций, которая показывает, что подразумевается под утверждением, согласно которому для потоков требуется определенная синхронизация. В данном случае в качестве механизма синхронизации применяется еще один вызов `sleep()`. При этом используется значение продолжительности приостановки, равное 6 секундам, поскольку известно, что оба потока (которые занимают 4 и 2 секунды) должны были завершиться до того, как в основном потоке будет отсчитан интервал времени 6 секунд.

Напрашивается вывод, что должен быть какой-то более удобный способ управления потоками по сравнению с созданием дополнительной задержки в 6 секунд в основном потоке. Дело в том, что из-за этой задержки общее время прогона ненамного лучше по сравнению с однопоточной версией. К тому же применение функции `sleep()` для синхронизации потоков, как в данном примере, не позволяет обеспечить полную надежность. Например, может оказаться, что синхронизируемые потоки являются независимыми друг от друга, а значения времени их выполнения изменяются. В таком случае выход из основного потока может произойти слишком рано или слишком поздно. Как оказалось, гораздо лучшим способом синхронизации является применение блокировок.

В примере 4.3 показан сценарий `mtsleepB.py`, полученный в результате следующего обновления кода, в котором добавляются блокировки и исключается дополнительная функция установки задержки. Выполнение этого сценария показывает, что полученный вывод аналогичен выводу сценария `mtsleepA.py`. Единственное различие состоит в том, что не пришлось устанавливать дополнительное время ожидания завершения работы, как в сценарии `mtsleepA.py`. С использованием блокировок мы получили возможность выйти из программы сразу после того, как оба потока завершили выполнение. При этом был получен следующий вывод:

```
$ mtsleepB.py
starting at: Sun Aug 13 16:34:41 2006
start loop 0 at: Sun Aug 13 16:34:41 2006
start loop 1 at: Sun Aug 13 16:34:41 2006
loop 1 done at: Sun Aug 13 16:34:43 2006
loop 0 done at: Sun Aug 13 16:34:45 2006
all DONE at: Sun Aug 13 16:34:45 2006
```

Пример 4.3. Использование блокировок и модуля `thread` (mtsleepB.py)

Очевидно, что гораздо удобнее использовать блокировки по сравнению с вызовом `sleep()` для задержки выполнения основного потока, как в сценарии `mtsleepA.py`.

```
1  #!/usr/bin/env python
2
3  import thread
4  from time import sleep, ctime
5
6  loops = [4,2]
7
8  def loop(nloop, nsec, lock):
9      print 'start loop', nloop, 'at:', ctime()
10     sleep(nsec)
11     print 'loop', nloop, 'done at:', ctime()
12     lock.release()
13
14  def main():
15     print 'starting at:', ctime()
16     locks = []
17     nloops = range(len(loops))
18
19     for i in nloops:
20         lock = thread.allocate_lock()
21         lock.acquire()
22         locks.append(lock)
23
24     for i in nloops:
25         thread.start_new_thread(loop,
26                                 (i, loops[i], locks[i]))
27
28     for i in nloops:
29         while locks[i].locked(): pass
30
31     print 'all DONE at:', ctime()
32
33  if __name__ == '__main__':
34     main()
```

Рассмотрим, как в данном случае организовано применение блокировок. Обратимся к исходному коду.

Построчное объяснение

Строки 1–6

После начальной строки Unix располагаются инструкции импорта модуля `thread` и задания нескольких знакомых атрибутов модуля `time`. Вместо жесткого задания в коде отдельных функций для отсчета 4 и 2 секунд используется единственная функция `loop()`, а константы, применяемые в ее вызове, задаются в списке `loops`.

Строки 8–12

Эта функция `loop()` действует в качестве замены исключенных из кода функций `loop*()`, которые были предусмотрены в предыдущих примерах. В функцию `loop()` пришлось внести несколько небольших изменений и дополнений, чтобы обеспечить выполнение этой функцией своего назначения с помощью блокировок. Одно из очевидных изменений состоит в том, что циклы обозначены номерами, с которыми связана продолжительность приостановки. Еще одним дополнением стало применение самой блокировки. Для каждого потока распределяется и захватывается блокировка. По истечении времени, установленного функцией `sleep()`, соответствующая блокировка освобождается, тем самым основному потоку передается указание, что данный дочерний поток завершен.

Строки 14–34

В этом сценарии значительная часть работы выполняется в функции `main()`, для чего применяются три отдельных цикла `for`. Вначале создается список блокировок, для получения которых используется функция `thread.allocate_lock()`, затем происходит захват каждой блокировки (отдельно) с помощью метода `acquire()`. Захват блокировки приводит к тому, что блокировка становится недоступной для дальнейшего манипулирования ею. После того как блокировка становится заблокированной, она добавляется к списку блокировок `locks`. Операция порождения потоков осуществляется в следующем цикле, после чего для каждого потока вызывается функция `loop()`, потоку присваивается номер цикла, задается продолжительность приостановки, и, наконец, для этого потока захватывается блокировка. Почему в данном случае не происходит запуск потоков в цикле захвата блокировок? На это есть две причины. Во-первых, необходимо обеспечить синхронизацию потоков, чтобы все наши лошади выскочили из ворот на беговую дорожку примерно в одно и то же время, и, во-вторых, приходится учитывать, что захват блокировок связано с определенными затратами времени. Если поток выполняет свою задачу слишком быстро, то может завершиться еще до того, как появится шанс захватить блокировку.

Задача разблокирования своего объекта блокировки после завершения выполнения возлагается на сам поток. В последнем цикле осуществляются лишь ожидание и возврат в начало (тем самым обеспечивается приостановка основного потока), и это происходит до тех пор, пока не будут освобождены обе блокировки. Затем выполнение продолжается. Проверка каждой блокировки происходит последовательно, поэтому может оказаться, что при размещении всех продолжительных циклов ближе к началу списка циклов весь ход программы будет определяться задержками в медленных циклах. В таких случаях основная часть времени ожидания будет затрачиваться в первом (или первых) цикле. К моменту освобождения этой блокировки все остальные блокировки могут уже быть разблокированы (иными словами, соответствующие потоки могут быть завершены). В результате в основном потоке выполнение

операций проверки остальных, освобожденных блокировок произойдет мгновенно, без пауз. Наконец, необходимо полностью гарантировать, чтобы в заключительной паре строк выполнение функции `main()` происходило лишь при непосредственном вызове этого сценария.

Как было указано в предыдущем основном примечании, в данной главе модуль `thread` представлен исключительно для того, чтобы ознакомить читателя с начальными сведениями о многопоточном программировании. В многопоточном приложении, как правило, следует использовать высокоуровневые модули, такие как модуль `threading`, который рассматривается в следующем разделе.

4.5. Модуль `threading`

Перейдем к описанию высокоуровневого модуля `threading`, который не только предоставляет класс `Thread`, но и дает возможность воспользоваться широким разнообразием механизмов синхронизации, позволяющих успешно решать многие важные задачи. В табл. 4.2 представлен список всех объектов, имеющих в модуле `threading`.

Таблица 4.2. Объекты модуля `threading`

Объект	Описание
<code>Thread</code>	Объект, который представляет отдельный поток выполнения
<code>Lock</code>	Примитивный объект блокировки (такая же блокировка, как и в модуле <code>thread</code>)
<code>Rlock</code>	Реентерабельный объект блокировки предоставляет возможность в отдельном потоке (повторно) захватывать уже захваченную блокировку (это рекурсивная блокировка)
<code>Condition</code>	Объект условной переменной вынуждает один поток ожидать, пока определенное условие не будет выполнено другим потоком. Таким условием может быть изменение состояния или задание определенного значения данных
<code>Event</code>	Обобщенная версия условных переменных, которая позволяет обеспечить ожидание некоторого события любым количеством потоков, так что после обнаружения этого события происходит активизация всех потоков
<code>Semaphore</code>	Предоставляет счетчик конечных ресурсов, совместно используемый потоками; если ни один из ресурсов не доступен, происходит блокировка
<code>BoundedSemaphore</code>	Аналогично <code>Semaphore</code> , но гарантируется, что превышение начального значения никогда не произойдет
<code>Timer</code>	Аналогично <code>Thread</code> , за исключением того, что происходит ожидание в течение выделенного промежутка времени перед выполнением
<code>Barrier^a</code>	Создает барьер, которого должно достичь определенное количество потоков, прежде чем всем этим потокам будет разрешено продолжить работу

3.2 ^a Новое в Python 3.2.

В этом разделе описано, как использовать класс `Thread` для реализации многопоточного режима работы. Выше в данной главе уже были приведены основные сведения о блокировках, поэтому в данном разделе не будут рассматриваться примитивы

блокировки. Кроме того, сам класс `Thread()` позволяет решать некоторые задачи синхронизации, поэтому нет необходимости явно использовать примитивы блокировки.



Потоки, функционирующие в качестве демонов

Еще одна причина, по которой следует избегать использования модуля `thread`, состоит в том, что он не поддерживает принцип организации работы программы на основе демонов (потоков, работающих в фоновом режиме). Модуль `thread` действует так, что после выхода из основного потока все дочерние потоки уничтожаются, без учета того, должны ли они продолжить выполнение определенной работы. Если это нежелательно, то можно организовать функционирование потоков в качестве демонов.

Поддержка демонов предусмотрена в модуле `threading`. Ниже описано, как они функционируют. Обычно демон применяется в качестве сервера, который ожидает поступления клиентских запросов, подлежащих выполнению. Если нет никакой работы, поступившей от клиентов, которая должна быть сделана, то демон простаивает. Для потока может быть установлен флаг, указывающий, что этот поток может выполнять роль демона. Такое указание равносильно обозначению родительского потока как не требующего после своего завершения, чтобы был завершен дочерний поток. Как было описано в главе 2, потоки сервера функционируют в виде бесконечных циклов и в обычных ситуациях не завершают свою работу.

Дочерние потоки могут быть также обозначены флагами как демоны, если в основном потоке могут складываться условия готовности к выходу, но нет необходимости ожидать завершения работы дочерних потоков, чтобы выйти из основной программы. Значение `true` указывает, что дочерний поток не приносит результатов, от которых зависит возможность завершения всей программы, и в основном рассматривается как указание, что единственным назначением потока является ожидание запросов от клиентов и их обслуживание.

Чтобы обозначить поток как выполняющий функции демона, необходимо применить оператор присваивания `thread.daemon = True`, прежде чем запустить этот поток. (Устаревший способ осуществления этой задачи, заключавшийся в вызове оператора `thread.setDaemon(True)`, теперь запрещен.) То же является справедливым в отношении проверки того, выполняет ли поток функции демона; достаточно проверить значение соответствующей переменной (а не вызывать функцию `thread.isDaemon()`). Новый дочерний поток наследует свой флаг, обозначающий его в качестве демона, от родительского потока. Вся программа Python (рассматриваемая как основной поток) продолжает функционировать до тех пор, пока не произойдет выход из всех потоков, не обозначенных как демоны, иными словами, до тех пор, пока остаются активными какие-либо потоки, не действующие как демоны.

4.5.1. Класс `Thread`

В программе с многопоточной организацией главным инструментом является класс `Thread` модуля `threading`. Модуль `threading` поддерживает целый ряд функций, отсутствующих в модуле `thread`. В табл. 4.3 представлен список атрибутов и методов модуля `threading`.

Таблица 4.3. Атрибуты и методы объекта класса Thread

Атрибут	Описание
Атрибуты данных объекта потока	
<code>name</code>	Имя потока
<code>Ident</code>	Идентификатор потока
<code>Daemon</code>	Булев флаг, указывающий, выполняет ли поток функции демона
Методы объекта потока	
<code>__init__(group=None, target=None, name=None, args=(), kwargs={}, verbose=None, daemon=None)^c</code>	Порождение объекта Thread с использованием целевого параметра <i>callable</i> и набора параметров <i>args</i> или <i>kwargs</i> . Может быть также передан параметр <i>name</i> или <i>group</i> , но обработка последнего не реализована. Принимается также флаг <i>verbose</i> . Любое ненулевое значение <i>daemon</i> задает атрибут/флаг <i>thread.daemon</i>
<code>start()</code>	Запуск выполнения потока
<code>run()</code>	Метод, определяющий функционирование потока (обычно перекрывается разработчиком приложения в подклассе)
<code>join(timeout=None)</code>	Приостановка до завершения запущенного потока; блокировка, если не задан параметр <i>timeout</i> (в секундах)
<code>getName()^a</code>	Возвращаемое имя потока
<code>setName(name)^a</code>	Заданное имя потока
<code>isAlive/is_alive()^b</code>	Булев флаг, указывающий, продолжает ли поток работать
<code>isDaemon()^c</code>	Возвращает True, если поток выполняет функции демона, в противном случае возвращает False
<code>setDaemon(daemonic)^c</code>	Задание флага работы в режиме демона равным указанному булеву значению <i>daemonic</i> (вызов должен осуществляться перед выполнением функции <code>start()</code> для потока)

^a Обозначается как устаревший путем задания (или получения) атрибута `thread.name` или передачи его во время порождения экземпляра.

^b Имена в так называемом *Верблюжьей Стили* (CamelCase) рассматриваются как устаревшие и заменяются, начиная с версии Python 2.6.

^c Метод `is/setDaemon()` обозначается как устаревший путем задания атрибута `thread.daemon`; значение `thread.daemon` может быть также задано во время порождения экземпляра путем указания необязательного значения для демона; новое в версии Python 3.3.

Предусмотрен целый ряд способов, с помощью которых могут создаваться потоки на основе класса Thread. В данной главе рассматриваются три из этих способов, которые мало отличаются друг от друга. Программист может выбрать способ, который является для него наиболее удобным, не говоря уже о том, что выбранный способ должен быть наиболее подходящим с точки зрения приложения и масштабирования в будущем (предпочтительным является последний из приведенных способов).

- Создание экземпляра Thread с передачей функции.
- Создание экземпляра Thread и передача вызываемого экземпляра класса.
- Формирование подкласса Thread и создание экземпляра подкласса.

Как правило, программисты выбирают первый или третий вариант. Последний становится предпочтительным, если требуется создать в большей степени объектно-ориентированный интерфейс, а первый — в противном случае. Второй вариант, откровенно говоря, является немного более громоздким, и, как показывает практика, его применение приводит к созданию программ, более сложных для восприятия.

Создание экземпляра Thread с передачей функции

В первом примере будет лишь создан экземпляр Thread с передачей функции (с ее параметрами) в форме, аналогичной предыдущим примерам. Именно эта функция должна быть выполнена после передачи потоку указания, что он должен начать выполнение. Взяв за основу сценарий `mtsleePB.py` из примера 4.3 и внеся в него корректировки, необходимые для использования объектов Thread, получим сценарий `mtsleePC.py`, как показано в примере 4.4.

Пример 4.4. Использование модуля threading (`mtsleePC.py`)

В классе Thread из модуля threading предусмотрен метод `join()`, который позволяет обеспечить ожидание в основном потоке завершения текущего потока.

```

1  #!/usr/bin/env python
2
3  import threading
4  from time import sleep, ctime
5
6  loops = [4,2]
7
8  def loop(nloop, nsec):
9      print 'start loop', nloop, 'at:', ctime()
10     sleep(nsec)
11     print 'loop', nloop, 'done at:', ctime()
12
13  def main():
14      print 'starting at:', ctime()
15      threads = []
16      nloops = range(len(loops))
17
18      for i in nloops:
19          t = threading.Thread(target=loop,
20                              args=(i, loops[i]))
21          threads.append(t)
22
23      for i in nloops:          # запуск потоков
24          threads[i].start()
25
26      for i in nloops:          # ожидание завершения
27          threads[i].join()      # всех потоков
28
29      print 'all DONE at:', ctime()
30
31  if __name__ == '__main__':
32      main()

```

После выполнения сценария, приведенного в примере 4.4, формируется примерно такой же вывод, как и при вызове предыдущих сценариев:

```
$ mtsleepC.py
starting at: Sun Aug 13 18:16:38 2006
start loop 0 at: Sun Aug 13 18:16:38 2006
start loop 1 at: Sun Aug 13 18:16:38 2006
loop 1 done at: Sun Aug 13 18:16:40 2006
loop 0 done at: Sun Aug 13 18:16:42 2006
all DONE at: Sun Aug 13 18:16:42 2006
```

Так что же фактически изменилось? Удалось избавиться от блокировок, которые приходилось реализовывать при использовании модуля `thread`. Вместо этого создается ряд объектов `Thread`. После создания экземпляра каждого объекта `Thread` остается лишь передать функцию (`target`) и параметры (`args`) и получить взамен экземпляр `Thread`. Наибольшее различие между созданием экземпляра `Thread` (путем вызова `Thread()`) и вызовом `thread.start_new_thread()` состоит в том, что в первом случае запуск нового потока не происходит немедленно. Это удобно с точки зрения синхронизации, особенно если не требуется, чтобы потоки запускались сразу после их создания.

Иными словами, появляется возможность почти одновременно запустить все потоки по окончании их распределения, но не раньше, для чего остается лишь вызвать метод `start()` каждого потока. Кроме того, отпадает необходимость заниматься управлением целым рядом блокировок (выделением, захватом, освобождением, проверкой состояния блокировки и т.д.), поскольку достаточно лишь вызвать метод `join()` для каждого потока. Метод `join()` обеспечивает переход в состояние ожидания до завершения работы потока или до истечения тайм-аута, если он предусмотрен. Использование метода `join()` открывает путь к созданию гораздо более наглядных программ по сравнению с применением бесконечного цикла, в котором происходит ожидание освобождения блокировок (такие блокировки иногда именуются *спин-блокировками*, или “крутящимися” блокировками, именно по той причине, что применяются в бесконечном цикле).

Еще одной важной отличительной особенностью метода `join()` является то, что он вообще не требует вызова. После запуска потока его выполнение происходит до завершения переданной ему функции, после чего осуществляется выход из потока. Если в основном потоке должны быть выполнены какие-то другие действия, кроме ожидания завершения потоков (такие как дополнительная обработка или ожидание новых клиентских запросов), организовать это совсем несложно. Метод `join()` становится удобным, только если требуется обеспечить ожидание завершения потока.

Создание экземпляра `Thread` и передача вызываемого экземпляра класса

Подход, аналогичный передаче функции при создании потока, состоит в применении вызываемого класса и передаче его экземпляра на выполнение; в этом состоит в большей степени объектно-ориентированный способ многопоточного программирования. Такой вызываемый класс воплощает в себе среду выполнения, а это открывает намного больше возможностей по сравнению с применением функции или выбором из ряда функций. Теперь в руках у программиста оказывается вся мощь объекта класса, а не просто единственной функции или даже ряда функций, определяемого списком или кортежем.

После введения в код нового класса ThreadFunc и внесения других небольших изменений в сценарий mtsleepC.py был создан сценарий mtsleepD.py, показанный в примере 4.5.

Пример 4.5. Использование вызываемых классов (mtsleepD.py)

В этом примере передается вызываемый класс (экземпляр), в отличие от отдельной функции. Такой подход является в большей степени объектно-ориентированным по сравнению с применяемым в mtsleepC.py.

```

1  #!/usr/bin/env python
2
3  import threading
4  from time import sleep, ctime
5
6  loops = [4,2]
7
8  class ThreadFunc(object):
9
10     def __init__(self, func, args, name=''):
11         self.name = name
12         self.func = func
13         self.args = args
14
15     def __call__(self):
16         self.func(*self.args)
17
18 def loop(nloop, nsec):
19     print 'start loop', nloop, 'at:', ctime()
20     sleep(nsec)
21     print 'loop', nloop, 'done at:', ctime()
22
23 def main():
24     print 'starting at:', ctime()
25     threads = []
26     nloops = range(len(loops))
27
28     for i in nloops: # создание всех потоков
29         t = threading.Thread(
30             target=ThreadFunc(loop, (i, loops[i])),
31             loop.__name__)
32         threads.append(t)
33
34     for i in nloops: # запуск всех потоков
35         threads[i].start()
36
37     for i in nloops: # ожидание завершения
38         threads[i].join()
39
40     print 'all DONE at:', ctime()
41
42 if __name__ == '__main__':
43     main()

```

После вызова на выполнение сценария mtsleepD.py формируется ожидаемый вывод:

```
$ mtsleepD.py
starting at: Sun Aug 13 18:49:17 2006
start loop 0 at: Sun Aug 13 18:49:17 2006
start loop 1 at: Sun Aug 13 18:49:17 2006
loop 1 done at: Sun Aug 13 18:49:19 2006
loop 0 done at: Sun Aug 13 18:49:21 2006
all DONE at: Sun Aug 13 18:49:21 2006
```

Так что же изменилось на этот раз? Произошло добавление класса `ThreadFunc` и внесены небольшие изменения в процедуру создания экземпляра объекта `Thread`, в которой также порождается `ThreadFunc`, новый вызываемый класс. Фактически в данном примере применяется процедура создания не одного, а двух экземпляров. Рассмотрим класс `ThreadFunc` более подробно.

Этот класс должен быть достаточно общим, для того чтобы его можно было использовать не только с функцией `loop()`, но и с другими функциями, поэтому была добавлена некоторая новая инфраструктура, которая обеспечивает хранение этим классом параметров для функции, самой функции, а также строки с именем функции. Конструктор `__init__()` лишь задает все необходимые значения.

При вызове в коде `Thread` объекта `ThreadFunc` в связи с созданием нового потока вызывается специальный метод `__call__()`. Необходимый набор параметров уже задан, поэтому его не обязательно передавать конструктору `Thread()` и можно вызывать функцию непосредственно.

Подкласс `Thread` и создание экземпляра подкласса

В последнем вводном примере рассмотрим создание подкласса `Thread()`. Как оказалось, применяемая при этом последовательность действий весьма напоминает то, что происходит при создании вызываемого класса, как в предыдущем примере. Код создания подкласса является немного более легким для восприятия, если дело касается создания потоков (строки 29-30). Код сценария `mtsleepE.py` представлен в примере 4.6, затем показан вывод, полученный в результате выполнения этого сценария, а сравнение сценариев `mtsleepE.py` и `mtsleepD.py` оставлено в качестве упражнения для читателя.

Пример 4.6. Создание подкласса `Thread` (`mtsleepE.py`)

Вместо создания экземпляра класса `Thread` создается его подкласс. Благодаря этому открываются более широкие возможности настройки объектов многопоточной поддержки и упрощается осуществление действий по созданию потока.

```
1  #!/usr/bin/env python
2
3  import threading
4  from time import sleep, ctime
5
6  loops = (4, 2)
7
8  class MyThread(threading.Thread):
9      def __init__(self, func, args, name=''):
10         threading.Thread.__init__(self)
11         self.name = name
12         self.func = func
13         self.args = args
```



```

14
15     def run(self):
16         self.func(*self.args)
17
18     def loop(nloop, nsec):
19         print 'start loop', nloop, 'at:', ctime()
20         sleep(nsec)
21         print 'loop', nloop, 'done at:', ctime()
22
23     def main():
24         print 'starting at:', ctime()
25         threads = []
26         nloops = range(len(loops))
27
28         for i in nloops:
29             t = MyThread(loop, (i, loops[i]),
30                           loop.__name__)
31             threads.append(t)
32
33         for i in nloops:
34             threads[i].start()
35
36         for i in nloops:
37             threads[i].join()
38
39         print 'all DONE at:', ctime()
40
41     if __name__ == '__main__':
42         main()

```

Ниже приведен вывод сценария mtsleepE.py. В данном случае полученные результаты вполне соответствуют ожиданию:

```

$ mtsleepE.py
starting at: Sun Aug 13 19:14:26 2006
start loop 0 at: Sun Aug 13 19:14:26 2006
start loop 1 at: Sun Aug 13 19:14:26 2006
loop 1 done at: Sun Aug 13 19:14:28 2006
loop 0 done at: Sun Aug 13 19:14:30 2006
all DONE at: Sun Aug 13 19:14:30 2006

```

Сравнивая исходный код модулей mtsleep4 и mtsleep5, необходимо подчеркнуть наиболее значительные отличия: во-первых, в конструкторе подкласса MyThread приходится вначале вызывать конструктор базового класса (строка 9), и, во-вторых, применявшийся ранее специальный метод `__call__()` должен получить в подклассе имя `run()`.

После этого дополним класс MyThread некоторыми средствами формирования диагностического вывода и сохраним его в отдельном модуле myThread (как показано в примере 4.7). В следующих примерах этот класс будет применяться для импорта. Вместо того чтобы просто вызывать применяемые функции, сохраним результат в атрибуте экземпляра `self.res` и создадим новый метод для получения этого значения, `getResult()`.

Пример 4.7. Подкласс `MyThread` потока (`myThread.py`)

Для того чтобы повысить общность подкласса `Thread` из сценария `mtsleepe.py`, переместим этот подкласс в отдельный модуль и добавим метод `getResult()` для вызова функций, которые формируют возвращаемые значения.

```

1  #!/usr/bin/env python
2
3  import threading
4  from time import ctime
5
6  class MyThread(threading.Thread):
7      def __init__(self, func, args, name=''):
8          threading.Thread.__init__(self)
9          self.name = name
10         self.func = func
11         self.args = args
12
13     def getResult(self):
14         return self.res
15
16     def run(self):
17         print 'starting', self.name, 'at:', \
18             ctime()
19         self.res = self.func(*self.args)
20         print self.name, 'finished at:', \
21             ctime()

```

4.5.2. Другие функции модуля `Threading`

В модуле `Threading`, кроме различных объектов синхронизации и обеспечения многопоточной поддержки, предусмотрены также некоторые поддерживающие функции, представленные в табл. 4.4.

Таблица 4.4. Функции модуля `threading`

Функция	Описание
<code>activeCount/active_count()</code> ^a	Возвращает количество активных в настоящее время объектов <code>Thread</code>
<code>currentThread()/current_thread</code> ^a	Возвращает текущий объект <code>Thread</code>
<code>enumerate()</code>	Возвращает список всех активных в настоящее время объектов <code>Thread</code>
<code>settrace(func)</code> ^b	Задаёт функцию трассировки для всех потоков
<code>setprofile(func)</code> ^b	Задаёт профиль <i>function</i> для всех потоков
<code>stack_size(size=0)</code> ^c	Возвращает размер стека вновь созданных потоков; с учетом потоков, которые будут создаваться в дальнейшем, может быть задан необязательный параметр <i>size</i>

^a Имена в так называемом ВерблюжьемСтиле рассматриваются как устаревшие и заменяются, начиная с версии Python 2.6.

^b Новое в версии Python 2.3.

^c Псевдоним метода `thread.stack_size()`. Метод и его псевдоним впервые введены в версии Python 2.5.

4.6. Сравнение однопоточного и многопоточного выполнения

В сценарии `mtfacfib.py`, представленном в примере 4.8, происходит сравнение хода выполнения рекурсивных функций, включая функции вычисления числа Фибоначчи, факториала и суммы. В этом сценарии все три функции выполняются в однопоточном режиме. После этого та же задача решается с использованием потоков, что позволяет показать одно из преимуществ применения среды многопоточной поддержки.

Пример 4.8. Функции вычисления числа Фибоначчи, факториала и суммы (`mtfacfib.py`)

В этом многопоточном приложении выполняются три отдельные рекурсивные функции, сначала в однопоточном режиме, а затем с применением альтернативной организации работы с несколькими потоками.

```
1  #!/usr/bin/env python
2
3  from myThread import MyThread
4  from time import ctime, sleep
5
6  def fib(x):
7      sleep(0.005)
8      if x < 2: return 1
9      return (fib(x-2) + fib(x-1))
10
11 def fac(x):
12     sleep(0.1)
13     if x < 2: return 1
14     return (x * fac(x-1))
15
16 def sum(x):
17     sleep(0.1)
18     if x < 2: return 1
19     return (x + sum(x-1))
20
21 funcs = [fib, fac, sum]
22 n = 12
23
24 def main():
25     nfuncs = range(len(funcs))
26
27     print '*** SINGLE THREAD'
28     for i in nfuncs:
29         print 'starting', funcs[i].__name__, 'at:', \
30             ctime()
31         print funcs[i](n)
32         print funcs[i].__name__, 'finished at:', \
33             ctime()
34
35     print '\n*** MULTIPLE THREADS'
36     threads = []
37     for i in nfuncs:
38         t = MyThread(funcs[i], (n,))
39         funcs[i].__name__
40         threads.append(t)
```

```

41
42     for i in nfuncs:
43         threads[i].start()
44
45     for i in nfuncs:
46         threads[i].join()
47         print threads[i].getResult()
48
49     print 'all DONE'
50
51 if __name__ == '__main__':
52     main()

```

Выполнение в однопоточном режиме сводится к тому, что функции вызываются одна за другой и после завершения их работы сразу же отображаются полученные результаты вызова.

Если же выполнение функций происходит в многопоточном режиме, то результат не отображается немедленно. Желательно, чтобы класс `MyThread` был настолько общим, насколько это возможно (способным выполнять вызываемые функции, которые формируют и не формируют вывод), поэтому вызов метода `getResult()` откладывается до самого конца, что позволяет показать значения, возвращенные в каждом вызове функции, после того, как все будет сделано.

В данном примере вызовы всех функций выполняются очень быстро (вернее, исключением может стать вызов функции вычисления числа Фибоначчи), поэтому, как можно заметить, мы были вынуждены добавить вызовы `sleep()` к каждой функции для замедления процесса выполнения, чтобы можно было убедиться в том, что многопоточная организация действительно способствует повышению производительности, если в разных потоках применяются функции с различным временем выполнения. Безусловно, на практике дополнять функции вызовами `sleep()` нет необходимости. Так или иначе, получим такой вывод:

```

$ mtfacfib.py
*** SINGLE THREAD
starting fib at: Wed Nov 16 18:52:20 2011
233
fib finished at: Wed Nov 16 18:52:24 2011
starting fac at: Wed Nov 16 18:52:24 2011
479001600
fac finished at: Wed Nov 16 18:52:26 2011
starting sum at: Wed Nov 16 18:52:26 2011
78
sum finished at: Wed Nov 16 18:52:27 2011

*** MULTIPLE THREADS
starting fib at: Wed Nov 16 18:52:27 2011
starting fac at: Wed Nov 16 18:52:27 2011
starting sum at: Wed Nov 16 18:52:27 2011

fac finished at: Wed Nov 16 18:52:28 2011
sum finished at: Wed Nov 16 18:52:28 2011
fib finished at: Wed Nov 16 18:52:31 2011
233
479001600
78
all DONE

```

4.7. Практическое применение многопоточной обработки

До сих пор были представлены лишь упрощенные, применяемые в качестве примеров фрагменты кода, которые весьма далеки от того, что должно применяться в реальном приложении. Фактически единственное назначение этих примеров состоит лишь в том, чтобы показать потоки в работе и продемонстрировать различные способы их создания. При этом во всех примерах запуск потоков и ожидание их завершения происходит почти одинаково, а действия, выполняемые потоками, главным образом сводятся к приостановке.

Кроме того, как уже было сказано в разделе 4.3.1, виртуальная машина Python в действительности работает в однопоточном режиме (с применением глобальной блокировки интерпретатора), поэтому достижение большего распараллеливания в программе Python возможно, только если многопоточная организация применяется в приложении, ограничиваемом пропускной способностью ввода-вывода, а не пропускной способностью процессора, в котором так или иначе происходит лишь циклическая передача управления от одного процесса к другому. По этой причине мы рассмотрим пример приложения первого типа и в качестве дальнейшего упражнения попытаемся перенести его в версию Python 3, чтобы можно было понять, что с этим связано.

4.7.1. Пример ранжирования книг

Сценарий `bookrank.py`, показанный в примере 4.9, весьма прост. Он выполняет переход на сайт интернет-торговли Amazon и запрашивает текущее ранжирование книг, написанных вашим покорным слугой. В рассматриваемом примере кода представлены функция `getRanking()`, в которой используется регулярное выражение для извлечения и возврата текущего ранжирования, и функция `showRanking()`, которая отображает результаты для пользователя.

Следует отметить, что согласно условиям использования *“компания Amazon предоставляет пользователю сайта ограниченные права доступа к сайту и использования его в личных целях, но не позволяет загружать содержимое сайта (исключая кэширование страниц) либо изменять его или любую его часть без явно выраженного письменного согласия Amazon.”* Задача рассматриваемого приложения состоит в том, чтобы выбрать данные о текущем ранжировании конкретной книги, а затем отбросить остальные сведения о ранжировании; в данном случае даже кэширование страницы не применяется.

В примере 4.9 представлена первая (и почти последняя) попытка создания сценария `bookrank.py`, который не относится к многопоточной версии.

Пример 4.9. Программа извлечения с веб-страницы данных о ранжировании книг (`bookrank.py`)

В этом сценарии выполняются вызовы, обеспечивающие загрузку информации о ранжировании книг через отдельные потоки.

```
1  #!/usr/bin/env python
2
3  from atexit import register
4  from re import compile
5  from threading import Thread
```

```

6  from time import ctime
7  from urllib2 import urlopen as uopen
8
9  REGEX = compile('#{[\\d,]+} in Books ')
10 AMZN = 'http://amazon.com/dp/'
11 ISBNs = {
12     '0132269937': 'Core Python Programming',
13     '0132356139': 'Python Web Development with Django',
14     '0137143419': 'Python Fundamentals',
15: }
16
17 def getRanking(isbn):
18     page = uopen('%s%s' % (AMZN, isbn)) #или str.format()
19     data = page.read()
20     page.close()
21     return REGEX.findall(data)[0]
22
23 def _showRanking(isbn):
24     print '- %r ranked %s' % (
25         ISBNs[isbn], getRanking(isbn))
26
27 def _main():
28     print 'At', ctime(), 'on Amazon...'
29     for isbn in ISBNs:
30         _showRanking(isbn)
31
32 @register
33 def _atexit():
34     print 'all DONE at:', ctime()
35
36 if __name__ == '__main__':
37     main()

```

Построчное объяснение

Строки 1–7

Это строки запуска и импорта. Для определения того, когда будет завершено выполнение сценария, используется функция `atexit.register()` (почему это сделано, будет описано ниже). Кроме того, для работы с шаблоном, с помощью которого извлекаются сведения о ранжировании книг со страниц с описанием товаров на сайте Amazon, применяется функция поддержки регулярных выражений `re.compile()`. Затем результаты импорта `threading.Thread` сохраняются для использования в намеченном на будущее усовершенствованном варианте сценария (об этом немного позже), вызывается метод `time.ctime()` для получения строки с текущей отметкой времени, а для получения доступа к каждой ссылке применяется метод `urllib2.urlopen()`.

Строки 9–15

В этом сценарии используются три константы. Первой из них является `REGEX`, объект регулярного выражения (полученный путем компиляции шаблона регулярного выражения, который согласуется с данными по ранжированию книги); вторая константа — `AMZN`, префикс каждой ссылки на товары Amazon. За этим префиксом следует ISBN (International Standard Book Number) искомой книги; ISBN служит для

книги уникальным обозначением и позволяет отличить ее от других. Предусмотрены два стандарта ISBN: ISBN-10, с десятисимвольным обозначением, и ISBN-13, принятый позднее стандарт, в котором применяются тринадцать символов. В настоящее время поисковая система Amazon распознает ISBN обоих типов, поэтому для краткости будем использовать только ISBN-10. Искомые ISBN хранятся в словаре ISBNs (который представляет собой третью константу) наряду с соответствующими названиями книг.

Строки 17–21

Функция `getRanking()` предназначена для получения ISBN, создания конечного значения URL, которое применяется для доступа к серверам Amazon, а затем вызова для этого URL метода `urllib2.urlopen()`. Для соединения воедино компонентов значения URL используется оператор форматирования строки (в строке 18), но если работа ведется с версией Python 2.6 или последующей версией, то можно также попытаться воспользоваться методом `str.format()`, например `'{0}{1}'.format(AMZN, isbn)`.

После получения полного URL вызывается метод `urllib2.urlopen()` (в данном сценарии в качестве него применяется сокращение `urlopen()`), который в случае успеха возвращает файловый объект после ответа веб-сервера. Затем происходит вызов функции `read()` для загрузки всей веб-страницы, и файловый объект закрывается. Если регулярное выражение составлено правильно, то при его использовании должно происходить одно и только одно сопоставление, поэтому достаточно извлечь необходимый результат из сформированного списка (все остальные результаты будут пропущены) и вернуть его в вызывающую функцию.

Строки 23–25

Функция `_showRanking()` представляет собой всего лишь короткий фрагмент кода, в котором берется ISBN, осуществляется поиск названия книги, которую представляет этот ISBN, вызывается метод `getRanking()` для получения текущего ранга книги на веб-сайте Amazon, после чего ISBN и название передаются пользователю. В имени этого метода применяется префикс в виде одного знака подчеркивания. Этот префикс служит в качестве указания, что данная функция является специальной, предназначена для использования только в данном модуле и не должна быть импортирована в каком-либо другом приложении в составе библиотечного или вспомогательного модуля.

Строки 27–30

Функция `_main()` также относится к категории специальных и вызывается на выполнение, только если данный модуль вызван непосредственно из командной строки (а не импортируется для использования другим модулем). Происходит отображение времени начала и окончания (чтобы пользователь мог определить, сколько времени потребовалось для выполнения всего сценария), затем вызывается функции `_showRanking()` применительно к каждому ISBN для поиска и отображения данных о текущем ранжировании каждой книги на сайте Amazon.

Строки 32–37

В этих строках выполняются действия, которые прежде нами не рассматривались. Рассмотрим назначение функции `atexit.register()`. Такие функции принято называть декораторами. Декораторы — одна из разновидностей служебных функций.

В данном случае с помощью указанной функции происходит регистрация функции выхода в интерпретаторе Python. Это равносильно запросу, чтобы интерпретатор вызвал некоторую специальную функцию непосредственно перед завершением сценария. (Вместо вызова декоратора можно также применить конструкцию `register(_atexit())`).

Рассмотрим причины использования декоратора в данном коде. Прежде всего необходимо отметить, что можно было бы вполне обойтись без него. Оператор вывода может быть размещен в последней части функции `_main()`, в строках 27–31, но в действительности при этом организация программ оставляла бы желать лучшего. Кроме того, здесь демонстрируется пример применения функционального средства, без которого при определенных обстоятельствах нельзя было бы обойтись в приложении, применяемом на производстве. Предполагается, что читатель сам сможет определить назначение строк 36–37, поэтому перейдем к описанию полученного вывода:

```
$ python bookrank.py
At Wed Mar 30 22:11:19 2011 PDT on Amazon...
- 'Core Python Programming' ranked 87,118
- 'Python Fundamentals' ranked 851,816
- 'Python Web Development with Django' ranked 184,735
all DONE at: Wed Mar 30 22:11:25 2011
```

Заслуживает внимания то, что в данном примере разделены процессы получения данных (`getRanking()`) и их отображения (`_showRanking()` и `_main()`), что позволяет при желании вместо отображения результатов для пользователя с помощью терминала предусмотреть какой-то другой способ обработки вывода. На практике может потребоваться отправить эти данные назад с помощью веб-шаблона, сохранить в базе данных, вывести в виде текста на экран мобильного телефона и т.д. Если бы весь этот код был помещен в одну функцию, то было бы сложнее обеспечить его повторное использование и (или) отправить по другому назначению.

Кроме того, если компания Amazon изменит компоновку своих страниц с описанием товаров, то может потребоваться лишь изменить регулярное выражение, предназначенное для выборки данных с веб-страниц, чтобы по-прежнему иметь возможность извлекать сведения о книгах. Следует также отметить, что в этом простом примере вполне оправдывает себя способ обработки данных с помощью регулярного выражения (который может быть даже заменен простыми традиционными операциями работы со строками), но в какое-то время может потребоваться более мощный синтаксический анализатор разметки, такой как `HTMLParser` из стандартной библиотеки, а возможно, нельзя будет обойтись без таких инструментов сторонних производителей, как `BeautifulSoup`, `html5lib` или `lxml`. (Некоторые из этих инструментов будут продемонстрированы в главе 9.)

Добавление в программу средств многопоточной поддержки

Очевидно, что приведенный выше пример все еще относится к категории несложных однопоточных программ. Теперь перейдем к внесению изменений в приложение, чтобы в нем вместо этого использовались потоки. Это приложение, ограничиваемое пропускной способностью ввода-вывода, поэтому вполне подходит для применения в нем многопоточной организации. Для упрощения на первых порах мы не будем использовать классы и объектно-ориентированное программирование; вместо этого в программе будет применяться непосредственно метод `threading.Thread`, поэтому

данный пример в большей степени должен напоминать сценарий `mtsleepC.py`, чем любой из последующих примеров. Дополнением является лишь то, что в приложении создаются потоки, которые немедленно запускаются.

Возьмем за основу ранее разработанное приложение и изменим вызов `_showRanking(isbn)` следующим образом:

```
Thread(target=_showRanking, args=(isbn,)).start().
```

Получен именно такой результат, который требуется! Теперь в нашем распоряжении имеется окончательная версия сценария `bookrank.py`, которая показывает, что это приложение (как правило) выполняется быстрее, чем предыдущее, благодаря дополнительному распараллеливанию. Тем не менее быстроедействие приложения ограничивается тем, насколько быстро будет получен ответ, обработка которого потребовала больше всего времени.

```
$ python bookrank.py
At Thu Mar 31 10:11:32 2011 on Amazon...
- 'Python Fundamentals' ranked 869,010
- 'Core Python Programming' ranked 36,481
- 'Python Web Development with Django' ranked 219,228
all DONE at: Thu Mar 31 10:11:35 2011
```

Как показывает полученный вывод, вместо шести секунд, в течение которых выполнялась однопоточная версия, для многопоточной достаточно трех. Кроме того, важно отметить, что в окончательно полученном выводе последовательность результатов может изменяться в зависимости от времени завершения работы потоков, в отличие от однопоточной версии. В версии, в которой не применялась многопоточная организация, последовательность расположения результатов всегда определяется ключами словаря, а теперь все запросы выполняются параллельно и вывод формируется в той последовательности, которая определяется значениями времени завершения отдельных потоков.

В предыдущих примерах (в сценариях `mtsleepX.py`) применительно ко всем потокам вызывался метод `Thread.join()` для блокирования выполнения до завершения каждого потока. Это равносильно блокированию дальнейшей работы основного потока до завершения всех потоков, поэтому инструкция вывода на печать “all DONE at” вызывается после того, как действительно закончится вся работа.

В указанных примерах не было необходимости применять метод `join()` ко всем потокам, поскольку ни один из потоков не функционировал в качестве демона. В основном потоке не происходит выход из сценария до тех пор, пока не произойдет успешное или неудачное завершение всех порожденных потоков. Опираясь на эти рассуждения, мы удалили все вызовы `join()` из сценария `mtsleepF.py`. Тем не менее следует учитывать, что неправильно было бы отображать строку “all done” (все сделано) на том же этапе выполнения сценария, как и прежде.

Строка “all done” должна быть выведена в основном потоке, т.е. до завершения дочерних потоков, чтобы не было необходимости вызывать оператор печати, выше функции `_main()`. Этот оператор `print` можно поместить в одно из двух мест в сценарии: после строки 37, где происходит возврат из функции `_main()` (самая последняя строка, выполняемая в сценарии), или в месте, которое определяется в связи с использованием метода `atexit.register()` для регистрации функции выхода. Эта тема в настоящей книге еще не рассматривалась, и в дальнейшем мы к ней обязательно вернемся, но именно здесь удобно впервые затронуть вопрос регистрации

функций. Важно также, что для регистрации функций применяется интерфейс, который останется неизменным после перехода от Python 2 к Python 3 (это — тема следующего раздела).

Перенос приложения в версию Python 3

3.x

Теперь перейдем к рассмотрению еще одного варианта данного сценария, который предназначен для работы с версией Python 3. С этой темой необходимо ознакомиться, изучая пути переноса проектов и приложений из текущей версии интерпретатора Python в последующую версию. К счастью, эту работу не требуется выполнять вручную, поскольку уже предусмотрены необходимые инструменты, одним из которых является инструмент `2to3`. Вообще говоря, предусмотрены два способа его использования:

```
$ 2to3 foo.py      # в выводе показаны только различия
$ 2to3 -w foo.py  # переопределяет с помощью кода версии 3.x
```

В первой команде инструмент `2to3` лишь отображает различия между исходным сценарием в версии 2.x и сформированным с его помощью эквивалентом для версии 3.x. Флаг `-w` служит для инструмента `2to3` указанием, что исходный сценарий должен быть перезаписан вновь полученным сценарием для версии 3.x, а сценарий для версии 2.x переименован в `foo.py.bak`.

Вызовем на выполнение инструмент `2to3` применительно к файлу сценария `bookrank.py` с перезаписью существующего файла. Предусмотрен не только вывод различий; сохраняется также новая версия, как уже было сказано:

```
$ 2to3 -w bookrank.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
--- bookrank.py (original)
+++ bookrank.py (refactored)
@@ -4,7 +4,7 @@
     from re import compile
     from threading import Thread
     from time import ctime
-    from urllib2 import urlopen as uopen
+    from urllib.request import urlopen as uopen

    REGEX = compile('#([\\d,]+) in Books ')
    AMZN = 'http://amazon.com/dp/'
@@ -21,17 +21,17 @@
     return REGEX.findall(data)[0]

def _showRanking(isbn):
-    print '- %r ranked %s' % (
-        ISBNs[isbn], getRanking(isbn))
+    print('- %r ranked %s' % (
+        ISBNs[isbn], getRanking(isbn)))

def _main():
-    print 'At', ctime(), 'on Amazon...'
```

```
+ print('At', ctime(), 'on Amazon...')
for isbn in ISBNs:
    Thread(target=_showRanking, args=(isbn,)).start()#_showRanking(isbn)

@register
def _atexit():
- print 'all DONE at:', ctime()
+ print('all DONE at:', ctime())

if __name__ == '__main__':
    _main()
RefactoringTool: Files that were modified:
RefactoringTool: bookrank.py
```

Следующий шаг читатели могут рассматривать как необязательный. Достаточно лишь отметить, что в нем рассматриваемые файлы были переименованы в `bookrank.py` и `bookrank3.py` с использованием команд POSIX (пользователи компьютеров с операционной системой Windows должны использовать команду `ren`):

```
$ mv bookrank.py bookrank3.py
$ mv bookrank.py.bak bookrank.py
```

Разумеется, было бы желательно, чтобы преобразование сценария для использования в новой версии интерпретатора прошло идеально, чтобы не пришлось ни о чем заботиться, приступая к работе со сценарием нового поколения. Однако в данном случае произошло нечто непредвиденное и в каждом потоке возникает исключение (приведенный вывод относится только к одному потоку; нет смысла показывать результаты для других потоков, поскольку они являются такими же):

```
$ python3 bookrank3.py
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/
    3.2/lib/python3.2/threading.py", line 736, in
    _bootstrap_inner
    self.run()
  File "/Library/Frameworks/Python.framework/Versions/
    3.2/lib/python3.2/threading.py", line 689, in run
    self._target(*self._args, **self._kwargs)
  File "bookrank3.py", line 25, in _showRanking
    ISBNs[isbn], getRanking(isbn))
  File "bookrank3.py", line 21, in getRanking
    return REGEX.findall(data)[0]
TypeError: can't use a string pattern on a bytes-like object
:
```

Что же случилось? По-видимому, проблема заключается в том, что регулярное выражение представлено в виде строки Юникода, тогда как данные, полученные с помощью метода `read()` файлового объекта (возвращенного функцией `urlopen()`), имеют вид строки ASCII/bytes. Чтобы исправить эту ошибку, откомпилируем вместо текстовой строки объект `bytes`. Для этого внесем изменения в строку 9, чтобы в методе `re.compile()` производилась компиляция строки `bytes` (добавим строку `bytes`). Для этого добавим обозначение `b` строки `bytes` непосредственно перед открывающей кавычкой следующим образом:

```

REGEX = compile(b'#([\d,]+) in Books ')
Now let's try it again:
$ python3 bookrank3.py
At Sun Apr  3 00:45:46 2011 on Amazon...
- 'Core Python Programming' ranked b'108,796'
- 'Python Web Development with Django' ranked b'268,660'
- 'Python Fundamentals' ranked b'969,149'
all DONE at: Sun Apr  3 00:45:49 2011

```

Опять что-то не так! Что же случилось теперь? Безусловно, результат стал немного лучше (нет ошибок), но выглядит странно. В данных ранжирования, полученных с помощью регулярных выражений, после передачи в функцию `str()` отображаются символы `b` и кавычки. Для устранения этого недостатка первым побуждением может стать попытка применить операцию получения среза строки, которая также выглядит довольно неуклюже:

```

>>> x = b'xxx'
>>> repr(x)
"b'xxx'"
>>> str(x)
"b'xxx'"
>>> str(x)[2:-1]
'xxx'

```

Тем не менее более подходящий вариант состоит в применении операции преобразования данных в действительное значение (строка в Юникоде, возможно, с использованием UTF-8):

```

>>> str(x, 'utf-8')
'xxx'

```

Для реализации этого решения в текущем сценарии внесем аналогичное изменение в строку 53, чтобы она выглядела следующим образом:

```

return str(REGEX.findall(data)[0], 'utf-8')

```

После этого вывод сценария для версии Python 3 полностью совпадает с тем, что получен в сценарии Python 2:

```

$ python3 bookrank3.py
At Sun Apr  3 00:47:31 2011 on Amazon...
- 'Python Fundamentals' ranked 969,149
- 'Python Web Development with Django' ranked 268,660
- 'Core Python Programming' ranked 108,796
all DONE at: Sun Apr  3 00:47:34 2011

```

Вообще говоря, практика показывает, что перенос сценария из версии 2.x в версию 3.x осуществляется по аналогичному принципу: необходимо убедиться, что код проходит все тесты модульности и интеграции, провести основное преобразование с использованием инструмента `2to3` (и других инструментов), а затем устранить возможные расхождения, добиваясь того, чтобы код успешно выполнялся и проходил такие же проверки, как и исходный сценарий. Попробуем повторить это упражнение снова на следующем примере, в котором демонстрируется использование синхронизации с помощью потоков.

4.7.2. Примитивы синхронизации

В основной части этой главы рассматривались основные концепции многопоточной организации и было показано, как использовать многопоточность в приложениях Python. Однако в этом изложении не затрагивался один очень важный аспект многопоточного программирования: синхронизация. Довольно часто в многопоточном коде содержатся определенные функции или блоки, в которых необходимо (или желательно) ограничить количество выполняемых потоков до одного. Обычно такие ситуации обнаруживаются при внесении изменений в базу данных, обновлении файла или выполнении подобных действий, при которых может возникнуть состояние состязания. Как уже было сказано в этой главе, такое состояние проявляется, если код допускает появление нескольких путей выполнения или вариантов поведения либо формирование несогласованных данных, если один поток будет запущен раньше другого, или наоборот. (С дополнительными сведениями о состояниях состязания можно ознакомиться на странице http://en.wikipedia.org/wiki/Race_condition.)

В таких случаях возникает необходимость обеспечения синхронизации. Синхронизация должна использоваться, если к какому-то из критических участков кода могут подойти одновременно несколько потоков (см. http://en.wikipedia.org/wiki/Critical_section), но в каждый конкретный момент времени должно быть разрешено дальнейшее выполнение только одного потока. Программист регламентирует прохождение потоков и для управления ими выбирает подходящие примитивы синхронизации, или механизмы управления потоками, с помощью которых вводит в действие синхронизацию. Предусмотрено несколько различных методов синхронизации процессов (см. [http://en.wikipedia.org/wiki/Synchronization_\(computer_science\)](http://en.wikipedia.org/wiki/Synchronization_(computer_science))), часть которых поддерживается языком Python. Эта поддержка предоставляет достаточно возможностей для выбора метода, наиболее подходящего для конкретной задачи.

Методы синхронизации уже были представлены ранее, в начале этого раздела, поэтому перейдем к рассмотрению нескольких примеров сценариев, в которых используются примитивы синхронизации двух типов: блокировки/мьютексы и семафоры. Блокировка относится к числу самых простых среди всех механизмов синхронизации и находится на самом низком уровне, а семафоры предназначены для применения в таких ситуациях, в которых несколько потоков конкурируют друг с другом, стремясь получить доступ к ограниченным ресурсам. Понять назначение блокировок проще, поэтому начнем рассмотрение примитивов синхронизации с них, а затем перейдем к семафорам.

4.7.3. Пример применения блокировки

Блокировки, как и следовало ожидать, имеют два состояния: заблокированное и разблокированное. Блокировки поддерживают только две функции: `acquire` и `release`. Эти функции действуют в полном соответствии с их именами — захват и освобождение.

Иногда необходимость пройти критический участок кода возникает в нескольких потоках. В таком случае можно организовать конкуренцию между потоками за блокировку, и первый поток, который сможет ее захватить, получит разрешение войти в критический участок и выполнить содержащийся в нем код. Все остальные одновременно поступающие потоки блокируются до того времени, когда первый поток завершит свою работу, выйдет из критического участка и освободит блокировку.

С этого момента возможность захватить блокировку и войти в критический участок получает любой из оставшихся ожидающих потоков. Заслуживает внимания то, что отсутствует какое-либо упорядочение потоков, работа которых организована с помощью блокировок (т.е. применяется принцип простой очереди — “первым пришел, первым обслуживается”); процесс выбора потока-победителя не детерминирован и может зависеть даже от применяемой реализации Python.

Рассмотрим, с чем связана необходимость применения блокировок. Сценарий `mtsleepF.py` представляет собой приложение, в котором происходит порождение случайным образом выбранного количества потоков, в каждом из которых осуществляется выход после завершения работы. Рассмотрим следующий базовый фрагмент исходного кода (для версии Python 2):

```
from atexit import register
from random import randrange
from threading import Thread, currentThread
from time import sleep, ctime

class CleanOutputSet(set):
    def __str__(self):
        return ', '.join(x for x in self)

loops = (randrange(2,5) for x in xrange(randrange(3,7)))
remaining = CleanOutputSet()

def loop(nsec):
    myname = currentThread().name
    remaining.add(myname)
    print '[%s] Started %s' % (ctime(), myname)
    sleep(nsec)
    remaining.remove(myname)
    print '[%s] Completed %s (%d secs)' % (
        ctime(), myname, nsec)
    print '      (remaining: %s)' % (remaining or 'NONE')

def _main():
    for pause in loops:
        Thread(target=loop, args=(pause,)).start()

@register
def _atexit():
    print 'all DONE at:', ctime()
```

Более подробное построчное описание кода мы приведем вслед за окончательным вариантом сценария, в котором применяются блокировки, но вкратце можно отметить, что сценарий `mtsleepF.py` по сути лишь дополняет приведенные ранее примеры. Как и в примере сценария `bookrank.py`, немного упростим код. Для этого отложим на время применение средств объектно-ориентированного программирования, исключим список объектов потока и операции `join()` с потоками и снова введем в действие метод `atexit.register()` (по тем же причинам, как и в коде `bookrank.py`).

Проведем еще одно небольшое изменение по отношению к приведенным ранее примерам `mtsleepX.py`. Вместо жесткого задания пары операций приостановки циклов/потоков на 4 и 2 секунды соответственно, внесем некую неопределенность, создавая случайным образом от 3 до 6 потоков, каждый из которых может приостанавливаться на какой-то промежуток времени от 2 до 4 секунд.

В этом сценарии применяются также некоторые новые средства, причем наиболее заметным среди них является использование множества для хранения имен оставшихся потоков, которые все еще функционируют. Причина, по которой создается подкласс объекта множества вместо непосредственного использования самого класса, состоит в том, что это позволяет продемонстрировать еще один вариант использования множества, в котором изменяется применяемое по умолчанию для вывода строковое представление множества.

При использовании операции вывода содержимого множества формируются примерно такие результаты: `set([X, Y, Z, ...])`. Однако это не очень удобно, поскольку потенциальные пользователи нашего приложения не знают (и не должны знать) о том, что такое множества и для чего они используются в программе. Таким образом, необходимо вместо этого вывести данные, которые выглядят примерно как `X, Y, Z, ...`. Именно по этой причине мы создали подкласс класса `set` и реализовали его метод `__str__()`.

После этого изменения, при условии, что вся остальная часть сценария будет работать правильно, должен сформироваться аккуратный вывод, который будет иметь подходящее выравнивание:

```
$ python mtsleepF.py
[Sat Apr  2 11:37:26 2011] Started Thread-1
[Sat Apr  2 11:37:26 2011] Started Thread-2
[Sat Apr  2 11:37:26 2011] Started Thread-3
[Sat Apr  2 11:37:29 2011] Completed Thread-2 (3 secs)
                        (remaining: Thread-3, Thread-1)
[Sat Apr  2 11:37:30 2011] Completed Thread-1 (4 secs)
                        (remaining: Thread-3)
[Sat Apr  2 11:37:30 2011] Completed Thread-3 (4 secs)
                        (remaining: NONE)
all DONE at: Sat Apr  2 11:37:30 2011
However, if you're unlucky, you might get strange output such as this pair of example
executions:
$ python mtsleepF.py
[Sat Apr  2 11:37:09 2011] Started Thread-1
[Sat Apr  2 11:37:09 2011] Started Thread-2
[Sat Apr  2 11:37:09 2011] Started Thread-3
[Sat Apr  2 11:37:12 2011] Completed Thread-1 (3 secs)
[Sat Apr  2 11:37:12 2011] Completed Thread-2 (3 secs)
                        (remaining: Thread-3)
                        (remaining: Thread-3)
[Sat Apr  2 11:37:12 2011] Completed Thread-3 (3 secs)
                        (remaining: NONE)
all DONE at: Sat Apr  2 11:37:12 2011

$ python mtsleepF.py
[Sat Apr  2 11:37:56 2011] Started Thread-1
[Sat Apr  2 11:37:56 2011] Started Thread-2
[Sat Apr  2 11:37:56 2011] Started Thread-3
[Sat Apr  2 11:37:56 2011] Started Thread-4

[Sat Apr  2 11:37:58 2011] Completed Thread-2 (2 secs)
[Sat Apr  2 11:37:58 2011] Completed Thread-4 (2 secs)
                        (remaining: Thread-3, Thread-1)
                        (remaining: Thread-3, Thread-1)

[Sat Apr  2 11:38:00 2011] Completed Thread-1 (4 secs)
```

```
(remaining: Thread-3)
[Sat Apr  2 11:38:00 2011] Completed Thread-3 (4 secs)
(remaining: NONE)
all DONE at: Sat Apr  2 11:38:00 2011
```

Что же произошло? Во-первых, очевидно, что результаты далеко не однородны (поскольку возможность выполнять операции ввода-вывода параллельно предоставлена сразу нескольким потокам). Подобное чередование результирующих данных можно было также наблюдать и в некоторых примерах приведенного ранее кода. Во-вторых, обнаруживается такая проблема, что два потока изменяют значение одной и той же переменной (множества, содержащего имена оставшихся потоков).

Операции ввода-вывода и операции доступа к одной и той же структуре данных входят в состав критических разделов кода, поэтому необходимы блокировки, которые смогли бы воспрепятствовать одновременному вхождению в эти разделы нескольких потоков. Чтобы ввести в действие блокировки, необходимо добавить строку кода для импорта объекта `Lock` (или `RLock`), создать объект блокировки и внести дополнения или изменения в код, позволяющие применять блокировки в нужных местах:

```
from threading import Thread, Lock, currentThread
lock = Lock()
```

Теперь необходимо обеспечить установку и снятие блокировки. В следующем коде показаны вызовы `acquire()` и `release()`, которые должны быть введены в функцию `loop()`:

```
def loop(nsec):
    myname = currentThread().name
    lock.acquire()
    remaining.add(myname)
    print '[%s] Started %s' % (ctime(), myname)
    lock.release()
    sleep(nsec)
    lock.acquire()
    remaining.remove(myname)
    print '[%s] Completed %s (%d secs)' % (
        ctime(), myname, nsec)
    print '    (remaining: %s)' % (remaining or 'NONE')
    lock.release()
```

После внесения изменений полученный вывод уже не должен содержать прежних искажений:

```
$ python mtsleepF.py
[Sun Apr  3 23:16:59 2011] Started Thread-1
[Sun Apr  3 23:16:59 2011] Started Thread-2
[Sun Apr  3 23:16:59 2011] Started Thread-3
[Sun Apr  3 23:16:59 2011] Started Thread-4
[Sun Apr  3 23:17:01 2011] Completed Thread-3 (2 secs)
    (remaining: Thread-4, Thread-2, Thread-1)
[Sun Apr  3 23:17:01 2011] Completed Thread-4 (2 secs)
    (remaining: Thread-2, Thread-1)
```



```
[Sun Apr  3 23:17:02 2011] Completed Thread-1 (3 secs)
      (remaining: Thread-2)
[Sun Apr  3 23:17:03 2011] Completed Thread-2 (4 secs)
      (remaining: NONE)
all DONE at: Sun Apr  3 23:17:03 2011
```

Исправленная (и окончательная) версия `mtsleepF.py` показана в примере 4.10.

Пример 4.10. Сценарий, в котором предусмотрены блокировки и введены дополнительные средства случайного выбора (`mtsleepF.py`)

В этом примере демонстрируется использование блокировок и других инструментов обеспечения многопоточного функционирования.

```
1  #!/usr/bin/env python
2
3  from atexit import register
4  from random import randrange
5  from threading import Thread, Lock, currentThread
6  from time import sleep, ctime
7
8  class CleanOutputSet(set):
9      def __str__(self):
10         return ', '.join(x for x in self)
11
12  lock = Lock()
13  loops = (randrange(2,5) for x in xrange(randrange(3,7)))
14  remaining = CleanOutputSet()
15
16  def loop(nsec):
17      myname = currentThread().name
18      lock.acquire()
19      remaining.add(myname)
20      print '[%s] Started %s' % (ctime(), myname)
21      lock.release()
22      sleep(nsec)
23      lock.acquire()
24      remaining.remove(myname)
25      print '[%s] Completed %s (%d secs)' % (
26          ctime(), myname, nsec)
27      print '      (remaining: %s)' % (remaining or 'NONE')
28      lock.release()
29
30  def _main():
31      for pause in loops:
32          Thread(target=loop, args=(pause,)).start()
33
34  @register
35  def _atexit():
36      print 'all DONE at:', ctime()
37
38  if __name__ == '__main__':
39      main()
```

Построчное объяснение

Строки 1–6

2.6

Это обычные строки запуска и импорта. Следует учитывать, что начиная с версии 2.6 метод `threading.currentThread()` переименован в `threading.current_thread()`, но первый компонент имени метода остался неизменным в целях обеспечения обратной совместимости.

Строки 8–10

Это подкласс множества, о котором речь шла выше. Он содержит реализацию метода `__str__()`, который позволяет вместо формата вывода, применяемого по умолчанию, перейти к формату, представляющему собой строку элементов, разделенную запятыми.

Строки 12–14

В состав применяемых глобальных переменных входят блокировка, экземпляр позаимствованного из предыдущего кода откорректированного множества и случайно выбранного числа потоков (от трех до шести), каждый из которых останавливается (вернее, приостанавливается) на время от двух до четырех секунд.

Строки 16–28

Функция `loop()` сохраняет имя текущего потока, в котором она выполняется, затем захватывает блокировку, что позволяет сделать неразрывным (атомарным) следующий ряд операций: добавление имени к множеству `remaining` и формирование вывода с указанием на начало потока (с этого момента больше никакой другой поток не сможет войти в критический участок кода). После освобождения блокировки этот поток приостанавливается на время в секундах, выбранное случайным образом, затем повторно захватывает блокировку, чтобы сформировать свой окончательный вывод перед ее освобождением.

Строки 30–39

Функция `_main()` выполняется, только если этот сценарий не был импортирован для использования в другой программе. Назначение этой функции состоит в том, что она порождает и запускает каждый из потоков. Как было указано выше, применяется метод `atexit.register()` для регистрации функции `_atexit()`, которую интерпретатор может выполнить перед выходом из программы.

В качестве альтернативного по отношению к варианту, в котором поддерживается собственное множество выполняющихся в настоящее время потоков, можно применить вариант с использованием метода `threading.enumerate()`, который возвращает список всех потоков, работающих в настоящее время (этот список включает потоки, действующие в качестве демона, но, безусловно, в него не входят потоки, которые еще не запущены). В рассматриваемом примере этот вариант не используется, поскольку его применение приводит к созданию двух дополнительных потоков, которые придется удалять для сокращения объема вывода, в том числе текущего потока (поскольку он еще не завершен) и основного потока (который так или иначе не должен быть показан).

Не следует также забывать, что вместо оператора форматирования строки можно использовать метод `str.format()`, при условии, что для работы применяется версия Python 2.6 или более новая (включая версии 3.x). Иными словами, следующая инструкция `print`:

```
print "[%s] Started %s" % (ctime(), myname)
```

2.6-2.7 может быть заменена в версии 2.6 и последующей таким вызовом:

```
print "[%0] Started {1}".format(ctime(), myname)
```

3.x или (в версии 3.x) таким вызовом `print()`:

```
print("[%0] Started {1}".format(ctime(), myname))
```

Если задача состоит лишь в том, чтобы подсчитать число потоков, выполняющихся в настоящее время, то вместо этого можно использовать метод `threading.activeCount()` (переименованный в `active_count()`, начиная с версии 2.6).

Применение управления контекстом

2.5 Программисты, работающие в версии Python 2.5 и более новой, могут воспользоваться еще одним вариантом, который вообще не требует вызова методов `acquire()` и `release()` применительно к блокировке, что способствует еще большему упрощению кода. С помощью инструкции `with` можно вводить в действие для любого объекта диспетчер контекста, который обеспечит вызов метода `acquire()` перед входом в критический участок и метода `release()`, когда этот блок завершит выполнение.

Диспетчеры контекста предусмотрены для всех объектов модуля `threading`, таких как `Lock`, `RLock`, `Condition`, `Semaphore` и `BoundedSemaphore`, а это означает, что для работы с этими объектами может применяться инструкция `with`. С помощью инструкции `with` можно еще больше упростить код функции `loop()` следующим образом:

```
from __future__ import with_statement # только в версии 2.5
def loop(nsec):
    myname = currentThread().name
    with lock:
        remaining.add(myname)
        print "[%s] Started %s" % (ctime(), myname)
    sleep(nsec)
    with lock:
        remaining.remove(myname)
        print "[%s] Completed %s (%d secs)" % (
            ctime(), myname, nsec)
        print "    (remaining: %s)" % (
            remaining or 'NONE',)
```

Перенос приложения в версию Python 3

3.x

Теперь можно сравнительно легко перенести приведенный выше сценарий в версию Python 3.x, применив к нему инструмент 2to3 (следующий вывод приведен в сокращенном виде, поскольку полные результаты применения diff уже были показаны ранее):

```
$ 2to3 -w mtsleepF.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
:
RefactoringTool: Files that were modified:
RefactoringTool: mtsleepF.py
```

После переименования `mtsleepF.py` в `mtsleepF3.py` и `mtsleep.py.bak` в `mtsleepF.py` обнаруживается, что этот сценарий относится к той замечательной категории программ, перенос которых в новую версию интерпретатора происходит идеально, без малейших проблем:

```
$ python3 mtsleepF3.py
[Sun Apr  3 23:29:39 2011] Started Thread-1
[Sun Apr  3 23:29:39 2011] Started Thread-2
[Sun Apr  3 23:29:39 2011] Started Thread-3
[Sun Apr  3 23:29:41 2011] Completed Thread-3 (2 secs)
(remaining: Thread-2, Thread-1)
[Sun Apr  3 23:29:42 2011] Completed Thread-2 (3 secs)
(remaining: Thread-1)
[Sun Apr  3 23:29:43 2011] Completed Thread-1 (4 secs)
(remaining: NONE)
all DONE at: Sun Apr  3 23:29:43 2011
```

Теперь, после ознакомления с блокировками, перейдем к изучению семафоров и рассмотрим пример, в котором используется и то и другое.

4.7.4. Пример семафора

Как уже было сказано, блокировки являются довольно простыми для понимания и могут быть легко реализованы. Кроме того, можно довольно легко определить, в каком случае они действительно необходимы. Однако ситуация может оказаться сложнее, и тогда вместо блокировки потребуется более мощный примитив синхронизации. Если в приложении приходится иметь дело с ограниченными ресурсами, то семафоры могут оказаться более приемлемыми.

Семафоры относятся к числу примитивов синхронизации, которые были введены в действие раньше других. Семафор, по существу, представляет собой счетчик, значение которого уменьшается после захвата ресурса (и снова увеличивается после освобождения ресурса). Семафоры, представляющие закрепленные за ними ресурсы, можно рассматривать как доступные или недоступные. Действие по захвату ресурса и уменьшению значения счетчика принято обозначать как `P()` (от голландского слова *probeer/proberen*), но для обозначения этого действия применяются также термины “переход в состояние ожидания”, “осуществление попытки”, “захват”, “приостановка” или “получение”. И наоборот, после завершения работы потока с ресурсом

должен быть произведен возврат ресурса в пул ресурсов. Для этого применяется действие, по традиции обозначаемое `V()` (от голландского слова *verhogen/verhoog*). Это действие обозначается также как “сигнализация”, “наращивание”, “отпускание”, “отправка”, “освобождение”. В языке Python вместо всех этих вариантов именования применяются упрощенные обозначения, согласно которым функции и (или) методы обозначаются как те, что служат для работы с блокировками: `acquire` и `release`. Семафоры являются более гибкими, чем блокировки, поскольку обеспечивают работу с несколькими потоками, в каждом из которых используется один из экземпляров конечного ресурса.

В качестве следующего примера рассмотрим предельно упрощенную модель автомата для торговли конфетами. В данном конкретном автомате имеются в наличии только пять карманов, заполняемых запасом товара (шоколадными батончиками). Если заполнены все карманы, то в автомат больше нельзя заправить ни одной конфеты и, аналогично, при отсутствии в автомате шоколадных батончиков какого-то конкретного типа покупателя, желающие приобрести именно их, будут вынуждены возвратиться с пустыми руками. В данном случае ресурсы (карманы для конфет) являются конечными, и для отслеживания их состояния можно использовать семафор.

Исходный код сценария (`candy.py`) приведен в примере 4.11.

Пример 4.11. Автомат для торговли конфетами, управляемый с помощью семафоров (`candy.py`)

В этом сценарии блокировки и семафоры используются для моделирования автомата для торговли конфетами.

```
1  #!/usr/bin/env python
2
3  from atexit import register
4  from random import randrange
5  from threading import BoundedSemaphore, Lock, Thread
6  from time import sleep, ctime
7
8  lock = Lock()
9  MAX = 5
10 candytray = BoundedSemaphore(MAX)
11
12 def refill():
13     lock.acquire()
14     print 'Refilling candy...',
15     try:
16         candytray.release()
17     except ValueError:
18         print 'full, skipping'
19     else:
20         print 'OK'
21     lock.release()
22
23 def buy():
24     lock.acquire()
25     print 'Buying candy...',
26     if candytray.acquire(False):
27         print 'OK'
28     else:
29         print 'empty, skipping'
```

```
30:     lock.release()
31:
32: def producer(loops):
33:     for i in xrange(loops):
34:         refill()
35:         sleep(randrange(3))
36:
37: def consumer(loops):
38:     for i in xrange(loops):
39:         buy()
40:         sleep(randrange(3))
41:
42: def _main():
43:     print 'starting at:', ctime()
44:     nloops = randrange(2, 6)
45:     print 'THE CANDY MACHINE (full with %d bars!)' % MAX
46:     Thread(target=consumer, args=(randrange(
47:         nloops, nloops+MAX+2),)).start() # покупатель
48:     Thread(target=producer, args=(nloops,)).start() # продавец
49:
50: @register
51: def _atexit():
52:     print 'all DONE at:', ctime()
53:
54: if __name__ == '__main__':
55:     _main()
```

Построчное объяснение

Строки 1–6

Строки запуска и импорта практически полностью совпадают с теми, которые были приведены в предыдущих примерах этой главы. Добавлено лишь объявление семафора. В модуле `threading` предусмотрены два класса семафоров, `Semaphore` и `BoundedSemaphore`. Как уже было сказано, в действительности семафоры — просто счетчики; при запуске семафора задается некоторое постоянное число, которое определяет конечное количество единиц ресурса.

Значение этого счетчика уменьшается на 1 после потребления одной единицы из конечного количества единиц ресурса, а после возврата этой единицы в пул значение счетчика увеличивается. Объект `BoundedSemaphore` предоставляет дополнительную возможность, которая состоит в том, что значение счетчика не может быть увеличено свыше установленного для него начального значения; иными словами, позволяет предотвратить возникновение такой абсурдной ситуации, при которой количество операций освобождения семафора превышает количество операций его захвата.

Строки 8–10

Глобальные переменные в этом сценарии определяют блокировку, константу, представляющую максимальное количество единиц ресурса, которые могут составить потребляемый запас, а также лоток с конфетами.

Строки 12–21

Оператор, обслуживающий эти вымышленные торговые автоматы, время от времени пополняет запасы товара. Для моделирования этого действия применяется функция `refill()`. Вся процедура представляет критический участок кода; именно поэтому захват блокировки становится обязательным условием, при котором могут быть беспрепятственно выполнены все строки от начала до конца. В коде предусмотрен вывод журнала со сведениями о выполненных действиях, с которым может ознакомиться пользователь. Кроме того, вырабатывается предупреждение при попытке превысить максимально допустимый объем запасов (строки 17–18).

Строки 23–30

Функция `buy()` противоположна по своему назначению функции `refill()`; она позволяет покупателю каждый раз получать по одной единице из хранимых запасов. Для обнаружения того, не исчерпаны ли уже конечные ресурсы, применяется условное выражение (строка 26). Значение счетчика ни в коем случае не может стать отрицательным, поэтому после достижения нулевого значения количества запасов вызов функции `buy()` блокируется до того момента, когда значение счетчика будет снова увеличено. Если этой функции передается значение неблокирующего флага, `False`, то вместо блокирования функция возвращает `False`, указывая на то, что ресурсы исчерпаны.

Строки 32–40

Функции `producer()` и `consumer()`, моделирующие пополнение и потребление запаса конфет, просто повторяются в цикле и обеспечивают выполнение соответствующих вызовов функций `refill()` и `buy()`, приостанавливаясь на короткое время между вызовами.

Строки 42–55

В последней части кода сценария содержится вызов функции `_main()`, выполняемый при запуске сценария из командной строки, предусмотрена регистрация функции выхода, а также приведено определение функции `_main()`, в которой предусмотрена инициализация вновь созданной пары потоков, которые моделируют пополнение и потребление конфет.

В коде создания потока, который представляет покупателя (потребителя), дополнительно предусмотрена возможность вводить установленное случайным образом положительное смещение, при котором покупатель фактически может попытаться получить больше шоколадных батончиков, чем заправлено в торговый автомат поставщиком/производителем (в противном случае в коде никогда не возникла бы ситуация, в которой покупателю разрешалось бы совершать попытку купить шоколадный батончик из пустого автомата).

Выполнение сценарием приводит к получению примерно такого вывода:

```
$ python candy.py
starting at: Mon Apr  4 00:56:02 2011
THE CANDY MACHINE (full with 5 bars)!
Buying candy... OK
Refilling candy... OK
Refilling candy... full, skipping
Buying candy... OK
```

```

Buying candy... OK
Refilling candy... OK
Buying candy... OK
Buying candy... OK
Buying candy... OK
all DONE at: Mon Apr 4 00:56:08 2011

```



В процессе отладки этого сценария может потребоваться вмешательство вручную

Если в какой-то момент нужно будет приступить к отладке сценария, в котором используются семафоры, прежде всего необходимо точно знать, какое значение находится в счетчике семафора в любое заданное время. В одном из упражнений в конце данной главы предлагается реализовать такое решение в сценарии `candy.py`, возможно, переименовав его в `candydebug.py`, и предусмотреть в нем возможность отображать значение счетчика. Для этого может потребоваться рассмотреть исходный код модуля `threading.py` (причем, вероятно, и в версии Python 2, и в версии Python 3).

3.x

При этом следует учитывать, что имена примитивов синхронизации модуля `threading` не являются именами классов, даже притом, что в них используется выделение прописными буквами, как в `ВерблюдьемСтиле`, что делает их похожими на классы. В действительности эти примитивы представляют собой всего лишь однострочные функции, применяемые для порождения необходимых объектов. При работе с модулем `threading` необходимо учитывать две сложности: во-первых, невозможно порождать подклассы создаваемых объектов (поскольку они представляют собой функции), а во-вторых, при переходе от версий 2.x к версиям 3.x изменились имена переменных.

Обе эти сложности можно было бы легко преодолеть, если бы в обеих версиях объекты предоставляли свободный и единообразный доступ к счетчику, что в действительности не обеспечивается. Непосредственный доступ к значению счетчика может быть получен, поскольку он представляет собой всего лишь атрибут класса, но, как уже было сказано, имя переменной `self._value` изменилось. Это означает, что переменная `self._Semaphore__value` в Python 2 была переименована в `self._value` в Python 3.

Что касается разработчиков, то наиболее удобный вариант применения интерфейса прикладного программирования API (по крайней мере, по мнению автора) состоит в создании подкласса класса `threading._BoundedSemaphore` и реализации метода `__len__()`. Но если планируется поддерживать сценарий и в версии 2.x, и в версии 3.x, то следует использовать правильное значение счетчика, как было только что описано.

Перенос приложения в версию Python 3

По аналогии с `mtsleef.py`, `candy.py` представляет собой еще один пример того, что достаточно применить инструмент `2to3`, чтобы сформировать работоспособную версию для Python 3 (для которой должно использоваться имя `candy3.py`). Оставляем проверку этого утверждения в качестве упражнения для читателя.

Резюме

В настоящей главе было продемонстрировано использование только некоторых примитивов синхронизации, которые входят в состав модуля `threading`. Поэтому при желании читатель может найти для себя широкие возможности по исследованию этой тематики. Однако следует учитывать, что объекты, представленные в

указанном модуле, полностью соответствуют своему определению, т.е. являются примитивами. Это означает, что разработчик вполне может заняться созданием на их основе собственных классов и структур данных, обеспечив, в частности, их потоковую безопасность. Для этого в стандартной библиотеке Python предусмотрен еще один объект, Queue.

4.8. Проблема “производитель–потребитель” и модуль Queue/queue

В заключительном примере иллюстрируется принцип работы “производитель–потребитель”, согласно которому производитель товаров или поставщик услуг производит товары или подготавливает услуги и размещает их в структуре данных наподобие очереди. Интервалы между отдельными событиями передачи произведенных товаров в очередь не детерминированы, как и интервалы потребления товаров.

3.x

Модуль Queue (который носит это имя в версии Python 2.x, но переименован в queue в версии 3.x) предоставляет механизм связи между потоками, с помощью которого отдельные потоки могут совместно использовать данные. В данном случае создается очередь, в которую производитель (один поток) помещает новые товары, а потребитель (другой поток) их расходует. В табл. 4.5 перечислены различные атрибуты, которые представлены в указанном модуле.

Таблица 4.5. Общие атрибуты модуля Queue/queue

Атрибут	Описание
Классы модуля Queue/queue	
Queue (maxsize=0)	Создает очередь с последовательной организацией, имеющую указанный размер <i>maxsize</i> , которая не позволяет вставлять новые блоки после достижения этого размера. Если размер не указан, то длина очереди становится неограниченной
LifoQueue (maxsize=0)	Создает стек, имеющий указанный размер <i>maxsize</i> , который не позволяет вставлять новые блоки после достижения этого размера. Если размер не указан, то длина стека становится неограниченной
PriorityQueue (maxsize=0)	Создает очередь по приоритету, имеющую указанный размер <i>maxsize</i> , которая не позволяет вставлять новые блоки после достижения этого размера. Если размер не указан, то длина очереди становится неограниченной
Исключения модуля Queue/queue	
Empty	Активируется при вызове метода <code>get*()</code> применительно к пустой очереди
Full	Активируется при вызове метода <code>put*()</code> применительно к заполненной очереди
Методы объекта Queue/queue	
qsize()	Возвращает размер очереди (это — приблизительное значение, поскольку при выполнении этого метода может происходить обновление очереди другими потоками)

Атрибут	Описание
<code>empty()</code>	Возвращает <code>True</code> , если очередь пуста; в противном случае возвращает <code>False</code>
<code>full()</code>	Возвращает <code>True</code> , если очередь заполнена; в противном случае возвращает <code>False</code>
<code>put(item, block=True, timeout=None)</code>	Помещает элемент <code>item</code> в очередь; если значение <code>block</code> равно <code>True</code> (по умолчанию) и значение <code>timeout</code> равно <code>None</code> , устанавливает блокировку до тех пор, пока в очереди не появится свободное место. Если значение <code>timeout</code> является положительным, блокирует очередь самое больше на <code>timeout</code> секунд, а если значение <code>block</code> равно <code>False</code> , активизирует исключение <code>Empty</code>
<code>put_nowait(item)</code>	То же, что и <code>put(item, False)</code>
<code>get(block=True, timeout=None)</code>	Получает элемент из очереди, если задано значение <code>block</code> (отличное от 0); устанавливает блокировку до того времени, пока элемент не станет доступным
<code>get_nowait()</code>	То же, что и <code>get(False)</code>
<code>task_done()</code>	Используется для указания на то, что работа по постановке элемента в очередь завершена, в сочетании с описанным ниже методом <code>join()</code>
<code>join()</code>	Устанавливает блокировку до того времени, пока не будут обработаны все элементы в очереди; сигнал об этом вырабатывается путем вызова описанного выше метода <code>task_done()</code>

Для демонстрации того, как реализуется принцип “производитель–потребитель” с помощью модуля `Queue/queue`, воспользуемся примером 4.12 (`prodcons.py`). Ниже приведен вывод, полученный при одном запуске на выполнение этого сценария.

```
$ prodcons.py
starting writer at: Sun Jun 18 20:27:07 2006
producing object for Q... size now 1
starting reader at: Sun Jun 18 20:27:07 2006
consumed object from Q... size now 0
producing object for Q... size now 1
consumed object from Q... size now 0
producing object for Q... size now 1
producing object for Q... size now 2
producing object for Q... size now 3
consumed object from Q... size now 2
consumed object from Q... size now 1
writer finished at: Sun Jun 18 20:27:17 2006
consumed object from Q... size now 0
reader finished at: Sun Jun 18 20:27:25 2006
all DONE
```

Пример 4.12. Пример реализации принципа “производитель–потребитель” (`prodcons.py`)

В этой реализации принципа “производитель–потребитель” используются объекты `Queue` и вырабатывается случайным образом количество произведенных (и потребленных) товаров. Производитель и потребитель моделируются с помощью потоков, действующих отдельно и одновременно.

```
1  #!/usr/bin/env python
2
3  from random import randint
4  from time import sleep
5  from Queue import Queue
6  from myThread import MyThread
7
8  def writeQ(queue):
9      print 'producing object for Q...',
10     queue.put('xxx', 1)
11     print "size now", queue.qsize()
12
13  def readQ(queue):
14     val = queue.get(1)
15     print 'consumed object from Q... size now', \
16         queue.qsize()
17
18  def writer(queue, loops):
19     for i in range(loops):
20         writeQ(queue)
21         sleep(randint(1, 3))
22
23  def reader(queue, loops):
24     for i in range(loops):
25         readQ(queue)
26         sleep(randint(2, 5))
27
28  funcs = [writer, reader]
29  nfuncs = range(len(funcs))
30
31  def main():
32     nloops = randint(2, 5)
33     q = Queue(32)
34
35     threads = []
36     for i in nfuncs:
37         t = MyThread(funcs[i], (q, nloops),
38             funcs[i].__name__)
39         threads.append(t)
40
41     for i in nfuncs:
42         threads[i].start()
43
44     for i in nfuncs:
45         threads[i].join()
46
47     print 'all DONE'
48
49  if __name__ == '__main__':
50     main()
```

Вполне очевидно, что операции, выполняемые производителем и потребителем, не всегда чередуются, поскольку образуют две независимые последовательности. (Весьма удачно то, что в нашем распоряжении имеется готовый механизм выработки случайных чисел!) Если же говорить серьезно, то события, происходящие в действительности, как правило, подчиняются законам случайности и недетерминированы.

Построчное объяснение

Строки 1–6

В этом модуле используется объект `Queue.Queue`, а также потоки, сформированные с помощью класса `myThread.MyThread`, как было описано ранее. Метод `random.randint()` применяется для внесения элемента случайности в операции производства и потребления. (Заслуживает внимания то, что метод `random.randint()` действует точно так же, как и метод `random.randrange()`, но предусматривает включение в интервал вырабатываемых случайных чисел начального и конечного значений.)

Строки 8–16

Функции `writeQ()` и `readQ()` выполняют следующие операции: первая из них помещает объект в очередь (в качестве объекта может использоваться, например, строка `'xxx'`), а вторая извлекает объект из очереди. Следует учитывать, что операции постановки в очередь и изъятия из очереди осуществляются одновременно по отношению только к одному объекту.

Строки 18–26

Метод `writer()` выполняется как отдельный поток, единственным назначением которого является выработка одного элемента для постановки в очередь, переход на время в состояние ожидания, а затем повтор этого цикла указанное количество раз, причем количество повторов устанавливается при выполнении сценария случайным образом. Метод `reader()` действует аналогично, если не считать того, что он не ставит, а извлекает элементы из очереди.

Необходимо отметить, что устанавливаемая случайным образом продолжительность приостановки метода-производителя в секундах, как правило, меньше по сравнению с той продолжительностью, на которую приостанавливается метод-потребитель. Это сделано для того, чтобы метод-потребитель не мог предпринять попытки извлечения элементов из пустой очереди. Сокращение продолжительности приостановки метода-производителя способствует повышению вероятности того, что в распоряжении метода-потребителя всегда будет пригодный для извлечения элемент, когда настанет время очередного выполнения этой операции.

Строки 28–29

Это всего лишь подготовительные строки, с помощью которых задается общее количество потоков, подлежащих порождению и запуску.

Строки 31–47

Наконец, предусмотрена функция `main()`, которая должна выглядеть весьма подобной функциям `main()` из всех прочих сценариев, приведенных в этой главе. Создаются необходимые потоки и осуществляется их запуск, а окончание работы наступает после того, как оба потока завершают свое выполнение.

На основании этого примера можно сделать вывод, что программа, предназначенная для выполнения нескольких задач, может быть организована так, чтобы для реализации каждой из задач применялись отдельные потоки. Результатом может стать получение гораздо более наглядного проекта программы по сравнению с однопоточной программой, в которой предпринимается попытка обеспечить выполнение всех задач.

В настоящей главе было показано, что применение однопоточного процесса может стать препятствием к повышению производительности приложения. Особенно значительно может быть повышена производительность программ, основанных на последовательном выполнении независимых, недетерминированных и не имеющих причинных зависимостей задач, в результате их разбиения на отдельные задачи, выполняемые отдельными потоками. Существенный выигрыш от перехода к многопоточной обработке может быть достигнут не во всех приложениях. Причинами этого могут стать дополнительные издержки, а также тот факт, что сам интерпретатор Python представляет собой однопоточное приложение. Тем не менее овладение функциональными возможностями многопоточной организации Python позволяет взять этот инструмент на вооружение, когда это оправдано.

4.9. Дополнительные сведения об использовании потоков

Прежде чем приступить к повсеместному применению средств поддержки многопоточности, следует провести краткий обзор особенностей такой организации программирования. Вообще говоря, применение нескольких потоков в программе может способствовать ее улучшению. Однако в интерпретаторе Python применяется глобальная блокировка, которая накладывает свои ограничения, поэтому многопоточная организация является более подходящей для приложений, ограничиваемых пропускной способностью ввода-вывода (при вводе-выводе происходит освобождение глобальной блокировки интерпретатора, что способствует повышению степени распараллеливания), а не приложений, ограничиваемых пропускной способностью процессора. В последнем случае для достижения более высокой степени распараллеливания необходимо иметь возможность параллельного выполнения процессов несколькими ядрами или процессорами.

Не вдаваясь в дополнительные подробности (поскольку некоторые из соответствующих тем уже рассматривались в главе “Среда выполнения” книги “Core Python Programming” или “Core Python Language Fundamentals”), перечислим основные альтернативы модулю `threading`, касающиеся поддержки нескольких потоков или процессов.

4.9.1. Модуль `subprocess`

2.4

В первую очередь вместо модуля `threading` можно применить модуль `subprocess`, когда возникает необходимость запуска новых процессов, либо для выполнения кода, либо для обеспечения обмена данными с другими процессами через стандартные файлы ввода-вывода (`stdin`, `stdout`, `stderr`). Этот модуль был введен в версии Python 2.4.

4.9.2. Модуль `multiprocessing`

2.6

Этот модуль, впервые введенный в Python 2.6, позволяет запускать процессы для нескольких ядер или процессоров, но с интерфейсом, весьма напоминающим интерфейс модуля `threading`. Он также поддерживает различные механизмы передачи данных между процессами, применяемыми для выполнения совместной работы.

4.9.3. Модуль `concurrent.futures`

3.2

Это новая высокоуровневая библиотека, которая работает только на уровне заданий. Это означает, что при использовании модуля `concurrent.futures` исключается необходимость заботиться о синхронизации либо управлять потоками или процессами. Достаточно лишь указать поток или пул процесса с определенным количеством рабочих потоков, передать задания на выполнение и собрать полученные результаты. Этот модуль впервые появился в версии Python 3.2, но перенесен также в версию Python 2.6 и последующие версии. Модуль можно получить по адресу <http://code.google.com/p/pythonfutures>.

Рассмотрим вариант сценария `bookrank3.py` с указанными изменениями. При условии, что все прочее остается таким, как прежде, рассмотрим новые операторы импорта и изменившуюся часть сценария `_main()`:

```
from concurrent.futures import ThreadPoolExecutor
...
def _main():
    print('At', ctime(), 'on Amazon...')
    with ThreadPoolExecutor(3) as executor:
        for isbn in ISBNs:
            executor.submit(_showRanking, isbn)
    print('all DONE at:', ctime())
```

Методу `concurrent.futures.ThreadPoolExecutor` передается параметр, представляющий собой размер пула потоков, а приложение применяется для определения рангов трех книг. Безусловно, это — приложение, ограничиваемое пропускной способностью ввода-вывода, для которого применение потоков оказывает наибольшую пользу. Что касается приложений, ограничиваемых пропускной способностью процессора, то вместо указанного метода целесообразно было бы использовать `concurrent.futures.ProcessPoolExecutor`.

После создания управляющего объекта (действие которого распространяется на потоки или процессы), отвечающего за планирование заданий и сбор результатов, можно вызвать его метод `submit()` для выполнения намеченной ранее задачи порождения потока.

После полного переноса в версию Python 3 путем замены оператора форматирования строки методом `str.format()`, повсеместного введения инструкции `with` и использования метода `map()` управляющего объекта появляется возможность полностью удалить метод `_showRanking()` и передать его функции в программу `_main()`. Заключительная версия сценария `bookrank3CF.py` приведена в примере 4.13.

Пример 4.13. Применение средств управления заданиями высокого уровня (`bookrank3CF.py`)

В этом участке кода, как и в предыдущих примерах, осуществляется сбор с экрана данных о рангах книг, но на этот раз с помощью модуля `concurrent.futures`.

```
1  #!/usr/bin/env python
2
3  from concurrent.futures import ThreadPoolExecutor
4  from re import compile
```

```

5  from time import ctime
6  from urllib.request import urlopen as uopen
7
8  REGEX = compile(b'#[([\\d,]+) in Books ')
9  AMZN = 'http://amazon.com/dp/'
10 ISBNs = {
11      '0132269937': 'Core Python Programming',
12      '0132356139': 'Python Web Development with Django',
13      '0137143419': 'Python Fundamentals',
14  }
15
16  def getRanking(isbn):
17      with uopen('{0}{1}'.format(AMZN, isbn)) as page:
18          return str(REGEX.findall(page.read())[0], 'utf-8')
19
20  def _main():
21      print('At', ctime(), 'on Amazon...')
22      with ThreadPoolExecutor(3) as executor:
23          for isbn, ranking in zip(
24              ISBNs, executor.map(getRanking, ISBNs)):
25              print('- %r ranked %s' % (ISBNs[isbn], ranking))
26      print('all DONE at:', ctime())
27
28  if __name__ == '__main__':
29      main()

```

Построчное объяснение

Строки 1–14

Если не считать новой инструкции **import**, то вся первая половина этого сценария полностью идентична той, что приведена в файле `bookrank3.py`, который рассматривался выше в главе.

Строки 16–18

В новой функции `getRanking()` используются инструкция **with** и функция `str.format()`. Аналогичные изменения можно внести в сценарий `bookrank.py`, поскольку оба указанных средства доступны также в версии 2.6 и последующих (а не предусмотрены исключительно в версиях 3.x).

Строки 20–26

В предыдущем примере кода использовался метод `executor.submit()` для формирования заданий. В данном примере предусмотрены некоторые изменения в связи с использованием метода `executor.map()`, поскольку он позволяет реализовать функции из `_showRanking()` и полностью исключить их поддержку из нашего кода.

Полученный вывод почти аналогичен тому, который рассматривался ранее:

```

$ python3 bookrank3CF.py
At Wed Apr 6 00:21:50 2011 on Amazon...
- 'Core Python Programming' ranked 43,992
- 'Python Fundamentals' ranked 1,018,454
- 'Python Web Development with Django' ranked 502,566
all DONE at: Wed Apr 6 00:21:55 2011

```

Дополнительные сведения об истории создания модуля `concurrent.futures` можно найти с помощью приведенных ниже ссылок.

<http://docs.python.org/dev/py3k/library/concurrent.futures.html>
<http://code.google.com/p/pythonfutures/>
<http://www.python.org/dev/peps/pep-3148/>

Краткое описание этих параметров, а также другая информация, касающаяся модулей и пакетов для многопоточной организации программы, приведена в следующем разделе.

4.10. Связанные модули

В табл. 4.6 перечислены некоторые модули, которые можно использовать при программировании многопоточных приложений.

Таблица 4.6. Стандартные библиотечные модули, связанные с многопоточной поддержкой

Модуль	Описание
<code>thread</code> ^a	Основной, находящийся на низком уровне модуль поддержки потоков
<code>threading</code>	Объекты для многопоточной организации работы и синхронизации высокого уровня
<code>multiprocessing</code> ^b	Запуск/использование подпроцессов с помощью интерфейса <code>threading</code>
<code>subprocess</code> ^c	Полный отказ от потоков и выполнение вместо них процессов
<code>Queue</code>	Синхронизированная очередь с последовательной организацией для нескольких потоков
<code>mutex</code> ^d	Объекты мьютексов
<code>concurrent.futures</code> ^e	Библиотека высокого уровня для асинхронного выполнения
<code>SocketServer</code>	Создание/управление многопоточными серверами TCP или UDP

^a Переименован в `_thread` в Python 3.0.

^b Новое в версии Python 2.6.

^c Новое в версии Python 2.4.

^d Обозначен как устаревший в Python 2.6 и удален в версии 3.0.

^e Впервые введенный в Python 3.2, но предоставляемый вне стандартной библиотеки для версии 2.6 и последующих версий.

4.11. Упражнения

- 4.1. *Сопоставление процессов с потоками.* В чем состоят различия между процессами и потоками?
- 4.2. *Потоки Python.* Какие типы многопоточных приложений обеспечивают наибольшую производительность в Python, ограничиваемые пропускной способностью ввода-вывода или пропускной способностью процессора?
- 4.3. *Потоки.* Какие наиболее заметные отличия будут обнаружены после запуска многопоточного приложения в системе не с одним, а с несколькими процессорами? Как, по вашему мнению, будут выполняться несколько потоков в этих системах?

4.4. *Потоки и файлы.*

- а) Создайте функцию, которая получает байтовое значение и имя файла (в виде параметров или данных, введенных пользователем) и определяет, сколько раз байтовое значение появляется в файле.
- б) Теперь предположим, что входной файл является чрезвычайно большим. Допускается применение для чтения файла одновременно нескольких функций, поэтому измените полученное ранее решение в целях создания многочисленных потоков, обрабатывающих разные части файла, чтобы каждый поток отвечал лишь за определенную часть файла. Соберите данные, полученные каждым потоком, и рассчитайте суммарное значение. Воспользуйтесь модулем `timeit` для измерения продолжительности работы обоих решений, однопоточного и многопоточного, и прокомментируйте разницу в производительности, если таковая будет обнаружена.

4.5. *Потоки, файлы и регулярные выражения.* Для выполнения этого задания требуется очень большой файл почтового ящика. Если таковой отсутствует, соедините все свои сообщения электронной почты в общий текстовый файл. Задание заключается в следующем. Воспользуйтесь регулярными выражениями, предназначенными для распознавания адресов электронной почты и URL веб-сайтов, которые рассматривались ранее в этой книге, для преобразования всех адресов электронной почты и URL из указанного большого файла в актуальные ссылки. Эти ссылки необходимо сохранить в отдельном файле с расширением `.html` (или `.htm`), который можно открыть в веб-браузере для получения возможности переходить по ним с помощью щелчка мышью. Используйте потоки для проведения процесса преобразования одновременно по всему большому текстовому файлу, после чего соберите полученные результаты в отдельный новый файл `.html`. Откройте этот файл в веб-браузере, чтобы проверить, действительно ли работают ссылки.

4.6. *Потоки и сети.* В приложении службы интерактивной переписки, разработанном в качестве упражнения в предыдущей главе, требовалось использование в составе решения так называемых тяжеловесных потоков, или процессов. Преобразуйте это решение в многопоточное.

4.7. **Потоки и программирование для веб.* Приложение `Crawler`, которое представлено в главе 10 “Программирование для веб. CGI и WSGI”, является однопоточным приложением, предназначенным для загрузки веб-страниц. Его усовершенствованию способствовало бы применение многопоточного программирования. Обновите сценарий `crawl.py` (переименовав его в `mtcrawl.py`), чтобы для загрузки страниц использовались независимые потоки. Обязательно воспользуйтесь механизмом блокировки того или иного типа для предотвращения конфликтов доступа к очереди ссылок.

4.8. *Пулы потоков.* Внесите изменения в код сценария `prodcons.py`, который рассматривается в примере 4.12, чтобы в нем вместо потока производителя и одного потока потребителя могло применяться любое количество потоков потребителя (пул потоков) для обработки или выборки в любой момент времени нескольких элементов из очереди `Queue`.

4.9. *Файлы.* Создайте ряд потоков для подсчета количества строк в наборе текстовых файлов (который может иметь большой суммарный объем). Может

быть предусмотрена возможность выбирать количество используемых потоков. Сравните производительность многопоточной и однопоточной версий этого кода. Совет. Ознакомьтесь с упражнениями в конце главы 9 в книге *Core Python Programming* или *Core Python Language Fundamentals*.

- 4.10. *Параллельная обработка.* Возьмите за основу свое решение упражнения 4.9 и примите к выбранной вами произвольной задаче, такой как обработка набора сообщений электронной почты, загрузка веб-страниц, обработка веб-каналов RSS или Atom, усовершенствование обработки сообщений сервером интерактивной переписки, поиск решения головоломки и т.д.
- 4.11. *Примитивы синхронизации.* Изучите каждый из примитивов синхронизации в модуле `threading`. Опишите их назначение, укажите возможную область их применения и подготовьте примеры рабочего кода для каждого из них.

Следующий ряд упражнений касается сценария `candy.py`, который рассматривался в примере 4.11.

- 4.12. *Перенос в версию Python 3.* Возьмите за основу сценарий `candy.py` и примените к нему инструмент `2to3` для создания версии Python 3 с именем `candy3.py`.
- 4.13. *Модуль `threading`.* Добавьте к сценарию средства отладки. В частности, для приложений, в которых используются семафоры (с начальным значением, как правило, больше 1), можно предусмотреть точное определение значения счетчика в любое конкретное время. Подготовьте вариант сценария `candy.py` (который можно назвать `candydebug.py`) и реализуйте в нем возможность отображать значение счетчика. Вам может потребоваться изучить исходный код сценария `threading.py`, как было указано выше в одном из советов. После внесения этих изменений можно откорректировать вывод сценария, чтобы он выглядел примерно так:

```
$ python candydebug.py
starting at: Mon Apr  4 00:24:28 2011
THE CANDY MACHINE (full with 5 bars)!
Buying candy... inventory: 4
Refilling candy... inventory: 5
Refilling candy... full, skipping
Buying candy... inventory: 4
Buying candy... inventory: 3
Refilling candy... inventory: 4
Buying candy... inventory: 3
Buying candy... inventory: 2
Buying candy... inventory: 1
Buying candy... inventory: 0
Buying candy... empty, skipping
all DONE at: Mon Apr  4 00:24:36 2011
```

Программирование графического пользовательского интерфейса

В этой главе...

- Введение
- Библиотека Tkinter и программирование на языке Python
- Примеры Tkinter
- Краткий обзор других графических пользовательских интерфейсов
- Связанные модули и другие графические пользовательские интерфейсы

Тематику, связанную с графическим пользовательским интерфейсом, принято считать сложной. Ее изучение закаляет характер.
Джим Альстром (Jim Ahlstrom), май 1995 г.
(из выступления на практикуме Python)

В настоящей главе приведено краткое введение в программирование графического пользовательского интерфейса (graphical user interface — GUI). Эта глава предназначена для читателей, которые недостаточно хорошо знакомы с этой областью и хотят узнать о ней больше, а также для тех, кто желает ознакомиться с реализацией средств графического пользовательского интерфейса в языке Python. В одной главе невозможно описать все аспекты разработки приложений с графическим пользовательским интерфейсом, но здесь, по крайней мере, приведено достаточно полное введение в эту проблематику. В качестве основного набора инструментов будет применяться Tk — предусмотренный по умолчанию графический пользовательский интерфейс Python. Доступ к интерфейсу Tk будет осуществляться из интерфейса Python, именуемого Tkinter (сокращение от “Tk interface”).

Tk — это не самый новый, не самый лучший и даже не самый надежный набор стандартных блоков графического пользовательского интерфейса, но он достаточно прост в применении, и с его помощью могут создаваться графические пользовательские интерфейсы, работающие на большинстве платформ. В главе представлено несколько примеров использования Tkinter небольшой и средней сложности, а затем приводятся несколько примеров работы с другими наборами инструментов. После изучения данной главы читатель получит достаточную подготовку для создания более сложных приложений и/или для перехода к более современному набору инструментов. В языке Python предусмотрены привязки или адаптеры для большинства современных крупных наборов инструментов, включая коммерческие системы.

5.1. Введение

Прежде чем приступить к изучению программирования графического пользовательского интерфейса, вначале рассмотрим Tkinter как предусмотренный по умолчанию набор инструментов пользовательского интерфейса Python. Первым делом необходимо ознакомиться с его инсталляцией, поскольку интерфейс Tkinter не всегда устанавливается по умолчанию (особенно если пользователь занимается построением интерпретатора Python самостоятельно). Вслед за этим будет дан краткий обзор вопросов архитектуры “клиент–сервер”. Эта архитектура впервые была представлена в главе 2, но нам потребуются дополнительные сведения, относящиеся к рассматриваемому предмету.

5.1.1. Что такое Tcl, Tk и Tkinter

Tkinter — это предусмотренная по умолчанию библиотека графического пользовательского интерфейса Python. Она основана на наборе инструментов Tk, первоначально разработанного для языка Tcl (Tool Command Language). Набор Tk получил широкую известность, поэтому был перенесен во многие другие языки сценариев, включая Perl (Perl/Tk), Ruby (Ruby/Tk) и Python (Tkinter). Средства разработки графического пользовательского интерфейса с применением Tk являются переносимыми и гибкими, а также обеспечивают возможность применения простого языка сценариев.

Эти положительные качества в сочетании с преимуществами системных языков позволили создать инструменты для быстрой разработки и реализации широкого набора приложений с графическим пользовательским интерфейсом, обладающих качеством на уровне коммерческой системы.

Читатели, впервые изучающие программирование графического пользовательского интерфейса, будут приятно удивлены тем, насколько просто решается эта задача. Они легко убедятся в том, что язык Python, наряду со средствами Tkinter, предоставляет быстрый и увлекательный способ построения приложений, которые могут оказаться интересными (и часто даже полезными), причем сама эта работа потребовала бы гораздо больше времени, если бы пришлось программировать непосредственно на языке C/C++ с применением собственных библиотек системы управления окнами. Разрабатывая приложение и его интерфейс, программист обычно использует стандартные конструкции, известные как *графические элементы*, собирая требуемые компоненты вместе и наделяя их функциями.

Программисты, хорошо знакомые с использованием инструментов Tk, будь то в сочетании с языком Tcl или Perl, обнаружат, что язык Python предоставляет гораздо более удобный способ разработки графического пользовательского интерфейса. Кроме того, в этом языке предусмотрена такая система создания прототипов, которая позволяет быстрее решать эту задачу по сравнению с другими языками. Следует также учитывать, что при этом возникает возможность воспользоваться такими преимуществами Python, как доступность системы, сетевые функциональные средства, средства поддержки языка XML, числовой и визуальной обработки, доступа к базам данных, а также всеми прочими стандартными библиотечными и сторонними модулями расширения.

После установки библиотеки Tkinter в системе потребуется не больше 15 минут на то, чтобы создать свое первое работающее приложение с графическим пользовательским интерфейсом.

5.1.2. Установка и ввод в действие интерфейса Tkinter

3.x

Интерфейс Tkinter не всегда бывает включен в систему по умолчанию. Чтобы определить, доступен ли Tkinter для интерпретатора Python, можно попытаться импортировать модуль Tkinter (такое имя применяется в версиях Python 1 и 2; в версии Python 3 он переименован в `tkinter`). Если интерфейс Tkinter доступен, то ошибка не возникает, как показывает следующий пример:

```
>>> import Tkinter
>>>
```

Если же применяемый интерпретатор Python не был откомпилирован с включенной поддержкой интерфейса Tkinter, то операция импорта модуля закончится неудачей:

```
>>> import Tkinter
Traceback (innermost last):
  File "<stdin>", line 1, in ?
  File "/usr/lib/pythonX.Y/lib-tk/Tkinter.py", line 8, in ?
    import _tkinter # Если эта операция оканчивается неудачей, то, возможно,
интерпретатор Python не настроен для работы с Tk
ImportError: No module named _tkinter
```

Для того чтобы получить доступ к библиотеке Tkinter может потребоваться повторная компиляция интерпретатора Python. Для этого обычно достаточно отредактировать файл Modules/Setup, а затем правильно задать все параметры, необходимые для компиляции интерпретатора Python с привязками к библиотеке Tkinter, или установить в системе инструментарий Tk. С конкретными указаниями по компиляции библиотеки Tkinter можно ознакомиться в файле README, входящем в состав дистрибутива Python. После компиляции интерпретатора необходимо запустить *новый* интерпретатор Python, иначе интерпретатор будет действовать, как и прежде, без поддержки библиотеки Tkinter (поскольку это и будет *старая* версия интерпретатора).

5.1.3. Архитектура “клиент–сервер” — два компонента

Впервые концепция вычислений в среде “клиент–сервер” была представлена в главе 2. Еще одним примером программного сервера является система управления окнами. Такие системы работают на компьютерах с подключенными устройствами отображения, такими как монитор. В таких системах есть и клиенты — программы, которым для выполнения требуется оконная среда. Они называются *приложениями с графическим пользовательским интерфейсом*. Такие приложения не могут функционировать без системы окон.

Если же приложение работает в сети, то его архитектура становится еще более сложной. Обычно приложение с графическим пользовательским интерфейсом выводит данные на дисплей компьютера, на котором оно запущено (через оконный сервер), но в некоторых оконных сетевых средах, таких как оконная система X Window в операционной системе Unix, оконный сервер можно применять на другом компьютере. Это позволяет выполнять программу с графическим пользовательским интерфейсом на одном компьютере, а выводить изображение на другом.

5.2. Библиотека Tkinter и программирование на языке Python

В данном разделе рассматриваются общие вопросы программирования графического пользовательского интерфейса, а затем описываются способы использования библиотеки Tkinter и ее компонентов для построения графических пользовательских интерфейсов в программах на языке Python.

5.2.1. Модуль Tkinter, обеспечивающий реализацию интерфейса Tk в приложениях

Рассмотрим, что требуется для включения библиотеки Tkinter в состав приложения. Прежде всего следует отметить, что приступить к построению интерфейса на основе библиотеки Tkinter можно, еще не имея готового приложения. По желанию можно создать ничем не заполненный графический пользовательский интерфейс. Однако очевидно, что вряд ли от этого интерфейса будет какая-то польза, если он не поддерживает некоторую программу, выполняющую полезные и интересные действия.

Вообще говоря, для получения полноценного графического пользовательского интерфейса необходимо сделать пять основных шагов.

1. Импортировать модуль Tkinter (для этого применяется конструкция `from Tkinter import *`).
2. Создать объект для работы с окнами верхнего уровня, который содержит все приложение с графическим пользовательским интерфейсом.
3. Сформировать все компоненты графического пользовательского интерфейса (вместе с функциональными средствами) в качестве надстройки (или вложения) к объекту работы с окнами верхнего уровня.
4. Подключить эти компоненты графического пользовательского интерфейса к основному коду приложения.
5. Войти в основной цикл обработки событий.

Первый шаг является весьма несложным, поскольку все графические пользовательские интерфейсы, в которых используется библиотека Tkinter, должны импортировать модуль Tkinter. Прежде всего необходимо получить доступ к библиотеке Tkinter (см. раздел 5.1.2).

5.2.2. Введение в программирование графического пользовательского интерфейса

Прежде чем перейти к изучению примеров, рассмотрим краткое введение в разработку приложений с графическим пользовательским интерфейсом. Эти сведения представляют собой основную совокупность знаний, без которых невозможно следовать дальше.

Разработку приложения с графическим пользовательским интерфейсом можно сравнить с тем, как художник пишет картину на холсте. Работа начинается “с чистого листа” — окна верхнего уровня, на базе которого создается остальная часть применяемых компонентов. Такой объект можно рассматривать как аналог фундамента для дома или мольберта для художника. Иными словами, необходимо залить бетон или установить мольберт, прежде чем приступить к возведению здания или к грунтовке холста. В библиотеке Tkinter такой фундамент именуется *окном верхнего уровня*.

Окна и графические элементы

В программировании графического пользовательского интерфейса все мелкие объекты для работы с окнами, входящие в состав полного приложения, содержатся в корневом оконном объекте верхнего уровня. Этими объектами могут быть текстовые метки, кнопки, списки и т.д. Сами эти отдельные компоненты графического пользовательского интерфейса принято называть *графическими элементами* (widgets). Таким образом, если речь идет о создании окна верхнего уровня, под этим просто подразумевается предоставление такого места, куда будут помещены все применяемые графические элементы. В языке Python для этого применяется примерно такая конструкция:

```
top = Tkinter.Tk() # или просто Tk() после вызова "from Tkinter import *"
```

Объект, возвращаемый функцией `Tkinter.Tk()`, обычно именуется *корневым окном*. Вот почему в некоторых приложениях для указания на такое окно принято применять обозначение `root` (корень), а не `top` (вершина). Окнами *верхнего уровня* являются такие окна, которые отображаются в составе приложения автономно.

В графическом интерфейсе пользователя можно иметь несколько окон верхнего уровня, но только одно из них должно быть корневым. Разработку приложения можно осуществлять по такому принципу — вначале полностью спроектировать все графические элементы, а затем добавить реальные функциональные средства. Может быть принят также подход, предусматривающий постепенное развитие приложения, при котором по ходу дела осуществляется реализация необходимых функций одной за другой. (Иными словами, в одном случае акцент делается на компоновке, а в другом — на согласовании (шаги 3 и 4, соответственно, из приведенного выше списка)).

Графические элементы могут быть автономными или играть роль контейнеров. Если графический элемент содержит другие графические элементы, то он рассматривается как *родительский* по отношению к ним. Соответственно, если графический элемент содержится в другом графическом элементе, то он рассматривается как *дочерний* относительно контейнера, который содержит его непосредственно.

Обычно графические элементы предусматривают определенное поведение пользователя. Например, если это кнопка, то ее можно нажать, в поле редактирования можно ввести текст, а список можно развернуть. Такие действия пользователя называются *событиями* (*events*), а реакция графического пользовательского интерфейса на такие события называется *обратным вызовом* (*callback*).

Обработка, осуществляемая как реакция на события

Событиями могут быть нажатие (и отпускание) кнопки мыши, перемещение курсора мыши, нажатие клавиши <Enter> и т.д. Выполнение приложения с графическим пользовательским интерфейсом происходит под управлением всей последовательности событий, происходящих от начала и до конца работы приложения. Такое поведение системы называется *событийно-управляемым* (*event-driven processing*).

Одним из примеров события с обратным вызовом может служить перемещение курсора мыши. Предположим, что курсор находится в одном из окон верхнего уровня. Если пользователь перемещает курсор, скажем, в другую часть приложения, то программа воплощает это действие в движение курсора на экране, так что создается впечатление, будто перемещение осуществляется самим движением руки. В этом и состоят события перемещения курсора мыши, которые система должна обрабатывать, создавая изображение курсора, движущегося в окне. Когда пользователь отпустит кнопку мыши, события, подлежащие обработке, перестанут формироваться, поэтому изображение на экране вновь станет статичным.

Такой способ организации графического пользовательского интерфейса, управляемого событиями, вполне может быть реализован с помощью архитектуры “клиент–сервер”. При запуске приложения с графическим пользовательским интерфейсом выполняются предварительные процедуры настройки, аналогично тому, как сетевой сервер выделяет сокет и привязывает его к локальному адресу. Приложение должно определить все компоненты графического пользовательского интерфейса, а затем отобразить их на экране (это действие называют также *прорисовкой* (*rendering*) или *рисованием* (*painting*)). Для решения этой задачи применяется так называемый *диспетчер геометрии* (дополнительные сведения о нем приведены ниже). Когда диспетчер геометрии завершает расстановку всех графических элементов, включая окно верхнего уровня, приложения с графическим пользовательским интерфейсом входят в бесконечный цикл, подобный тому, в котором работает сетевой сервер. В процессе бесконечного повторного выполнения этого цикла происходит ожидание событий графического пользовательского интерфейса, их обработка, затем снова переход к ожиданию дальнейшего поступления событий.

Диспетчеры геометрии

В интерфейсе Tk предусмотрены три диспетчера геометрии, с помощью которых может осуществляться расстановка применяемого набора графических элементов. Первым для этой среды был создан диспетчер, именуемый *Placer*. Он был весьма несложным: разработчик указывал размеры графических элементов и места их расположения, затем диспетчер осуществлял расстановку элементов в соответствии с указаниями. Недостатком этого диспетчера было то, что в процессе программирования для каждого графического элемента приходилось задавать большой объем сведений, поэтому на разработчика возлагалась дополнительная нагрузка, связанная с подготовкой кода, которую можно было бы выполнить автоматически.

Второй, основной диспетчер геометрии, *Packer*, получил такое имя (*Упаковщик*) потому, что он упаковывает графические элементы в родительские объекты в соответствии с полученными инструкциями и распознает следующие “свободные участки”, которые нужно заполнить. Осуществляемый им процесс можно сравнить с укладыванием чемодана.

Третьим диспетчером геометрии является *Grid*. Он позволяет указывать местоположение графических элементов пользовательского интерфейса с помощью системы прямоугольных координат. Диспетчер *Grid* развертывает каждый объект графического пользовательского интерфейса в соответствующей позиции своей сетки. В настоящей главе в основном будет применяться диспетчер *Packer*.

Определив размеры и выровняв все графические элементы, диспетчер *Packer* размещает их на экране.

После того как все графические элементы встанут на свои места, приложению дается указание перейти в упомянутый выше бесконечный главный цикл. В интерфейсе Tkinter для этого применяется следующая инструкция:

```
Tkinter.mainloop()
```

Обычно эта инструкция стоит в самом конце последовательно выполняемой программы. После входа в главный цикл программа передает управление графическому пользовательскому интерфейсу. С этого момента все операции осуществляются с помощью обратных вызовов, даже выход из приложения. Как правило, выход из приложения осуществляется при выборе команды Exit в меню File или при непосредственном закрытии окна. При этом должен осуществляться обратный вызов, закрывающий приложение с графическим пользовательским интерфейсом.

5.2.3. Окно верхнего уровня: Tkinter.Tk()

Как уже было сказано, работа всех основных графических элементов основана на использовании окна верхнего уровня. Этот объект создается с помощью класса Tk интерфейса Tkinter. Для создания экземпляра объекта применяется следующий код:

```
>>> import Tkinter
>>> top = Tkinter.Tk()
```

В окне верхнего уровня размещаются как отдельные графические элементы, так и их группы, которые образуют графический пользовательский интерфейс. Перейдем к рассмотрению типов графических элементов. Прежде всего рассмотрим сами графические элементы Tk.

5.2.4. Графические элементы Tk

2.3

Во времени написания данной книги было создано 18 типов графических элементов для среды Tk. Эти графические элементы представлены в табл. 5.1. Новейшими из графических элементов являются `LabelFrame`, `PanedWindow` и `Spinbox`. Все эти три элемента появились в версии Python 2.3 (в результате поддержки версии Tk 8.4).

Таблица 5.1. Графические элементы Tk

Графический элемент	Описание
Button	Аналогичен <code>Label</code> , но предоставляет дополнительные функциональные возможности обработки операций наведения курсора, нажатия и отпускания кнопки мыши, а также действий/событий, связанных с клавиатурой
Canvas	Обеспечивает возможность рисовать фигуры (линии, овалы, многоугольники, прямоугольники); может содержать изображения, в том числе битовые
Checkbutton	Набор флажков, любые из которых могут быть <i>установлены</i> (по аналогии с элементом <code>checkbox</code> языка HTML)
Entry	Однорядное поле ввода, в котором можно вводить символы с клавиатуры (по аналогии с элементом ввода текста языка HTML)
Frame	Выполняет исключительно роль контейнера для других графических элементов
Label	Используется для размещения текста или изображений
LabelFrame	Сочетание метки и рамки, но с дополнительными атрибутами метки
Listbox	Предоставляет пользователю список вариантов, из которых может быть выбран только один вариант
Menu	Список команд элемента <code>Menubutton</code> , из которого пользователь может выбрать только одну команду
Menubutton	Предоставляет инфраструктуру для создания меню (ниспадающих, каскадных и т.д.)
Message	Аналогичен <code>Label</code> , но отображает многострочный текст
PanedWindow	Контейнерный графический элемент, с помощью которого можно управлять размещением других графических элементов, которые укладываются в нем
Radiobutton	Набор кнопок, из которых может быть нажата только одна (по аналогии с элементом ввода <code>radio</code> языка HTML)
Scale	Линейный графический элемент ползунок, позволяющий устанавливать точное значение заданного параметра; могут быть установлены начальное и конечное значения
Scrollbar	Предоставляет функциональные возможности прокрутки для графических элементов, поддерживающих эту операцию, таких как <code>Text</code> , <code>Canvas</code> , <code>Listbox</code> и <code>Entry</code>
Spinbox	Элемент, представляющий собой сочетание поля ввода с кнопкой, который позволяет задавать корректируемое значение
Text	Многострочное поле ввода, позволяющее собирать (или отображать) текст, вводимый пользователем (по аналогии с элементом <code>textarea</code> языка HTML)
Toplevel	Аналогичен <code>Frame</code> , но предоставляет отдельный контейнер окна

Мы не будем подробно рассматривать графические элементы Tk, поскольку эта тема представлена в многочисленных документальных источниках, включая разделы справки Tkinter на основном веб-сайте Python или огромное количество печатных

изданий и оперативных ресурсов, посвященных Tcl/Tk (некоторые из них представлены в приложении В, “Справочные таблицы”). Здесь будут приведены некоторые простые примеры, с помощью которых можно быстрее приступить к работе.



Необходимость в использовании параметров, заданных по умолчанию

Разработка графического пользовательского интерфейса на языке Python может быть значительно упрощена путем применения многочисленных параметров, заданных по умолчанию, которые предусмотрены в графических элементах Tkinter. Лучше всего начать с установки только хорошо изученных параметров и предоставить системе возможность самостоятельно определить остальные значения. Иным образом могут действовать только такие разработчики, которые знают каждый отдельный параметр каждого отдельно взятого графического элемента. Эти значения по умолчанию были выбраны очень тщательно. Пусть даже их значения не переопределены в программе, все равно можно не беспокоиться о том, что внешний вид интерфейса приложения на экране окажется несколько искаженным. Графические элементы, как правило, создаются с применением оптимизированных параметров, заданных по умолчанию, поэтому изменять значения этих параметров можно только в том случае, если точно известно, как должна быть выполнена настройка графических элементов.

5.3. Примеры Tkinter

Приступим к рассмотрению первых сценариев с применением графического пользовательского интерфейса. В каждом из этих сценариев представлены различные графические элементы, а иногда демонстрируются другие способы использования графического элемента, который уже был описан ранее. Вначале приведены очень простые примеры, а за ними следуют сценарии со средним уровнем сложности, которые уже приближаются к тому, с чем приходится сталкиваться на практике при разработке графических пользовательских интерфейсов.

5.3.1. Графический элемент Label

В примере 5.1 представлен сценарий `tkhello1.py`, который является аналогом версии программы Hello World! в интерфейсе Tkinter. В частности, в нем показано, как установить и выделить графический элемент Label.

Пример 5.1. Демонстрационная версия графического элемента Label (`tkhello1.py`)

В нашем первом примере Tkinter мы передаем привет всему миру, Hello World! (что же еще можно выбрать в качестве дебютного приложения?). Кроме того, в этом сценарии представлен первый из рассматриваемых графических элементов, Label.

```

1  #!/usr/bin/env python
2
3  import Tkinter
4
5  top = Tkinter.Tk()
6  label = Tkinter.Label(top, text='Hello World!')
7  label.pack()
8  Tkinter.mainloop()
```

В первой строке создается окно верхнего уровня. За ним следует графический элемент `Label`, который содержит знаменитую строку. Диспетчер окон `Packer` получает указание взять на себя управление этим графическим элементом и отобразить его, а затем следует вызов функции `mainloop()` и начинается выполнение приложения с графическим пользовательским интерфейсом. На рис. 5.1 показано, что появляется на экране после запуска этого приложения.



Рис. 5.1. Графический элемент `Label` интерфейса Tkinter

5.3.2. Графический элемент `Button`

Следующий пример (`tkhello2.py`) во многом напоминает предыдущий, но в нем вместо простой текстовой метки создается кнопка. Исходный код представлен в примере 5.2.

Пример 5.2. Демонстрационная версия графического элемента `Button` (`tkhello2.py`)

Этот пример полностью совпадает с примером сценария `tkhello1.py`, за исключением того, что вместо графического элемента `Label` создается графический элемент `Button`.

```

1  #!/usr/bin/env python
2
3  import Tkinter
4
5  top = Tkinter.Tk()
6  quit = Tkinter.Button(top, text='Hello World!',
7                        command=top.quit)
8  quit.pack()
9  Tkinter.mainloop()

```

Первые несколько строк сценариев практически совпадают. Отличия начинаются с момента создания графического элемента `Button`. Создаваемая кнопка имеет один дополнительный параметр, метод `Tkinter.quit()`. Этот метод устанавливает обратный вызов к кнопке, чтобы после ее нажатия (и отпускания) завершалось все приложение. Последние две строки представляют собой обычное обращение к диспетчеру, `pack()`, и вызов функции `mainloop()`. Это простое приложение с кнопкой показано на рис. 5.2.

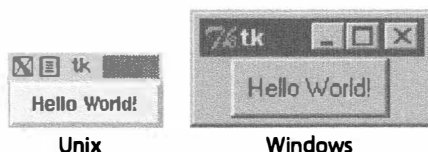


Рис. 5.2. Графический элемент `Button` интерфейса Tkinter

5.3.3. Графические элементы Label и Button

В примере 5.3 применяется сочетание кода сценариев tkhello1.py и tkhello2.py для формирования сценария tkhello3.py, в котором создаются и метка, и кнопка. Кроме того, здесь предоставлено больше параметров, чем в предыдущих сценариях, в которых мы ограничивались использованием параметров, заданных по умолчанию, которые определяются автоматически.

Пример 5.3. Демонстрационная версия графических элементов Label и Button (tkhello3.py)

Отличительной особенностью этого примера является то, что в нем создаются два графических элемента, Label и Button. Допустим, мы хорошо освоились с графическими элементами Button и знаем, как их настраивать, поэтому вместо заданных по умолчанию параметров при создании графического элемента задаем дополнительные параметры.

```
1  #!/usr/bin/env python
2
3  import Tkinter
4  top = Tkinter.Tk()
5
6  hello = Tkinter.Label(top, text='Hello World!')
7  hello.pack()
8
9  quit = Tkinter.Button(top, text='QUIT',
10                        command=top.quit, bg='red', fg='white')
11  quit.pack(fill=Tkinter.X, expand=1)
12
13  Tkinter.mainloop()
```

Дополнительные параметры заданы не только для графических элементов, но и для диспетчера окон Tacker. Параметр fill служит для диспетчера указанием, что кнопке QUIT может быть предоставлена вся оставшаяся часть пространства по горизонтали, а параметр expand указывает, что кнопка должна визуальнo заполнить всю область в горизонтальном направлении, от левого до правого края окна.

Как показано на рис. 5.3, если диспетчер окон Tacker не получает каких-либо иных инструкций, то графические элементы располагаются по вертикали (один над другим). Для создания горизонтального размещения необходимо создать новый объект Frame и добавлять кнопки с его помощью. Эта рамка занимает место родительского объекта как отдельный дочерний объект (см. кнопки в модуле listdir.py — пример 5.6 в разделе 5.3.6).

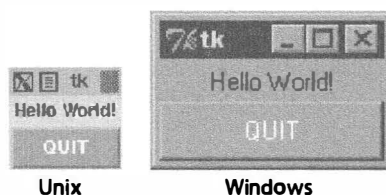


Рис. 5.3. Совместное применение графических элементов Label и Button интерфейса Tkinter

5.3.4. Графические элементы Label, Button и Scale

Особенностью последнего из вступительных примеров, tkhello4.py, является то, что в нем дополнительно введен графический элемент Scale. В данном случае Scale используется для взаимодействия с графическим элементом Label. Графический элемент Scale определяет ползунок — инструмент, который управляет размером текстового шрифта в графическом элементе Label. Чем выше расположен ползунок, тем крупнее шрифт, и наоборот. Код сценария tkhello4.py представлен в примере 5.4.

Пример 5.4. Демонстрация графических элементов Label, Button и Scale (tkhello4.py)

В последнем из вступительных примеров графических элементов представлен графический элемент Scale и показано, как графические элементы могут взаимодействовать друг с другом с использованием обратных вызовов (таких как `resize()`). Текст в графическом элементе Label затрагивается действиями, которые выполняются графическим элементом Scale.

```

1  #!/usr/bin/env python
2
3  from Tkinter import *
4
5  def resize(ev=None):
6      label.config(font='Helvetica -%d bold' % \
7                  scale.get())
8
9  top = Tk()
10 top.geometry('250x150')
11
12 label = Label(top, text='Hello World!',
13              font='Helvetica -12 bold')
14 label.pack(fill=Y, expand=1)
15
16 scale = Scale(top, from_=10, to=40,
17              orient=HORIZONTAL, command=resize)
18 scale.set(12)
19 scale.pack(fill=X, expand=1)
20
21 quit = Button(top, text='QUIT',
22              command=top.quit, activeforeground='white',
23              activebackground='red')
24 quit.pack()
25
26 mainloop()
```

В состав новых средств, применяемых в этом сценарии, входит функция обратного вызова `resize()` (строки 5–7), которая закреплена за элементом Scale. Именно этот код активизируется при перемещении ползунка элемента Scale и изменяет размер шрифта текста в элементе Label.

Кроме того, был определен размер (250 x 150) окна верхнего уровня (строка 10). Последним различием между этим сценарием и первыми тремя является то, что импорт атрибутов из модуля Tkinter в применяемое пространство имен осуществляется

с использованием инструкции `from Tkinter import *`. Обычно это не рекомендуется, поскольку приводит к заполнению пространства имен многими ненужными объектами, здесь же эта конструкция применяется. Такое решение главным образом обусловлено тем, что в данном приложении используется большое количество ссылок на атрибуты Tkinter. В противном случае нам пришлось бы задавать полные имена при доступе ко всем и к каждому атрибуту. Применение этого нежелательного сокращения позволяет обращаться к атрибутам с помощью более коротких строк программы и создавать код, более удобный для чтения, хотя и за счет некоторых издержек.

Как показано на рис. 5.4, в основной части окна можно видеть и механизм ползунка, и текущее заданное значение. На этом рисунке также показано состояние графического пользовательского интерфейса после перемещения указателя на шкале (ползунка) к значению 36. В коде заслуживает внимания то, что начальная установка для шкалы при запуске приложения равна 12 (строка 18).

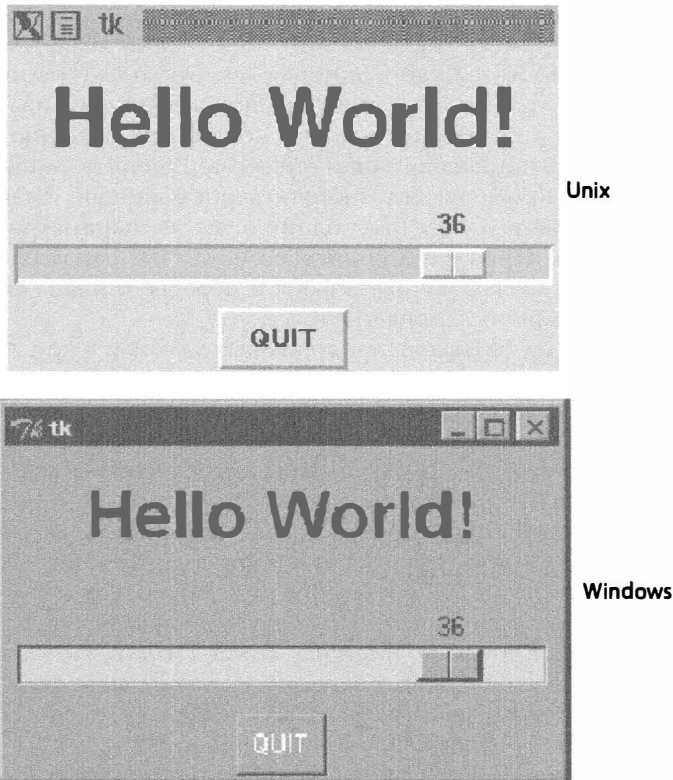


Рис. 5.4. Графические элементы Label, Button Scale интерфейса Tkinter

5.3.5. Более реальный пример

Прежде чем рассматривать более сложное приложение с графическим пользовательским интерфейсом, необходимо ознакомиться с концепцией *частичного применения функции* (Partial Function Application — PFA), описанным в книгах автора *Core Python Programming* и *Core Python Language Fundamentals*.

2.5

Механизм PFA был включен в версию 2.5 языка Python и представляет собой одно из существенных улучшений в области функционального программирования. С его помощью можно кешировать параметры функции, фактически фиксируя их как заранее заданные аргументы, а затем, во время выполнения программы, получив остальные необходимые параметры, извлечь из кэша зафиксированные аргументы, добавить их в список и вызвать функцию со всеми параметрами.

Лучше всего то, что механизм PFA не ограничивается только функциями. Он может применяться к любым вызываемым объектам, т.е. объектам, имеющим функциональный интерфейс, для чего достаточно лишь определить оператор вызова (круглые скобки) и указать классы, методы или вызываемые экземпляры. Использование механизма PFA идеально вписывается в ситуацию, когда имеется много вызываемых объектов и во многих вызовах снова и снова применяются одни и те же параметры.

Превосходным примером применения этих дополнений является программирование графического пользовательского интерфейса, поскольку велика вероятность, что от приложения будет требоваться согласованность внешнего вида элементов графического интерфейса, а этого можно добиться только при использовании одинаковых параметров при создании аналогичных объектов. Теперь перейдем к подготовке приложения, в котором применяются многочисленные кнопки, имеющие одинаковые цвета фона и переднего плана. Ввод одних и тех же параметров в каждой инструкции создания экземпляра для прорисовки почти одинаковых кнопок был бы напрасной затратой сил, поскольку цвета фона и переднего плана кнопок остаются одинаковыми, и лишь немного изменяется текст.

В данном примере будет создан аналог дорожных знаков и в приложении должна быть реализована попытка создать текстовые версии дорожных знаков с разделением их на три категории по типам, таким как запрещающие, предупреждающие или информационные (во многом по такому же принципу, как для ведения журнала устанавливаются три уровня сообщений). От типа знака зависит цветовая схема, применяемая при его создании. Например, в запрещающих знаках задается ярко красный цвет текста на белом фоне, в предупреждающих знаках применяется текст черного цвета на золотистом фоне, а информационные или регулирующие знаки отличаются наличием текста черного цвета на белом фоне. В реализуемый набор знаков входят знаки "Do Not Enter" (Въезд запрещен) и "Wrong Way" (Встречная полоса), которые являются запрещающими, а также "Merging Traffic" (Въезд на главную дорогу) и "Railroad Crossing" (Железнодорожный переезд), выполняющие функции предупреждающих. Наконец, предусмотрены регулирующие знаки "Speed Limit" (Ограничение скорости) и "One Way" (Одностороннее движение).

В приложении в примере 5.5 создаются знаки, которые представляют собой всего лишь кнопки. При нажатии пользователем кнопки отображается раскрывающееся диалоговое окно Tk, соответствующее ситуации запрещения и (или) ошибки, предупреждения или информирования. Безусловно, это приложение нельзя назвать слишком интересным, но на его примере можно изучить способ создания кнопок.

Пример 5.5. Приложение с графическим пользовательским интерфейсом, в котором применяется механизм PFA для создания дорожных знаков (pfaGUI2.py)

Создаются дорожные знаки с цветами фона и переднего плана, соответствующими типу знака. Механизм PFA позволяет объединять в шаблоны общие параметры графического пользовательского интерфейса.

```

1  #!/usr/bin/env python
2
3  from functools import partial as pto
4  from Tkinter import Tk, Button, X
5  from tkMessageBox import showinfo, showwarning, showerror
6
7  WARN = 'warn'
8  CRIT = 'crit'
9  REGU = 'regu'
10
11  SIGNS = {
12      'do not enter': CRIT,
13      'railroad crossing': WARN,
14      '55\speed limit': REGU,
15      'wrong way': CRIT,
16      'merging traffic': WARN,
17      'one way': REGU,
18  }
19
20  critCB = lambda: showerror('Error', 'Error Button Pressed!')
21  warnCB = lambda: showwarning('Warning',
22      'Warning Button Pressed!')
23  infoCB = lambda: showinfo('Info', 'Info Button Pressed!')
24
25  top = Tk()
26  top.title('Road Signs')
27  Button(top, text='QUIT', command=top.quit,
28      bg='red', fg='white').pack()
29
30  MyButton = pto(Button, top)
31  CritButton = pto(MyButton, command=critCB, bg='white', fg='red')
32  WarnButton = pto(MyButton, command=warnCB, bg='goldenrod1')
33  ReguButton = pto(MyButton, command=infoCB, bg='white')
34
35  for eachSign in SIGNS:
36      signType = SIGNS[eachSign]
37      cmd = '%sButton(text=%r%s).pack(fill=X, expand=True)' % (
38          signType.title(), eachSign,
39          '.upper()' if signType == CRIT else '.title()')
40      eval(cmd)
41
42  top.mainloop()

```

После запуска этого приложения разворачивается графический пользовательский интерфейс, показанный на рис. 5.5.

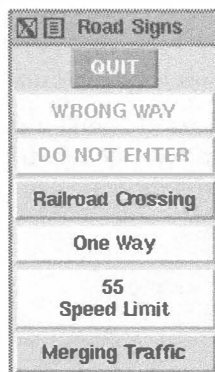


Рис. 5.5. Приложение с графическим пользовательским интерфейсом, в котором применяется механизм PFA для создания дорожных знаков, выполняемое в среде XDarwin в операционной системе Mac OS X

Построчное объяснение

Строки 1–18

Работа приложения начинается с импорта метода `functools.partial()`, задания нескольких атрибутов Tkinter и построения диалоговых окон Tk (строки 1–5). Затем определяются требуемые знаки с указанием их категории (строки 7–18).

Строки 20–28

Устанавливается связь между диалоговыми окнами Tk и функциями обратного вызова кнопок, которые используются для каждой созданной кнопки (строки 20–23). Затем происходит запуск среды Tk, задается название и создается кнопка QUIT (строки 25–28).

Строки 30–33

В этих строках заключается вся магия используемого механизма PFA. Применяются два уровня PFA. На первом уровне формируются шаблон класса `Button` и корневое окно `top`. Это означает, что при каждом вызове в программе объекта `MyButton` этот объект вызывает `Button` (метод `Tkinter.Button()` создает кнопку) с указанием `top` в качестве первого параметра. Эти действия предусмотрены в объекте `MyButton`.

На втором уровне использования механизма PFA применяется первый созданный объект, `MyButton`, и на его основе создаются шаблоны. Для знаков каждой категории формируются отдельные типы кнопок. При создании пользователем запрещающей кнопки `CritButton` (путем вызова ее, например, в форме `CritButton()`) следующим действием становится вызов `MyButton`, сопровождаемый определением необходимой функции обратного вызова кнопки и цветов фона и переднего плана. Иными словами, происходит вызов `Button` с указанием параметра `top`, функции обратного вызова и цветов. После этого можно наблюдать за тем, как происходит развертывание и переход вниз по уровням вплоть до каждой кнопки, в которой уже предусмотрен необходимый вызов. Если бы не возможности механизма PFA, то программисту пришлось бы задавать такой обратный вызов с нуля. Аналогичным образом создаются кнопки `WarnButton` и `ReguButton`.

Строки 35–42

2.5

После завершения этой подготовительной работы рассматривается список знаков и происходит их создание. Формируется строка, которую может вычислить интерпретатор Python, включающая имя создаваемой кнопки, в качестве текстового параметра передается метка на кнопке, затем к параметрам применяется метод `pack()`. Если формируемый знак является заглавным, то буквы метки на кнопке преобразуются в прописные, в противном случае в прописную преобразуются первые буквы каждого слова в метке. Последняя интересная особенность обнаруживается в строке 39, где демонстрируется еще одно средство, введенное в версии Python 2.5, — трехместный условный оператор. Порождение экземпляра каждой кнопки осуществляется с помощью метода `eval()`. Полученный результат показан на рис. 5.5. Наконец, происходит запуск графического пользовательского интерфейса путем перехода в основной цикл обработки событий.

Безусловно, трехместную операцию можно легко заменить синтаксическими конструкциями с операторами И/ИЛИ, если применяется версия 2.4 или предыдущая, что же касается средства `functools.partial()`, то его заменить сложнее, поэтому для работы с этим примером приложения рекомендуется использовать версию 2.5 или более новую.

5.3.6. Пример использования интерфейса Tkinter в более сложном приложении

В заключение рассмотрим более сложный сценарий, `listdir.py`, который представлен в примере 5.6. Это приложение может применяться в качестве инструмента обхода дерева каталогов. Оно начинает свою работу с текущего каталога и формирует листинг файлов. После двойного щелчка на любом другом каталоге из списка программа переходит в указанный каталог и формирует новый листинг, в котором представлены файлы нового каталога.

Пример 5.6. Программа с графическим пользовательским интерфейсом, предназначенная для навигации по файловой системе (`listdir.py`)

Создаваемый графический пользовательский интерфейс стал немного сложнее в результате применения графических элементов, добавления окон со списками, полей ввода текста и линейек прокрутки к прежнему набору. Увеличилось также разнообразие функций обратного вызова, которые теперь позволяют реагировать на щелчки кнопками мыши, нажатия клавиш и действия с линейкой прокрутки.

```

1  #!/usr/bin/env python
2
3  import os
4  from time import sleep
5  from Tkinter import *
6
7  class DirList(object):
8
9      def __init__(self, initdir=None):
10         self.top = Tk()
11         self.label = Label(self.top,
```

```

12         text='Directory Lister v1.1')
13     self.label.pack()
14
15     self.cwd = StringVar(self.top)
16
17     self.dirl = Label(self.top, fg='blue',
18         font=('Helvetica', 12, 'bold'))
19     self.dirl.pack()
20
21     self.dirfm = Frame(self.top)
22     self.dirsb = Scrollbar(self.dirfm)
23     self.dirsb.pack(side=RIGHT, fill=Y)
24     self.dirs = Listbox(self.dirfm, height=15,
25         width=50, yscrollcommand=self.dirsb.set)
26     self.dirs.bind('<Double-1>', self.setDirAndGo)
27     self.dirsb.config(command=self.dirs.yview)
28     self.dirs.pack(side=LEFT, fill=BOTH)
29     self.dirfm.pack()
30
31         self.dirn = Entry(self.top, width=50,
32             textvariable=self.cwd)
33         self.dirn.bind('<Return>', self.doLS)
34         self.dirn.pack()
35
36     self.bfm = Frame(self.top)
37     self.clr = Button(self.bfm, text='Clear',
38         command=self.clrDir,
39         activeforeground='white',
40         activebackground='blue')
41         self.ls = Button(self.bfm,
42             text='List Directory',
43             command=self.doLS,
44             activeforeground='white',
45             activebackground='green')
46     self.quit = Button(self.bfm, text='Quit',
47         command=self.top.quit,
48         activeforeground='white',
49         activebackground='red')
50     self.clr.pack(side=LEFT)
51     self.ls.pack(side=LEFT)
52     self.quit.pack(side=LEFT)
53     self.bfm.pack()
54
55     if initdir:
56         self.cwd.set(os.curdir)
57         self.doLS()
58
59     def clrDir(self, ev=None):
60         self.cwd.set('')
61
62     def setDirAndGo(self, ev=None):
63         self.last = self.cwd.get()
64         self.dirs.config(selectbackground='red')
65         check = self.dirs.get(self.dirs.curselection())
66         if not check:
67             check = os.curdir

```

```
68         self.cwd.set(check)
69         self.doLS()
70
71     def doLS(self, ev=None):
72         error = ''
73         tdir = self.cwd.get()
74         if not tdir: tdir = os.curdir
75
76         if not os.path.exists(tdir):
77             error = tdir + ': no such file'
78         elif not os.path.isdir(tdir):
79             error = tdir + ': not a directory'
80
81         if error:
82             self.cwd.set(error)
83
84         self.top.update()
85         sleep(2)
86         if not (hasattr(self, 'last') \
87                 and self.last):
88             self.last = os.curdir
89             self.cwd.set(self.last)
90             self.dirs.config(\
91                 selectbackground='LightSkyBlue')
92             self.top.update()
93             return
94
95         self.cwd.set(\
96             'FETCHING DIRECTORY CONTENTS...')
97         self.top.update()
98         dirlist = os.listdir(tdir)
99         dirlist.sort()
100        os.chdir(tdir)
101        self.dirl.config(text=os.getcwd())
102        self.dirs.delete(0, END)
103        self.dirs.insert(END, os.curdir)
104        self.dirs.insert(END, os.pardir)
105        for eachFile in dirlist:
106            self.dirs.insert(END, eachFile)
107        self.cwd.set(os.curdir)
108        self.dirs.config(\
109            selectbackground='LightSkyBlue')
110
111    def main():
112        d = DirList(os.curdir)
113        mainloop()
114
115    if __name__ == '__main__':
116        main()
```

На рис. 5.6 показано, как этот графический пользовательский интерфейс выглядит на экране ПК с операционной системой Windows. Снимок экрана пользовательского интерфейса этого приложения в операционной системе POSIX показан на рис. 5.7.

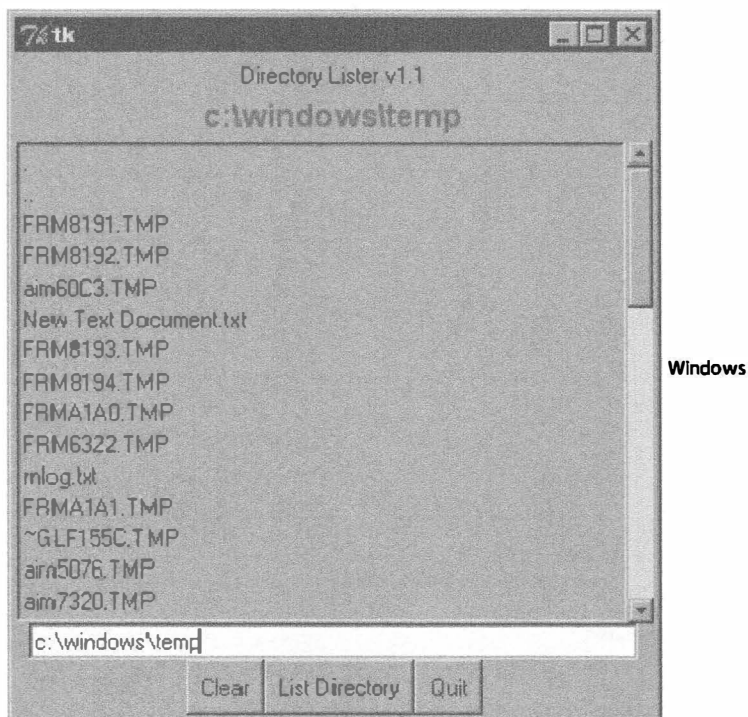


Рис. 5.6. Приложение с графическим пользовательским интерфейсом для работы с листингами каталогов в операционной системе Windows

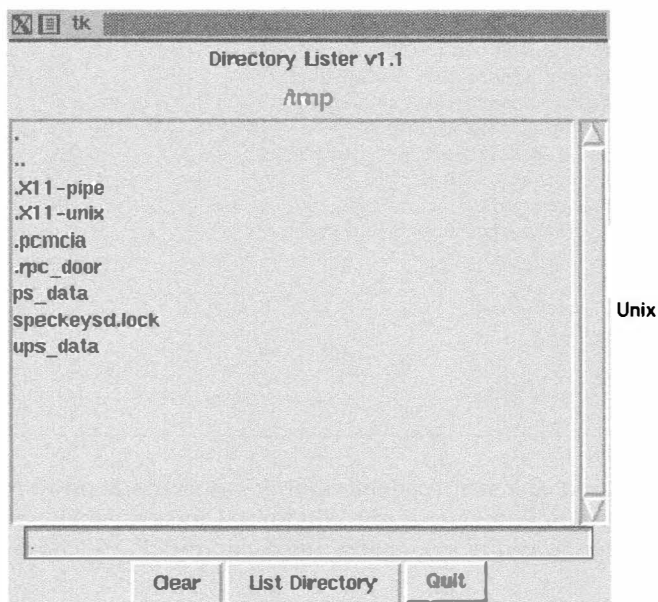


Рис. 5.7. Приложение с графическим пользовательским интерфейсом для работы с листингами каталогов в операционной системе Unix

Построчное объяснение

Строки 1–5

Эти первые несколько строк включают обычную строку запуска Unix, операторы импорта модуля `os`, метода `time.sleep()` и всех атрибутов модуля `Tkinter`.

Строки 9–13

В этих строках определен конструктор для класса `DirList` — объекта, который представляет рассматриваемое приложение. Первый создаваемый объект `Label` содержит метку с главным заголовком приложения и номером версии.

Строки 15–19

В объявлении переменной `Tk` с именем `cwd` задается имя текущего каталога; эта переменная потребуется в дальнейшем. Следующий создаваемый объект `Label` служит для отображения имени текущего каталога.

Строки 21–29

В этом разделе определяется основная часть применяемого графического пользовательского интерфейса, объект `Listbox` с именем `dirs`, который содержит листинг файлов рассматриваемого каталога. Применяется объект `Scrollbar`, позволяющий пользователю прокручивать листинг, если количество файлов превышает размер объекта `Listbox` по вертикали. Оба этих графических элемента содержатся в графическом элементе `Frame`. Элементы `Listbox` имеют обратный вызов (`setDirAndGo`), привязанный к ним с использованием метода `bind()` `Listbox`.

Под привязкой подразумевается установление соответствия между нажатием клавиши, действием мышью или каким-то другим событием и обратным вызовом, который выполняется при формировании такого события пользователем. Вызов функции `setDirAndGo()` происходит после двойного щелчка на одном из элементов в списке `Listbox`. Привязка элемента `Scrollbar` к элементу `Listbox` осуществляется путем вызова метода `Scrollbar.config()`.

Строки 31–34

Затем создается поле ввода `Entry`, в котором пользователь может ввести имя каталога, в который требуется перейти и просмотреть содержащиеся в нем файлы с помощью элемента `Listbox`. К этому полю ввода добавлена привязка к клавише `<Enter>`, чтобы пользователь мог нажимать эту клавишу возврата вместо щелчка кнопкой. Это относится и к привязке мыши, которая рассматривалась ранее, применительно к элементу `Listbox`. После того как пользователь дважды щелкнет на элементе в списке `Listbox`, осуществляется такое же действие, как при вводе имени каталога вручную в поле ввода `Entry` с последующим щелчком на кнопке `Go` (Перейти).

Строки 36–53

Затем определяется рамка `Button` (`bfm`) для размещения в ней трех кнопок: кнопки “очистки” (`clr`), кнопки “перехода” (`ls`) и кнопки “завершения” (`quit`). С каждой кнопкой связаны собственная конфигурация и обратный вызов, активизируемый после щелчка на ней.

Строки 55–57

В последней части конструктора инициализируется программа графического пользовательского интерфейса, действие которой начинается с текущего рабочего каталога.

Строки 59–60

Метод `clrDir()` очищает строковую переменную `Tk` с именем `cwd`, которая содержит имя текущего активного каталога. Эта переменная используется для отслеживания того, какой каталог рассматривается в данный момент, и, что более важно, помогает отслеживать предыдущий каталог на случай, если возникнут ошибки. Интерес представляют также переменные `ev` в функциях обратного вызова, имеющие значение по умолчанию `None`. Каждое из таких значений может быть передано по назначению с помощью системы управления окнами. Значения этих переменных в случае необходимости могут использоваться в функциях обратного вызова.

Строки 62–69

Метод `setDirAndGo()` задает каталог, в котором необходимо получить листинг файлов, и формирует вызов метода, с помощью которого осуществляется это действие, `doLS()`.

Строки 71–108

Метод `doLS()`, безусловно, является наиболее важным с точки зрения организации работы всего этого приложения с графическим пользовательским интерфейсом. В нем выполняются все предохранительные проверки (например, содержит ли текстовая строка имя каталога и существует ли этот каталог). Если возникает какая-либо ошибка, то в качестве текущего каталога снова устанавливается последний каталог. Если же ошибка не обнаруживается, то происходит вызов метода `os.listdir()` для получения фактических данных о файлах, присутствующих в каталоге, и с помощью этих данных происходит замена листинга в элементе `Listbox`. В то время как в фоновом режиме продолжается работа по выборке информации из нового каталога, синяя полоса, выделенная подсветкой, становится ярко красной. После окончательного перехода в новый каталог восстанавливается синий цвет полосы.

Строки 110–115

В сценарии `listdir.py` наиболее важной частью кода являются последние фрагменты кода. Функция `main()` выполняется только в случае непосредственного вызова сценария; после запуска `main()` на выполнение разворачивается приложение с графическим пользовательским интерфейсом, а затем вызывается функция `mainloop()` для запуска графического пользовательского интерфейса, которому передается контроль над приложением.

Оставляем рассмотрение всех прочих особенностей приложения в качестве упражнения для читателя. Напомним лишь, что проще рассматривать все это приложение как сочетание множества графических элементов и ряда функциональных средств. После достижения четкого понимания работы отдельных частей сценария можно будет успешно разобраться в том, как организована его работа в целом.

Автор надеется, что в этой главе ему удалось представить неплохое введение в программирование графического пользовательского интерфейса с помощью Python и Tkinter. Следует помнить, что наилучший способ ознакомления с

программированием с помощью интерфейса Tkinter состоит в неустанной практике и изучении существующих примеров! В состав дистрибутива Python входит большое количество демонстрационных приложений, которые можно изучать.

Загрузив исходный код интерпретатора Python, можно найти код, демонстрирующий работу Tkinter, в разделах Lib/lib-tk, Lib/idlelib и Demo/tkinter. После установки версии интерпретатора Python для среды Win32 и перехода в начальный каталог Python, допустим, C:\Python2x, можно найти демонстрационный код в подкаталогах Lib\lib-tk и Lib\idlelib. В последнем каталоге содержится наиболее значительное типичное приложение Tkinter: сама интегрированная среда разработки IDLE. В качестве источника дополнительных сведений можно воспользоваться несколькими книгами по программированию в среде Tk, в частности, теми, которые посвящены непосредственно интерфейсу Tkinter.

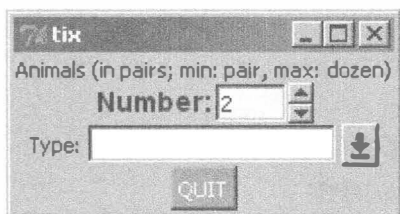
5.4. Краткий обзор других графических пользовательских интерфейсов

Автор надеется, что когда-либо подготовит отдельную главу, посвященную общим вопросам разработки графического пользовательского интерфейса, в которой будет использоваться весь широкий набор графических инструментальных средств, поддерживаемых языком Python, но эти планы пока не осуществились. На данный момент мы можем лишь рассмотреть единственное, простое приложение с графическим пользовательским интерфейсом, для написания которого использованы четыре из наиболее широко применяемых наборов инструментов: Tix (Tk Interface eXtensions), Pmw (расширение Python MegaWidgets для среды Tkinter), wxPython (привязка Python к wxWidgets) и PyGTK (привязка Python к GTK+). Последний пример демонстрирует, как использовать инструментальный Tile/Ttk в версиях Python 2 и Python 3. Ссылки на дополнительную информацию и на источники, из которых можно загрузить эти наборы инструментов, приведены в справочном разделе в конце этой главы.

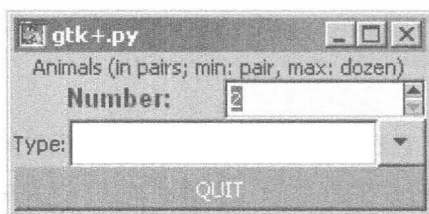
Модуль Tix уже представлен в стандартной библиотеке Python. Остальные модули разработаны отдельно от основного кода интерпретатора, поэтому должны быть загружены дополнительно. Модуль Pmw представляет собой лишь расширение среды Tkinter, поэтому его установка происходит проще всего (достаточно извлечь его из архива и включить в состав пакетов в каталоге site-packages). Что же касается модулей wxPython и PyGTK, то необходимо загрузить целый ряд файлов, а затем написать исполняемый код (в случае применения версий для Win32, которые обычно представлены в виде исполняемых программ, дело обстоит иначе). После установки и проверки этих инструментариев можно приступить к делу. В рассматриваемых примерах мы можем не ограничиваться теми графическими элементами, которые уже применялись в приложениях в этой главе, и перейти к использованию немного более сложных графических элементов.

Например, в дополнение к графическим элементам Label и Button можно ввести графические элементы Control или SpinButton и ComboBox. Графический элемент Control представляет собой сочетание текстового графического элемента и набора стрелок вверх и вниз. Текстовый графический элемент содержит значение, для управления которым, т.е. увеличения или уменьшения, применяется связанный с ним набор кнопок со стрелками. Графический элемент ComboBox обычно создается как сочетание текстового графического элемента и ниспадающего меню с набором вариантов, среди которых один является активным, или выбранным, и отображается в текстовом графическом элементе.

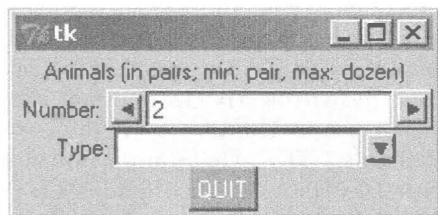
Рассматриваемое приложение является довольно простым: из одного места в другое перемещаются пары животных, в связи с чем изменяется общее количество животных, которое может принимать значение от двух до двенадцати. Для отслеживания общего количества служит графический элемент Control, а в элементе ComboBox представлено меню, содержащее различные типы животных, которые могут быть выбраны. На рис. 5.8 каждое изображение показывает состояние приложения с графическим пользовательским интерфейсом непосредственно после запуска. Следует учитывать, что по умолчанию количество животных равно двум, а тип животного не выбран.



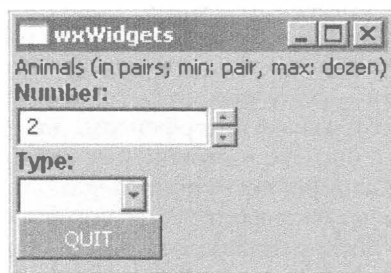
Tix



PyGTK



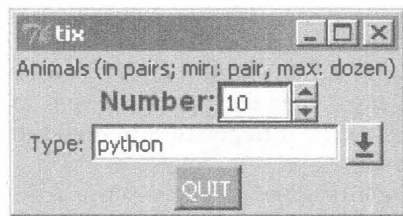
Pmw



wxPython

Рис. 5.8. Приложение, созданное в среде Win32 с применением различных вариантов графического пользовательского интерфейса

С началом использования этих вариантов обнаруживаются различия между приложениями. Об этом свидетельствует рис. 5.9, на котором показаны некоторые элементы после их модификации в приложении Tix.



Tix

Рис. 5.9. Версия приложения для графического пользовательского интерфейса в среде Tix после внесения изменений

Код для всех четырех версий рассматриваемого графического пользовательского интерфейса приведен в примерах 5.7–5.10. Вслед за этими примерами приведен пример 5.11, в котором используется среда Tile/Ttk (этот код поддерживается и в версии Python 2, и в версии Python 3). Можно видеть, что все варианты приложений в общем похожи друг на друга, но в каждом имеется свое особое отличие. Кроме того, используется расширение `.руw` для подавления всплывающих окон интерпретатора команд DOS, или терминальных окон.

5.4.1. Среда Tk Interface eXtensions (Tix)

Начнем с примера 5.7, в котором используется модуль Tix. Tix — это библиотека расширения для Tcl/Tk, в которой добавлено много новых графических элементов, типов изображения, а также дополнительных команд. Благодаря этому Tk становится практически применимым набором инструментов разработки графического пользовательского интерфейса. Рассмотрим, как использовать возможности Tix в сочетании с языком Python.

Пример 5.7. Демонстрационная версия графического пользовательского интерфейса Tix (`animalTix.руw`)

В первом из рассматриваемых примеров используется модуль Tix. Следует учитывать, что Tix входит в состав Python!

```
1  #!/usr/bin/env python
2
3  from Tkinter import Label, Button, END
4  from Tix import Tk, Control, ComboBox
5
6  top = Tk()
7  top.tk.eval('package require Tix')
8
9  lb = Label(top,
10             text='Animals (in pairs; min: pair, max: dozen)')
11  lb.pack()
12
13  ct = Control(top, label='Number:',
14               integer=True, max=12, min=2, value=2, step=2)
15  ct.label.config(font='Helvetica -14 bold')
16  ct.pack()
17
18  cb = ComboBox(top, label='Type:', editable=True)
19  for animal in ('dog', 'cat', 'hamster', 'python'):
20      cb.insert(END, animal)
21  cb.pack()
22
23  qb = Button(top, text='QUIT',
24              command=top.quit, bg='red', fg='white')
25  qb.pack()
26
27  top.mainloop()
```

Построчное объяснение

Строки 1–7

Это означает, что для него предусмотрен весь код настройки, имеются команды импорта модуля и задана основная инфраструктура графического пользовательского интерфейса. В строке 7 дается подтверждение того, что модуль Tkinter доступен в приложении.

Строки 8–27

В следующих строках создаются все графические элементы: Label (строки 9–11), Control (строки 13–16), ComboBox (строки 18–21) и, наконец, Button (строки 23–25). Конструкторы и параметры для графических элементов довольно очевидны и не требуют дополнительных пояснений. В конечном итоге происходит переход в основной цикл обработки событий графического пользовательского интерфейса (строка 27).

5.4.2. Объекты Python MegaWidgets (PMW)

Перейдем к рассмотрению объектов Python MegaWidgets (которые демонстрируются в примере 5.8). Модуль MegaWidgets был создан для решения проблемы устаревания Tkinter. По существу этот модуль позволяет продлить срок существования модуля Tkinter путем добавления более современных графических элементов к его палитре средств графического пользовательского интерфейса.

Пример 5.8. Демонстрационная версия графического пользовательского интерфейса Pmw (animalPmw.pyw)

Во втором примере используется пакет Python MegaWidgets.

```

1  #!/usr/bin/env python
2
3  from Tkinter import Button, END, Label, W
4  from Pmw import initialise, ComboBox, Counter
5
6  top = initialise()
7
8  lb = Label(top,
9      text='Animals (in pairs; min: pair, max: dozen)')
10 lb.pack()
11
12 ct = Counter(top, labelpos=W, label_text='Number:',
13     datatype='integer', entryfield_value=2,
14     increment=2, entryfield_validate={'validator':
15     'integer', 'min': 2, 'max': 12})
16 ct.pack()
17
18 cb = ComboBox(top, labelpos=W, label_text='Type:')
19 for animal in ('dog', 'cat', 'hamster', 'python'):
20     cb.insert(end, animal)
21 cb.pack()
22
23 qb = Button(top, text='QUIT',
24     command=top.quit, bg='red', fg='white')
25 qb.pack()
26
27 top.mainloop()
```

Пример применения модуля `Rmw` настолько напоминает пример для `Tix`, что мы оставляем его построчный анализ читателю. Наибольшие отличия содержатся в строке кода вызова конструктора для графического элемента управления, `Counter`. Конструктор обеспечивает проверку допустимости входных параметров. Вместо определения минимального и максимального возможных значений в качестве ключевых параметров для конструктора графического элемента в модуле `Rmw` используется средство проверки для контроля за тем, чтобы значения не выходили за пределы допустимого диапазона.

Модули `Tix` и `Rmw` представляют собой расширения для интерфейсов `Tk` и `Tkinter`, соответственно, но теперь мы отойдем от проблематики `Tk` и перейдем к рассмотрению совершенно других наборов инструментов: `wxWidgets` и `GTK+`. Это более современные и надежные наборы инструментов для графического пользовательского интерфейса, причем заслуживает внимания то, что по мере все более широкого применения объектно-ориентированных средств в программах увеличивается количество начальных строк, которые служат для подготовки к работе.

5.4.3. Модули `wxWidgets` и `wxPython`

Модуль `wxWidgets` (который прежде именовался `wxWindows`) представляет собой кроссплатформенный набор инструментов, который может использоваться для создания приложений с графическим пользовательским интерфейсом. Он реализован с помощью языка `C++` и доступен на различных платформах, для которых `wxWidgets` определяет единообразный и общий программный интерфейс приложений (API). Лучше всего то, что модуль `wxWidgets` использует собственный графический пользовательский интерфейс для каждой платформы, поэтому интерфейс программы всегда имеет примерно такой же внешний вид, как и у всех прочих приложений на рабочем столе данной платформы. Еще одна особенность состоит в том, что можно не ограничиваться разработкой приложений `wxWidgets` на языке `C++`, поскольку имеются интерфейсы и для `Python`, и для `Perl`. Образец нашего экспериментального приложения, созданного с использованием `wxPython`, приведен в примере 5.9.

Пример 5.9. Демонстрационная версия приложения с графическим пользовательским интерфейсом `wxPython` (`animalWx.pyw`)

В третьем примере используются средства `wxPython` (и `wxWidgets`). Заслуживает внимания то, что в целях упорядочения все графические элементы помещены в объект, устанавливающий размеры. Кроме того, следует отметить, что данное приложение является в большей степени объектно-ориентированным по сравнению с предыдущими примерами.

```
1  #!/usr/bin/env python
2
3  import wx
4
5  class MyFrame(wx.Frame):
6      def __init__(self, parent=None, id=-1, title=''):
7          wx.Frame.__init__(self, parent, id, title,
8                          size=(200, 140))
9          top = wx.Panel(self)
10         sizer = wx.BoxSizer(wx.VERTICAL)
11         font = wx.Font(9, wx.SWISS, wx.NORMAL, wx.BOLD)
12         lb = wx.StaticText(top, -1,
```

```

13         'Animals (in pairs; min: pair, max: dozen)')
14     sizer.Add(lb)
15
16     c1 = wx.StaticText(top, -1, 'Number:')
17     c1.SetFont(font)
18     ct = wx.SpinCtrl(top, -1, '2', min=2, max=12)
19     sizer.Add(c1)
20     sizer.Add(ct)
21
22     c2 = wx.StaticText(top, -1, 'Type:')
23     c2.SetFont(font)
24     cb = wx.ComboBox(top, -1, '',
25                     choices=('dog', 'cat', 'hamster', 'python'))
26     sizer.Add(c2)
27     sizer.Add(cb)
28
29     qb = wx.Button(top, -1, "QUIT")
30     qb.SetBackgroundColour('red')
31     qb.SetForegroundColour('white')
32     self.Bind(wx.EVT_BUTTON,
33               lambda e: self.Close(True), qb)
34     sizer.Add(qb)
35
36     top.SetSizer(sizer)
37     self.Layout()
38
39     class MyApp(wx.App):
40         def OnInit(self):
41             frame = MyFrame(title="wxWidgets")
42             frame.Show(True)
43             self.SetTopWindow(frame)
44             return True
45
46     def main():
47         pp = MyApp()
48         app.MainLoop()
49
50     if __name__ == '__main__':
51         main()

```

Построчное объяснение

Строки 5–37

В данном случае создается экземпляр класса `Frame` (строки 5–8), единственным членом которого является конструктор. В действительности данный метод предназначен лишь для создания графических элементов. Создается рамка, а внутри нее — элемент `Panel`. В панели используется элемент `BoxSizer` для включения и размещения всех применяемых графических элементов (строки 10, 36), которые состоят из элементов `Label` (строки 12–14), `SpinCtrl` (строки 16–20), `ComboBox` (строки 22–27) и включают элемент `Button` — кнопку выхода (строки 29–34).

Элементы `Labels` с метками были добавлены к графическим элементам `SpinCtrl` и `ComboBox` вручную, поскольку вполне очевидно, что в них изначально не предусмотрены метки. После определения всех графических элементов производится их

добавление к элементу, определяющему размеры, последний устанавливается на панели и происходит упорядочение всех элементов. Как показывает строка 10, элемент, определяющий размер, расположен по вертикали, а это означает, что размещение графических элементов будет осуществляться сверху вниз.

Одним из недостатков графического элемента `SpinCtrl` является то, что он не поддерживает функциональное средство пошагового увеличения или уменьшения значения. В других трех примерах мы имели возможность щелкать на кнопке выделения со стрелкой, которая увеличивает или уменьшает значение с шагом два, но данный графический элемент этого не позволяет.

Строки 39–51

В классе приложения создается экземпляр объекта `Frame`, который был только что разработан, объект разворачивается на экране и задается в качестве самого верхнего окна нашего приложения. Наконец, в строках настройки осуществляются создание экземпляра приложения с графическим пользовательским интерфейсом и его запуск.

5.4.4. Интерфейсы GTK+ и PyGTK

Наконец, рассмотрим еще одну версию графических средств, `PyGTK`, которая весьма напоминает графический пользовательский интерфейс `wxPython` (см. пример 5.10). Важнейшей отличительной особенностью этой версии является то, что в ней используется только один класс, и создается впечатление, будто становится труднее задавать цвета фона и переднего плана для объектов, особенно кнопок.

Пример 5.10. Демонстрационная версия графического пользовательского интерфейса `PyGTK` (`animalGtk.pyw`)

В заключительном примере используются графические средства `PyGTK` (и `GTK+`). Как и в примере `wxPython`, в данном случае для приложения также используется один класс. Любопытно отметить, насколько похожими друг на друга и вместе с тем разными являются все рассматриваемые приложения с графическим пользовательским интерфейсом. В этом нет ничего удивительного, а положительным фактором становится то, что программисты могут переключаться с одного набора инструментов на другой относительно легко.

```
1  #!/usr/bin/env python
2
3  import pygtk
4  pygtk.require('2.0')
5  import gtk
6  import pango
7
8  class GTKapp(object):
9      def __init__(self):
10         top = gtk.Window(gtk.WINDOW_TOPLEVEL)
11         top.connect("delete_event", gtk.main_quit)
12         top.connect("destroy", gtk.main_quit)
13         box = gtk.VBox(False, 0)
14         lb = gtk.Label(
15             'Animals (in pairs; min: pair, max: dozen)')
16         box.pack_start(lb)
```

```

17
18     sb = gtk.HBox(False, 0)
19     adj = gtk.Adjustment(2, 2, 12, 2, 4, 0)
20     sl = gtk.Label('Number:')
21     sl.modify_font(
22         pango.FontDescription("Arial Bold 10"))
23     sb.pack_start(sl)
24     ct = gtk.SpinButton(adj, 0, 0)
25     sb.pack_start(ct)
26     box.pack_start(sb)
27
28     cb = gtk.HBox(False, 0)
29     c2 = gtk.Label('Type:')
30     cb.pack_start(c2)
31     ce = gtk.combo_box_entry_new_text()
32     for animal in ('dog', 'cat', 'hamster', 'python'):
33         ce.append_text(animal)
34     cb.pack_start(ce)
35     box.pack_start(cb)
36
37     qb = gtk.Button("")
38     red = gtk.gdk.color_parse('red')
39     sty = qb.get_style()
40     for st in (gtk.STATE_NORMAL,
41               gtk.STATE_PRELIGHT, gtk.STATE_ACTIVE):
42         sty.bg[st] = red
43     qb.set_style(sty)
44     ql = qb.child
45     ql.set_markup('<span color="white">QUIT</span>')
46     qb.connect_object("clicked",
47                       gtk.Widget.destroy, top)
48     box.pack_start(qb)
49     top.add(box)
50     top.show_all()
51
52 if __name__ == '__main__':
53     animal = GTKapp()
54     gtk.main()

```

Построчное объяснение

Строки 1–6

Осуществляется импорт трех различных модулей и пакетов (PyGTK, GTK и Pango). Модуль Pango — это библиотека для компоновки и прорисовки текста, специально предназначенного для представления текста на разных языках, или как принято называть такую возможность, для интернационализации (I18N). В данном случае необходимость в этом обусловлена самой сутью организации обработки текста и шрифтов для GTK+ (версия 2.x).

Строки 8–50

Все графические элементы рассматриваемого приложения представлены в классе GTKapp. Создается самое верхнее окно (с обработчиками для его закрытия с помощью диспетчера окон), после чего создается расположенный по вертикали элемент

определения размеров (VBox) для хранения первичных графических элементов. Именно такие же действия выполнялись при работе с графическим пользовательским интерфейсом wxPython.

В данном случае решено располагать статические метки для SpinButton и ComboBoxEntry рядом с этими графическими элементами (а не над ними, как в примере wxPython), поэтому создаются небольшие расположенные по горизонтали поля для включения пар “метка—графический элемент” (строки 18–35) и полученные объекты VBox помещаются во всеобъемлющий элемент VBox.

После создания кнопки Button выхода из программы и добавления VBox к самому верхнему окну производится прорисовка всех объектов на экране. Важно отметить, что кнопка вначале создается с пустой меткой. Это делается для того, чтобы в составе кнопки можно было создать объект Label как дочерний объект. Затем, в строках 44–45, открывается доступ к метке и задается текст с белым цветом шрифта.

Дело в том, что если цвет переднего плана для стиля, допустим, будет определяться в цикле с использованием вспомогательного кода в строках 40–43, то это изменение цвета переднего плана повлияет лишь на цвет переднего плана кнопки, а не метки. Например, если задать стиль с белым цветом переднего плана и применить к кнопке выделение подсветкой (путем нажатия клавиши табуляции до тех пор, пока не произойдет ее выбор), то будет виден лишь внутренний выделенный точками прямоугольник, показывающий, что выбранный графический элемент имеет белый цвет, но текст метки по-прежнему будет черным, если не произойдет его изменения с помощью разметки в строке 45.

Строки 52–54

Теперь происходит создание приложения и переход в основной цикл событий.

5.4.5. Модуль Tile/Ttk

Со времени ее выпуска библиотека Tk приобрела солидную репутацию как гибкая и простая библиотека со связанным с ней набором инструментов, который позволяет создавать объекты графического пользовательского интерфейса. Однако по истечении первого десятилетия после начала работы с этой библиотекой пользователи и разработчики стали все более отчетливо осознавать, что без новых средств, серьезных изменений и обновлений библиотека Tk будет во все большей степени восприниматься как устаревающая и отстающая от более современных наборов инструментов, таких как wxWidgets и GTK+.

Попытка решить эту проблему была предпринята при создании инструментария Tix, в котором предусмотрены новые графические элементы, типы изображений и новые команды, позволяющие расширить возможности Tk. В некоторых из основных графических элементов Tix использовался даже собственный код пользовательского интерфейса, поэтому они в еще большей степени имели такой же внешний вид и напоминали другие приложения в той же системе управления окнами. Однако эти усилия влекли за собой лишь расширение функциональных возможностей Tk.

В середине 2000-х годов был предложен более радикальный подход: набор графических элементов Tile, который представляет собой новую реализацию большинства основных графических элементов Tk, в которую включено несколько новых элементов. В этой реализации, получившей название Tile, не только стал шире применяться собственный код, но и было включено ядро тематической организации.

Наборы графических элементов стали тематическими и появилась возможность создавать, импортировать и экспортировать темы, поэтому разработчики (и пользователи) получили больший контроль над визуальным внешним видом приложений, а также значительно упростилась интеграция с операционной системой и работающей в ней системой управления окнами. Эта особенность Tile оказалась настолько привлекательной, что были предприняты усилия по интеграции этой библиотеки с ядром Tk в версии 8.5 и созданию средств Ttk. Таким образом, набор графических элементов Ttk воспринимается не как замена, а как приложение к оригинальному набору основных графических элементов Tk.

2.7

Модуль Tile/Ttk впервые появился в версиях Python 2.7 и 3.1. Для работы с Ttk в применяемой версии Python необходимо иметь, как минимум, доступ к любой версии Tk 8.5; лучше если это будет последняя версия, но применимы и более старые версии, при условии, что установлена библиотека Tile. В версии Python 2.7 и последующих версиях доступ к средствам Tile/Ttk предоставляется с помощью модуля ttk; с другой стороны, в версии Python 3.1 и последующих версиях эти средства охвачены модулем tkinter, поэтому достаточно лишь выполнить импорт `tkinter.ttk`.

3.1

В примерах 5.11 и 5.12 показано, как используются версии Python 2 и 3 в приложениях `animalTtk.pyw` и `animalTtk3.pyw`. Независимо от того, используется ли версия Python 2 или 3, после запуска на выполнение открывается экран приложения с пользовательским интерфейсом, аналогичный показанному на рис. 5.10.

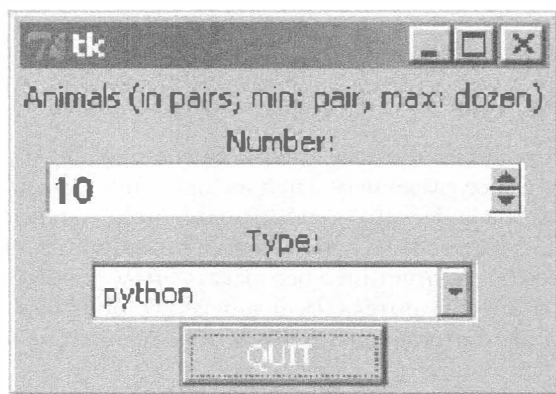


Рис. 5.10. Пользовательский интерфейс экспериментального приложения в Tile/Ttk

Пример 5.11. Демонстрационная версия графического пользовательского интерфейса Tile/Ttk (`animalTtk.pyw`)

В демонстрационном приложении используется набор инструментов Tile (получивший название Ttk после интеграции с Tk 8.5).

```

1  #!/usr/bin/env python
2
3  from Tkinter import Tk, Spinbox
4  from ttk import Style, Label, Button, Combobox
5

```

```
6 top = Tk()
7 Style().configure("TButton",
8     foreground='white', background='red')
9
10 Label(top,
11     text='Animals (in pairs; min: pair, '
12     'max: dozen)').pack()
13 Label(top, text='Number:').pack()
14
15 Spinbox(top, from_=2, to=12,
16     increment=2, font='Helvetica -14 bold').pack()
17
18 Label(top, text='Type:').pack()
19
20 Combobox(top, values=('dog',
21     'cat', 'hamster', 'python')).pack()
22
23 Button(top, text='QUIT',
24     command=top.quit, style="TButton").pack()
25
26 top.mainloop()
```

Пример 5.12. Демонстрационная версия графического пользовательского интерфейса Tile/Ttk для версии Python 3 (animalTtk3.pyw)

В демонстрации для версии Python 3 используется набор инструментов Tile (названный Ttk после интеграции с Tk 8.5).

```
1 #!/usr/bin/env python3
2
3 from tkinter import Tk, Spinbox
4 from tkinter.ttk import Style, Label, Button, Combobox
5
6 top = Tk()
7 Style().configure("TButton",
8     foreground='white', background='red')
9
10 Label(top,
11     text='Animals (in pairs; min: pair, '
12     'max: dozen)').pack()
13 Label(top, text='Number:').pack()
14
15 Spinbox(top, from_=2, to=12,
16     increment=2, font='Helvetica -14 bold').pack()
17
18 Label(top, text='Type:').pack()
19
20 Combobox(top, values=('dog',
21     'cat', 'hamster', 'python')).pack()
22
23 Button(top, text='QUIT',
24     command=top.quit, style="TButton").pack()
25
26 top.mainloop()
```

Построчное объяснение

Строки 1–4

В версии Tk 8.4 в состав основных графических элементов Tk вошли три новых графических элемента. Одним из них был Spinbox, который будет использоваться в данном приложении. (Двумя другими являются LabelFrame и PanedWindow.) Все остальные используемые графические элементы относятся к библиотеке Tile/Ttk: Label, Button и Combobox, а также класс Style, который позволяет обеспечить организацию графических элементов по темам.

Строки 6–8

Эти строки применяются исключительно для инициализации корневого окна, а также объекта Style, содержащего организованные по темам элементы для графических элементов, в которых решено их использовать. Темы позволяют добиваться единообразного внешнего вида графических элементов. Безусловно, может показаться расточительным использование такой организации для одной лишь кнопки выхода из программы, но без нее нельзя обойтись, поскольку отсутствует возможность непосредственно задавать конкретные цвета переднего плана и фона для кнопок. Кроме того, программист становится вынужденным разрабатывать приложения с применением более единообразного способа. То, что в этом простейшем примере кажется небольшим неудобством, оборачивается выработкой полезной привычки, которая оправдывает себя на практике.

Строки 10–26

В оставшейся части кода главным образом происходит определение (и упаковка) всего применяемого набора графических элементов таким образом, который ненамного отличается по сравнению с тем, что происходит в других приложениях с пользовательским интерфейсом, рассматриваемых в данной главе: создается метка, определяющая приложение, разворачивается комбинация элементов Label и Spinbox, которые управляют числовым диапазоном возможных значений (и их пошаговым изменением), а также пара элементов Label и Combobox, позволяющая пользователям выбрать животное, наконец, кнопка выхода Button. Код заканчивается вхождением в главный цикл графического пользовательского интерфейса.

Это построчное объяснение может подойти и для описания аналога для версии Python 3, показанного в примере 5.12, причем единственное изменение наблюдается в операторе импорта: библиотека Tkinter переименована в версии Python 3 в tkinter, а модуль ttk стал вспомогательным модулем tkinter.

5.5. Связанные модули и другие графические пользовательские интерфейсы

В языке Python могут использоваться также другие системы разработки программ с графическим пользовательским интерфейсом. Применимые модули вместе с соответствующими им системами окон перечислены в табл. 5.2.

Таблица 5.2. Системы создания графических пользовательских интерфейсов, предусмотренные для Python

Библиотека графического пользовательского интерфейса	Описание
Модули, связанные с Tk	
Tkinter/tkinter*	TK INTERface. Применяемый по умолчанию набор инструментов графического пользовательского интерфейса Python http://wiki.python.org/moin/TkInter
Pmw	Элементы Python MegaWidgets (расширение Tkinter) http://pmw.sf.net
Tix	Tk Interface eXtension (расширение Tk) http://tix.sf.net
Tile/Ttk	Организованный по темам набор графических элементов Tile/Ttk http://tktable.sf.net
TkZinc (Zinc)	Расширенный тип холста Tk (расширение Tk) http://www.tkzinc.org
EasyGUI (easygui)	Очень простые графические пользовательские интерфейсы (расширение Tkinter), в которых не применяется управление событиями http://ferg.org/easygui
TIDE + (IDE Studio)	Интегрированная среда разработки Tix (включает IDE Studio, дополненную средствами Tix расширенную версию стандартной интегрированной среды разработки IDLE), которая представлена по адресу http://starship.python.net/crew/mike
Модули, связанные с wxWidgets	
wxPython	Привязка языка Python к wxWidgets, межплатформенной инфраструктуре поддержки графических пользовательских интерфейсов (которая прежде именовалась wxWindows) http://wxpython.org
Boa Constructor	Интегрированная среда разработки Python и построитель графического пользовательского интерфейса wxPython http://boa-constructor.sf.net
PythonCard	Набор средств создания приложений для рабочего стола с графическим пользовательским интерфейсом, wxPython (созданный под влиянием идей HyperCard) http://pythoncard.sf.net
wxGlade	Еще один конструктор графического пользовательского интерфейса wxPython (созданный под влиянием идей Glade построитель графических пользовательских интерфейсов GTK+/GNOME) http://wxglade.sf.net
Модули, связанные с GTK+/GNOME	
PyGTK	Оболочка Python для библиотеки набора инструментов GIMP (GTK+) http://pygtk.org
Модули, связанные с GTK+/GNOME	
GNOME-Python	Привязка Python к рабочему столу и библиотекам разработки GNOME http://gnome.org/start/unstable/bindings http://download.gnome.org/sources/gnome-python

Библиотека графического пользовательского интерфейса	Описание
Glade	Построитель графических пользовательских интерфейсов для GTK+ и GNOME http://glade.gnome.org
PyGUI (графический пользовательский интерфейс)	Межплатформенный API графического пользовательского интерфейса в стиле Python, основанный на Cocoa (Mac OS X) и GTK+ (POSIX/X11 и Win32) http://www.cosc.canterbury.ac.nz/~greg/python_gui
Модули, связанные с Qt/KDE	
PyQt	Привязка Python для набора инструментов Qt GUI/XML/SQL C++ от компании Trolltech, частично представленная с открытым исходным кодом (имеет двойную лицензию) http://riverbankcomputing.co.uk/pyqt
PyKDE	Привязка Python для среды рабочего стола KDE http://riverbankcomputing.co.uk/pykde
eric	Интегрированная среда разработки Python, написанная в среде PyQt с использованием редактора графических элементов QScintilla http://die-offenbachs.de/detlev/eric3 http://ericide.python-hosting.com/
PyQtGPL	Qt (юниксоидная среда Cygwin, перенесенная в Win32), Sip, QScintilla, привязка PyQt http://pythonqt.vanrietpaap.nl
Другие наборы инструментов для графического пользовательского интерфейса с открытым исходным кодом	
FXPy	Привязка Python к набору инструментов FOX (http://fox-toolkit.org) http://fxpy.sf.net
pyFLTK (fltk)	Привязка Python к набору инструментов FLTK (http://fltk.org) http://pyfltk.sf.net
PyOpenGL (OpenGL)	Привязка Python к OpenGL (http://opengl.org) http://pyopengl.sf.net
Коммерческие пакеты	
win32ui	Microsoft MFC (доступен через Python для расширений Windows) http://starship.python.net/crew/mhammond/win32
swing	Sun Microsystems Java/Swing (доступен через Jython) http://jython.org

^a Tkinter для Python 2 и tkinter для Python 3.

Дополнительные сведения обо всех графических интерфейсах пользователя, связанных с языком Python, можно найти на общей странице по программированию графического пользовательского интерфейса вики-сайта Python по адресу <http://wiki.python.org/moin/GuiProgramming>.

5.6. Упражнения

- 5.1. *Архитектура “клиент–сервер”*. Опишите назначение оконных сервера и клиента.
- 5.2. *Объектно-ориентированное программирование*. Опишите связь между дочерними и родительскими графическими элементами.
- 5.3. *Графические элементы Label*. Внесите изменения в сценарий `tkhello1.py`, чтобы в нем отображалось ваше собственное сообщение, а не `Hello World!`
- 5.4. *Графические элементы Label и Button*. Внесите изменения в сценарий `tkhello3.py`, чтобы в нем раскрывались три новые кнопки в дополнение к кнопке `QUIT`. Нажатие любой из этих трех кнопок должно приводить к изменению текстовой метки так, чтобы в ней появлялся текст из нажатого графического элемента `Button`. **Подсказка.** Необходимо предусмотреть три отдельных обработчика или обеспечить настройку одного обработчика в зависимости от предварительно заданных параметров (количество объектов функции все равно должно быть равно трем).
- 5.5. *Графические элементы Label, Button и Radiobutton*. Внесите изменения в свое решение упражнения 5.4, чтобы разворачивались три кнопки `Radiobutton`, представляющие варианты текста для `Label`. Предусмотрены две кнопки: кнопка `QUIT` и кнопка `Update`. После щелчка на кнопке `Update` текстовая метка должна изменяться так, чтобы в ней появлялся текст из выбранной кнопки `Radiobutton`. Если отметка не стоит ни на одной кнопке `Radiobutton`, значение `Label` должно оставаться неизменным.
- 5.6. *Графические элементы Label, Button и Entry*. Внесите изменения в свое решение упражнения 5.5, чтобы три кнопки `Radiobutton` были заменены одним графическим элементом в виде поля текста `Entry` со значением по умолчанию `Hello World!` (отражающим начальное строковое значение `Label`). Поле `Entry` должно допускать редактирование пользователем с заданием новой текстовой строки для `Label`, которая будет обновляться после щелчка на кнопке `Update`.
- 5.7. *Графические элементы Label и Entry и система ввода-вывода Python*. Создайте приложение с графическим пользовательским интерфейсом, предусматривающее создание поля `Entry`, в котором пользователь может указать имя текстового файла. Откройте файл и прочитайте его, затем отобразите содержимое в элементе `Label`.

Дополнительное задание (меню). Замените графический элемент `Entry` элементом меню, имеющим команду `File Open`, предусматривающую появление всплывающего окна, которое позволяет пользователю задать имя файла для чтения. Кроме того, добавьте в меню команду `Exit` или `Quit`, чтобы дополнить кнопку `QUIT`.

- 5.8. *Простой текстовый редактор*. Используйте свое решение предыдущей задачи для создания простого текстового редактора. Файл может создаваться с нуля или считываться и отображаться в графическом элементе `Text`, который допускает редактирование пользователем. По завершении пользователем работы с приложением (либо с использованием кнопки `QUIT`, либо с помощью команды меню `Quit/Exit`) для пользователя должен выводиться запрос, сохранить ли введенные изменения или завершить работу без сохранения.

Дополнительное задание. Установите интерфейс между вашим сценарием и программой проверки правописания и добавьте кнопку или опцию меню, чтобы проверить орфографию текста в файле. Слова с орфографическими ошибками должны выделяться в графическом элементе Text подсветкой с использованием другого цвета текста переднего плана или фона.

- 5.9. *Многопоточные приложения для интерактивной переписки.* Программы интерактивной переписки, приведенные в предыдущих главах, требуют усовершенствования. Создайте полнофункциональный, многопоточный сервер интерактивной переписки. Для этого сервера фактически не требуется графический пользовательский интерфейс, если сервер должен создаваться лишь в качестве обслуживающего клиентские запросы в той конфигурации, в которой он применяется, такой как номер порта, имя, соединение с сервером имен и т.д. Создайте многопоточный клиент интерактивной переписки, в котором используются отдельные потоки для отслеживания ввода данных пользователем (и отправки сообщения на сервер для широковещательной рассылки), а еще один поток принимает входящие сообщения, которые должны быть отображены перед пользователем. Клиентская часть графического пользовательского интерфейса, обслуживающего клиентские запросы, должна представлять собой окно интерактивной переписки, состоящее из двух частей: более крупная часть должна допускать размещение большого числа строк, чтобы в ней мог быть представлен весь диалог, а меньшее поле ввода текста должно принимать ввод от пользователя.
- 5.10. *Применение других графических пользовательских интерфейсов.* Примеры приложений с графическим пользовательским интерфейсом, в которых применяются различные наборы инструментов, очень похожи друг на друга, но не являются одинаковыми. Разумеется, невозможно добиться того, чтобы все эти приложения ни в чем не отличались друг от друга. Попробуйте внести в них такие изменения, чтобы они выглядели более единообразными, чем сейчас.
- 5.11. *Применение строителей графических пользовательских интерфейсов.* Конструкторы графических пользовательских интерфейсов позволяют быстрее разрабатывать приложения с графическим пользовательским интерфейсом, поскольку автоматически вырабатывают от имени программиста стандартный код, возлагая на него выполнение более сложных задач. Загрузите тот или иной инструмент конструктора графических пользовательских интерфейсов и реализуйте графический пользовательский интерфейс для программ, рассматриваемых в этой главе, просто перетаскивая графические элементы из соответствующей палитры. Выполните привязку к графическим элементам функций обратного вызова, чтобы эти элементы действовали так же, как в примерах приложений, рассматриваемых в этой главе.

Какие конструкторы графических пользовательских интерфейсов вы могли бы применить? Для работы с wxWidgets см. PythonCard, wxGlade, XRCed, wxFormBuilder или даже Boa Constructor (больше не поддерживается), а для GTK+ предусмотрен Glade (а также связанный с ним GtkBuilder). Чтобы больше узнать о подобных инструментах, обратитесь к разделу *GUI Design Tools and IDEs* на странице вики-сайта инструментариев графического пользовательского интерфейса по адресу <http://wiki.python.org/moin/GuiProgramming>.

ГЛАВА

6

Программирование баз данных

В этой главе...

- Введение
- Спецификация DB-API Python
- Объектно-реляционные преобразователи
- Нереляционные базы данных
- Справочная информация

Вы действительно назвали своего сына Robert');

`DROP TABLE Students;-- ?`

Рэндалл Манро (Randall Munroe),

ХКCD, октябрь 2007 года

В главе показано, как осуществлять взаимодействие с базами данных с помощью Python. Потребности небольших приложений в хранении данных могут быть удовлетворены с помощью файлов или несложных систем постоянного хранения, но для более крупных серверных приложений или приложений с большим объемом данных может потребоваться полнофункциональная система баз данных. По этой причине в данной главе рассматриваются реляционные и нереляционные базы данных, а также объектно-реляционные преобразователи (Object-Relational Mapper — ORM).

6.1. Введение

Во вступительном разделе рассмотрено, для чего нужны базы данных, представлен язык SQL (Structured Query Language), а также описан программный интерфейс приложений для баз данных (API) Python.

6.1.1. Система постоянного хранения

Необходимость в постоянном хранении данных возникает практически в любом приложении. Вообще говоря, существуют три основных механизма хранения данных: файлы, системы баз данных или своего рода сочетание того и другого, например API, который обеспечивает сопряжение с одной из существующих систем, объектно-реляционным преобразователем, диспетчером файлов, электронной таблицей, файлом конфигурации и т.д.

В главе "Файлы" книги *Core Python Language Fundamentals* или *Core Python Programming* приведены сведения о том, как создать систему постоянного хранения данных с использованием открытого доступа к файлам, а также языка Python и диспетчера баз данных (DBM), который может представлять собой традиционный механизм постоянного хранения данных Unix, оболочку к файловой системе (к таким файлам, как `*dbm`, `dbhash/bsddb` и `shelve` — сочетания `pickle` и DBM) и использовать интерфейс для объекта, организованного по принципу словаря.

В настоящей главе основное внимание уделено использованию баз данных в таких ситуациях, когда попытка организовать работу на основе файлов или создать собственную систему хранения данных становится неприемлемой в связи с большими масштабами проекта. В подобных случаях приходится принимать много важных решений. Таким образом, цель настоящей главы состоит в том, чтобы представить основы и показать все возможные варианты организации работы с базами данных (в частности, из сценария на языке Python), чтобы читатель мог принять правильное решение. Начнем описание этой темы с рассмотрения языка SQL и реляционных баз данных, поскольку они все еще представляют собой преобладающую форму создания системы постоянного хранения данных.

6.1.2. Основные операции с базами данных и язык SQL

Прежде чем вплотную приступить к рассмотрению баз данных и их использования в языке Python, представим краткое введение в некоторые элементарные понятия

баз данных и SQL (которое может рассматриваться как обзор теми читателями, которые уже знакомы с этой темой).

Основы системы хранения

В основе баз данных обычно находится фундаментальная система постоянного хранения, в которой используется файловая система, в частности, обычные файлы, поддерживаемые операционной системой, специальные файлы в операционной системе и даже бесформатные разделы дисков.

Пользовательский интерфейс

Для большинства систем баз данных предусмотрен инструмент с интерфейсом командной строки, с помощью которого можно выдавать команды или запросы SQL к базе данных. Иногда применяются также некоторые инструменты с графическим интерфейсом пользователя, в которых используются клиентские команды для командной строки или клиентская библиотека базы данных; такие инструменты предоставляют пользователям намного более удобный интерфейс.

Базы данных

Система управления реляционными базами данных (сокращенно реляционная СУБД) обычно обеспечивает управление сразу несколькими базами данных, например, с данными о сбыте, маркетинге, поддержке пользователей и т.д. В таком случае все базы данных находятся на одном и том же сервере (если в основе работы реляционной СУБД лежит сервер; в более простых системах это правило не всегда соблюдается). В данной главе приведены примеры, в которых, в частности, показано, как используется библиотека MySQL для работы с реляционной СУБД на основе сервера, притом что серверный процесс непрерывно работает и ожидает поступления команд; ни SQLite, ни Gadfly не поддерживают постоянно работающие серверы.

Компоненты

Для баз данных таблица служит абстрактным представлением системы хранения данных. Каждая строка данных состоит из полей, а поля, в свою очередь, соответствуют столбцам таблицы базы данных. Вся совокупность определений таблиц, состоящих из столбцов, и типов данных в каждой таблице определяет так называемую *схему базы данных*.

Предусмотрена возможность создавать и уничтожать базы данных. Те же операции предусмотрены для таблиц. Операция добавления новых строк в таблицу базы данных называется *вставкой*; внесение изменений в существующие строки таблицы называется *обновлением*, а исключение существующих строк из таблицы — *удалением*. Эти действия обычно именуются *командами* или *операциями* базы данных. Выборку данных из строк таблицы базы данных с указанием необязательных условий принято называть выполнением *запросов*.

При выполнении запроса к базе данных можно осуществлять выборку сразу всех необходимых результатов (строк) или организовать последовательное получение данных, обрабатывая в цикле каждую результирующую строку. Некоторые базы данных позволяют использовать так называемые *курсоры* при выдаче команд и запросов SQL и получении результатов, которые могут поступать все одновременно или построчно.

SQL

Для передачи команд и запросов в базу данных главным образом используется язык SQL. Применение языка SQL предусмотрено не во всех базах данных, но в большинстве реляционных баз данных такая возможность существует. Ниже приведены некоторые примеры команд SQL. Следует учитывать, что в большинстве баз данных настройка выполнена так, что регистр в них не учитывается, особенно в командах. Поэтому общепринятым является такой стиль, что ключевые слова для базы данных записываются прописными буквами. Большинство программ с интерфейсом командной строки предъявляет требование, чтобы каждая инструкция SQL завершалась точкой с запятой (;).

Создание базы данных

```
CREATE DATABASE test;  
GRANT ALL ON test.* to user(s);
```

В первой строке создается база данных `test`, а во второй строке от имени администратора базы данных предоставляются разрешения конкретным пользователям на то, чтобы они могли выполнять следующие операции с базой данных.

Использование базы данных

```
USE test;
```

Если вход в систему баз данных произведен без указания того, какая база данных должна использоваться, то можно с помощью этой простой инструкции указать, с какой базой данных будет происходить работа.

Удаление базы данных

```
DROP DATABASE test;
```

Эта несложная инструкция позволяет удалить все таблицы и данные из базы данных, затем исключить саму базу данных из системы.

Создание таблицы

```
CREATE TABLE users (login VARCHAR(8), userid INT, projid INT);
```

Эта инструкция позволяет создать новую таблицу со строковым столбцом `login` и двумя целочисленными полями, `userid` и `projid`.

Удаление таблицы

```
DROP TABLE users;
```

Эта несложная инструкция позволяет удалить таблицу базы данных наряду со всеми ее данными.

Вставка строки

```
INSERT INTO users VALUES('leanna', 2111, 1);
```

Предусмотрена возможность вставить новую строку в базу данных с использованием инструкции INSERT. Необходимо указать таблицу и значения, относящиеся к каждому полю. В рассматриваемом примере строка 'leanna' записывается в поле login, а значения 2111 и 1 — в столбцы userid и projid соответственно.

Обновление строки

```
UPDATE users SET projid=4 WHERE projid=2;
UPDATE users SET projid=1 WHERE userid=311;
```

Для изменения существующих строк таблицы используется инструкция UPDATE. Предложение SET указывает, какие столбцы должны быть изменены, и предоставляет необходимые критерии для выбора строк, подлежащих изменению. В первом примере все пользователи с идентификатором проекта (или projid), равным 2, будут перемещены в проект #4. Во втором примере берутся данные об одном пользователе (с UID 311) и перемещаются в проект #1.

Удаление строки

```
DELETE FROM users WHERE projid=%d;
DELETE FROM users;
```

Для удаления строки таблицы необходимо взять за основу команду DELETE FROM, указать таблицу, из которой должны быть удалены строки, а также задать все дополнительные условия. Если это не будет сделано, как во втором примере, будут удалены все строки.

Эти краткие сведения позволяют проще приступить к ускоренному изучению основных понятий баз данных, а в дальнейшем следовать за изложением в остальной части главы и изучать приведенные в ней примеры. Если вам потребуется дополнительная помощь, обратитесь к учебникам по базам данных, в которых рассматривается данная тематика.

6.1.3. Базы данных и язык Python

Перейдем к рассмотрению API базы данных Python и описанию того, как получить доступ к реляционным базам данных непосредственно из сценария Python через интерфейс базы данных или с помощью объектно-реляционного преобразователя. Кроме того, будет показано, как выполнить ту же задачу, но при этом не выдавать в явном виде команды на языке SQL.

В настоящей главе не рассматриваются такие темы, как принципы работы баз данных, параллельность, схемы, атомарность, целостность, восстановление, правильное оформление сложных операций левого соединения, триггеры, оптимизация запроса, транзакции, хранимые процедуры и т.д. Эта глава посвящена исключительно тому, как работать с базами данных непосредственно из сценария на языке Python. Тем не менее будет показано, как осуществлять сохранение и выборку данных с

помощью реляционных СУБД, действуя в рамках инфраструктуры Python. Это даст возможность читателю решить, какой вариант организации работы является наилучшим для его текущего проекта или приложения. Кроме того, глава позволяет изучить образцы кода, которые дают возможность сразу же приступить к решению намеченной задачи. Это даст возможность овладеть тематикой как можно быстрее, что позволит интегрировать конкретное приложение Python с той или иной системой баз данных.

Помимо этого, в данной главе мы впервые отказываемся от применявшегося до сих пор принципа рассматривать только средства стандартной библиотеки Python, не требующие установки каких-либо дополнительных программ. Первоначальный замысел этой книги состоял действительно в том, чтобы не выходить за рамки стандартных средств, но в процессе подготовки ее текста стало очевидно, что такой подход не применим для баз данных, которые, без всякого сомнения, превратились в основной компонент повседневной разработки приложений в мире Python.

Настоящая книга рассчитана на программистов с инженерным уклоном, которые, по-видимому, не смогут прочно занять свое место, ничего не зная о базах данных: о том, как работать с ними (из командной строки или с помощью графического интерфейса пользователя), как извлекать данные с использованием языка SQL, добавлять или обновлять информацию в базе данных и т.д. Те, кто избрал Python в качестве своего инструмента программирования, могут не сомневаться в том, что для добавления средств обеспечения доступа к базе данных к инструментарию Python уже проделан огромный объем работы. Вначале рассмотрим API баз данных Python, или сокращенно DB-API, затем перейдем к примерам интерфейсов баз данных, которые соответствуют этому стандарту.

Кроме того, в некоторых примерах будут использоваться широко применяемые реляционные СУБД с открытым исходным кодом. Тем не менее мы не будем затрагивать темы, касающиеся сравнения программ с открытым исходным кодом и коммерческих программных продуктов. Задача освоения реляционных СУБД того или иного типа должна быть довольно несложной. Отдельного упоминания заслуживает база данных Gadfly Эрона Уоттерса (Aaron Watters) — простая реляционная СУБД, программное обеспечение которой написано полностью на языке Python.

Для доступа к любой базе данных из сценария Python применяются адаптеры. *Адаптер* — это модуль Python, с помощью которого можно установить интерфейс к клиентской библиотеке реляционной базы данных, обычно на языке C. Согласно общепринятой рекомендации, все адаптеры Python соответствуют определениям API группы пользователей по интересам, которые занимаются базами данных Python (DB-SIG). Перейдем к первой важной теме данной главы.

На рис. 6.1 показаны программные уровни, связанные с написанием приложения Python для базы данных, с применением или без применения объектно-реляционного преобразователя. Как показано на этом рисунке, в качестве интерфейса к библиотекам клиента базы данных на языке C послужит DB-API.

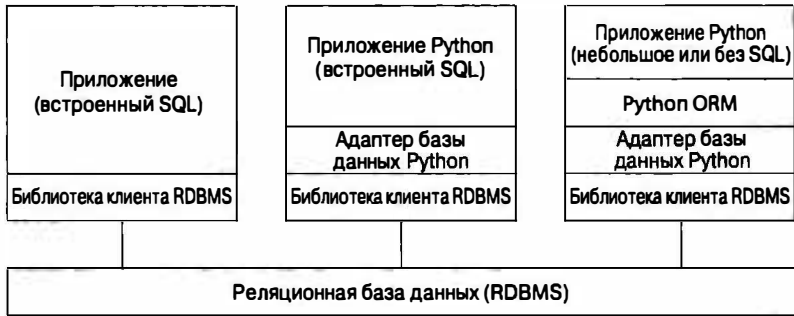


Рис. 6.1. Многоуровневое взаимодействие приложения и базы данных. В качестве приложения с встроенным кодом SQL (вверху слева) обычно применяется программа на языке C/C++, взаимодействующая с базой данных с помощью клиентской библиотеки, тогда как для приложений Python применяются адаптеры, совместимые с DB-API. Объектно-реляционные преобразователи позволяют упростить приложение, поскольку полностью берут на себя взаимодействие с базой данных

6.2. Спецификация DB-API Python

Возникает вопрос: как найти интерфейсы, необходимые для взаимодействия с базой данных? Ответ на него несложен. Достаточно перейти к тематическому разделу, посвященному базам данных, на главном веб-сайте Python. Здесь находятся ссылки на полную текущую версию DB-API (версия 2.0), существующие модули базы данных, документацию, группу пользователей по интересам (SIG) и т.д. Определение интерфейса DB-API сразу после его принятия было перенесено в документ PEP 249. (Этот документ PEP заменил старую спецификацию DB-API 1.0, которая определена в документе PEP 248.)

Краткое описание DB-API

DB-API — это спецификация, которая определяет ряд объектов и механизмов, необходимых для предоставления доступа к базе данных, которые остаются единообразными, какие бы адаптеры баз данных и лежащие в их основе системы баз данных не применялись. Как и в случае большинства разработок, проводимых силами сообщества пользователей, создание DB-API было продиктовано насущной необходимостью.

В старые времена существовало много баз данных, для которых разные специалисты реализовывали собственные адаптеры. Это можно сравнить с изобретением колеса, которое происходило снова и снова. Базы данных и адаптеры создавались в разное время и разными людьми без соблюдения какой-либо согласованности функциональных средств. К сожалению, в связи с этим прикладной код, созданный с использованием подобных интерфейсов, приходилось каждый раз переделывать с учетом того, какой модуль базы данных был выбран для применения. Кроме того, после внесения любых изменений в интерфейс приходилось также заниматься обновлением кода приложения.

Поэтому была сформирована группа SIG, занимающаяся вопросами взаимодействия с базами данных из сценариев на языке Python, и в конечном итоге ей удалось

сформулировать необходимый интерфейс: DB-API версии 1.0. DB-API стал выполнять роль спецификации для определения единообразного интерфейса к самым разным реляционным базам данных, поэтому перенос кода доступа от одной базы данных к другой стал намного проще, чаще всего требуя внесения правок лишь в несколько строк кода. Соответствующие примеры будут приведены ниже.

6.2.1. Атрибуты модуля

Спецификация DB-API требует, чтобы модуль доступа к базе данных предоставлял перечисленные ниже средства и атрибуты. Модуль, совместимый с DB-API, должен определять глобальные атрибуты, приведенные в табл. 6.1.

Таблица 6.1. Атрибуты модуля DB-API

Атрибут	Описание
<code>apilevel</code>	Версия DB-API, с которой совместим данный адаптер
<code>threadsafety</code>	Уровень потокобезопасности данного модуля
<code>paramstyle</code>	Стиль параметров инструкций SQL данного модуля
<code>connect()</code>	Функция <code>Connect()</code>
(Различные исключения)	(См. табл. 6.4)

Атрибуты данных

Атрибут `apilevel`

Данная строка (не представляющая собой число с плавающей точкой) указывает самую высокую версию DB-API, с которой совместим данный модуль, например 1.0, 2.0 и т.д. Если это обозначение отсутствует, то по умолчанию принято считать, что значение равно 1.0.

Атрибут `threadsafety`

Это целое число, которое может принимать следующие возможные значения.

- 0. Отсутствие потоковой безопасности: применять потоки для совместного доступа к модулю нельзя.
- 1. Минимальная потоковая безопасность: потокам может предоставляться совместный доступ к модулю, но не к соединениям.
- 2. Средняя потоковая безопасность: потокам может предоставляться совместный доступ к модулю и соединениям, но не к курсорам.
- 3. Полная потоковая безопасность: потокам может предоставляться совместный доступ к модулю, соединениям и к курсорам.

Если к ресурсу предоставляется совместный доступ, то для блокировки на атомарном уровне должны применяться примитивы синхронизации, такие как спин-блокировки или семафоры. Применение файлов на диске и глобальных переменных не обеспечивает достаточной надежности при решении данной задачи и может оказать отрицательное воздействие при выполнении стандартных операций с мьютексом. Ознакомьтесь с описанием модуля `threading` или возвратитесь к главе 4, за дополнительной информацией о том, как использовать блокировки.

Атрибут `paramstyle`

Спецификация DB-API поддерживает разнообразные способы указания того, как параметры должны быть встроены в инструкцию SQL, которая в конечном итоге передается на сервер для выполнения. Этот строковый параметр всего лишь указывает форму подстановки строки, которая используется при формировании строк для запроса или команды (табл. 6.2).

Таблица 6.2. Стили параметров базы данных `paramstyle`

Стиль параметра	Описание	Пример
<code>numeric</code>	Стиль с числовым обозначением позиции	<code>WHERE name=:1</code>
<code>named</code>	Стиль с именованием	<code>WHERE name=:name</code>
<code>pyformat</code>	Преобразование формата в словарь Python с помощью функции <code>printf()</code>	<code>WHERE name=% (name) s</code>
<code>qmark</code>	Стиль с вопросительными знаками	<code>WHERE name=?</code>
<code>format</code>	Преобразование формата в строку ANSI C с помощью функции <code>printf()</code>	<code>WHERE name=%s</code>

Атрибуты функции `connect()`

Доступ с помощью функций к базе данных предоставляется через объекты `Connection`. Совместимый модуль должен реализовывать функцию `connect()`, которая создает и возвращает объект `Connection`. Параметры функции `connect()` приведены в табл. 6.3.

Таблица 6.3. Атрибуты функции `connect()`

Параметр	Описание
<code>user</code>	Имя пользователя
<code>password</code>	Пароль
<code>host</code>	Имя хоста
<code>database</code>	Имя базы данных
<code>dsn</code>	Имя источника данных

Предусмотрена возможность передавать информацию о соединении с базой данных в виде строки с несколькими параметрами (DSN), указывать отдельные параметры как позиционные (если точно известен правильный порядок) или, что применяется чаще всего, задавать параметры с помощью ключевых слов. Ниже приведен пример использования функции `connect()` из документа PEP 249.

```
connect(dsn='myhost:MYDB', user='guido', password='234$')
```

Необходимость в использовании DSN (Data Set Number — номер набора данных), а не отдельных параметров определяется главным образом тем, к какой системе выполняется подключение. Например, если используется API наподобие ODBC (Open Database Connectivity) или JDBC (Java DataBase Connectivity), то с наибольшей вероятностью придется применять DSN, а если работа осуществляется непосредственно с базой данных, то скорее всего нужно будет задавать отдельные параметры регистрации. Еще одной причиной указанной ситуации является то, что в большинстве адаптеров баз данных не реализована поддержка для DSN. Ниже приведены некоторые примеры вызовов функции `connect()` без использования DSN. Заслуживает

внимания то, что не все адаптеры реализованы в точном соответствии со спецификацией, например, для MySQLdb используется ключевое слово db вместо database.

- MySQLdb.connect(host='dbserv', db='inv', user='smith')
- PostgreSQL.connect(database='sales')
- psycopg2.connect(database='template1', user='pgsql')
- gadsfly.dbapi20.connect('csrDB', '/usr/local/database')
- sqlite3.connect('marketing/test')

Исключения

В табл. 6.4 показаны исключения, которые также должны быть заданы в совместимом модуле в качестве глобальных переменных.

Таблица 6.4. Классы исключений DB-API

Исключение	Описание
Warning	Корневой класс исключений уровня предупреждений
Error	Корневой класс исключений уровня ошибок
InterfaceError	Ошибка интерфейса базы данных (не самой базы данных)
DatabaseError	Ошибка базы данных
DataError	Проблемы с обрабатываемыми данными
OperationalError	Ошибка во время выполнения операции с базой данных
IntegrityError	Ошибка реляционной целостности базы данных
InternalError	Ошибка, которая произошла в базе данных
ProgrammingError	Неудачное завершение команды SQL
NotSupportedError	Обнаружена неподдерживаемая операция

6.2.2. Объекты класса Connection

Соединение представляет собой способ взаимодействия приложения с базой данных. Соединения формируют фундаментальный механизм, с помощью которого происходит передача команд на сервер и возврат результатов с сервера. После установления соединения (или получения соединения из пула) создаются курсоры для передачи запросов и получения ответов из базы данных.

Методы объекта Connection

Объекты Connection не обязаны иметь какие-либо атрибуты данных, но должны определять методы, показанные в табл. 6.5.

Таблица 6.5. Методы объекта Connection

Имя метода	Описание
close()	Закрыть соединение с базой данных
commit()	Зафиксировать текущую транзакцию
rollback()	Отменить текущую транзакцию
cursor()	Создать (и вернуть) курсор или объект, подобный курсору, с использованием этого соединения
errorhandler(cxn, cur, errcls, errval)	Служит в качестве обработчика для указанного курсора соединения

После выполнения функции `close()` продолжение использования того же соединения становится невозможным, поскольку может быть получено исключение.

Метод `commit()` является неприменимым, если база данных не поддерживает транзакции или в ней включено средство автоматической фиксации. По желанию можно реализовать отдельные методы для включения или отключения автоматической фиксации. Этот метод определен в спецификации DB-API и является обязательным, поэтому для баз данных, которые не поддерживают транзакции, должен быть реализован вариант этого метода с параметром `pass`.

Как и `commit()`, функция `rollback()` имеет смысл, только если в базе данных поддерживаются транзакции. После выполнения функции `rollback()` база данных должна возвратиться точно в такое же состояние, в котором она находилась до начала транзакции. Согласно документу PEP 249, *“закрытие соединения без предварительной фиксации изменений должно приводить к неявному выполнению предварительного отката”*.

Даже если реляционная СУБД не поддерживает курсоры, функция `cursor()` все равно должна возвращать объект, который достоверно эмулирует или моделирует реальный объект курсора. Это лишь минимальные требования. Разработчик каждого отдельного адаптера всегда может добавить специальные атрибуты, определенно предназначенные для создаваемого им интерфейса или базы данных.

К разработчикам адаптеров относится также рекомендация (но не требование), чтобы доступ ко всем исключениям модуля базы данных (см. выше) предоставлялся через соединение. В противном случае предполагается, что активизацию соответствующего исключения уровня модуля обеспечивает объект `Connection`. По окончании использования соединения и курсоры должны быть закрыты, ко всем операциям следует применить функцию `commit()`, затем вызвать функцию `close()` для закрытия соединения.

6.2.3. Объекты класса `Cursor`

Сразу после установления соединения можно приступить к взаимодействию с базой данных. Как было указано ранее во вступительном разделе, курсор позволяет пользователю выдавать команды базы данных и осуществлять выборку строк, возвращенных запросами. Объект курсора DB-API Python полностью поддерживает функциональные возможности курсора, даже если в самой базе данных поддержка курсоров не предусмотрена. В этом случае, если создается адаптер базы данных, необходимо реализовывать объекты `cursor`, чтобы они действовали в качестве курсоров. Благодаря этому код Python остается единообразным, независимо от того, с какой системой баз данных ведется работа и поддерживает ли она курсоры или нет.

После создания курсора можно выполнить запрос или команду (или несколько запросов и команд) и осуществить выборку одной или нескольких строк из результирующего набора. Атрибуты данных и методы объекта класса `Cursor` представлены в табл. 6.6.

Таблица 6.6. Атрибуты объекта класса `Cursor`

Атрибут объекта	Описание
<code>arraysize</code>	Количество строк, подлежащих единовременной выборке с помощью <code>fetchmany()</code> ; значение по умолчанию — 1
<code>connection</code>	Соединение, в котором создан этот курсор (необязательный параметр)

Атрибут объекта	Описание
<code>description</code>	Возвращает данные об активности курсора (кортеж с 7 элементами): (<code>name</code> , <code>type_code</code> , <code>display_size</code> , <code>internal_size</code> , <code>precision</code> , <code>scale</code> , <code>null_ok</code>); только параметры <i>name</i> и <i>type_code</i> являются обязательными
<code>lastrowid</code>	Идентификатор последней измененной строки (необязательный параметр; если идентификаторы строк не поддерживаются, применяется значение по умолчанию <code>None</code>)
<code>rowcount</code>	Количество строк, которые сформировала или затронула последняя по времени функция <code>execute*()</code>
<code>callproc(func[, args])</code>	Вызвать хранимую процедуру
<code>close()</code>	Закрыть курсор
<code>execute(op [, args])</code>	Выполнить запрос или команду базы данных
<code>executemany(op, args)</code>	Действовать по принципу применения сочетания <code>execute()</code> и <code>map()</code> ; подготовить и выполнить запрос или команду базы данных с заданными параметрами
<code>fetchone()</code>	Осуществить выборку следующей строки из результатов запроса
<code>fetchmany([size=cursor.arraysize])</code>	Осуществить выборку следующей партии строк с указанным размером из результатов запроса
<code>fetchall()</code>	Осуществить выборку всех (оставшихся) строк из результатов запроса
<code>__iter__()</code>	Создать объект итератора на основе этого курсора (применение этой функции не является обязательным; см. также <code>next()</code>)
<code>messages</code>	Сформировать список сообщений (множество кортежей), полученных из базы данных при выполнении курсора (применение этой функции не является обязательным)
<code>next()</code>	Используется итератором для выборки следующей строки из результатов запроса (применение этой функции не является обязательным; аналогично <code>fetchone()</code> , см. также <code>__iter__()</code>)
<code>nextset()</code>	Перейти к следующему результирующему набору (если поддерживается)
<code>rownumber</code>	Индекс курсора (номер строки, с отсчетом от нуля) в текущем результирующем наборе (применение этой функции не является обязательным)
<code>setinputsizes(sizes)</code>	Задать максимально допустимый размер ввода (применение этой функции является обязательным, но реализация — необязательна)
<code>setoutputsize(size[, col])</code>	Задать максимально допустимый размер буфера для осуществления выборки столбцов с большими размерами данных (применение этой функции является обязательным, но реализация — необязательна)

Наиболее важными атрибутами объектов курсора являются методы `execute*()` и `fetch*()`, которые применяются для выполнения всех запросов на обслуживание в базе данных. Атрибут данных `arraysize` позволяет задать размер по умолчанию для `fetchmany()`. Безусловно, следует всегда предусматривать закрытие курсора, а если база данных поддерживает хранимые процедуры, то имеет смысл использовать `callproc()`.

6.2.4. Объекты и конструкторы типов

Чаще всего при обеспечении взаимодействия двух различных систем наиболее уязвимым является интерфейс. Эта особенность проявляется, в частности, при преобразовании объектов Python в типы C, и наоборот. Аналогичным образом, обнаруживаются ярко выраженные различия между объектами Python и собственными объектами базы данных. При написании сценариев с использованием спецификации DB-API языка Python программист задает все параметры, передаваемые в базу данных в виде строк, но в базе данных может потребоваться преобразовать полученные значения в самые разные, но поддерживаемые типы данных, которые останутся допустимыми для любого конкретного запроса.

Иногда приходится задумываться над тем, должна ли строка Python быть преобразована в данные типа VARCHAR, TEXT, BLOB, или в бесформатный объект BINARY, или даже в объект DATE либо TIME, если предполагается, что строка действительно содержит соответствующие значения. Необходимо позаботиться о том, чтобы база данных получала входные данные в ожидаемом формате; из этого вытекает еще одно требование к DB-API: этот интерфейс должен обеспечивать создание конструкторов, формирующих специальные объекты, которые могут быть легко преобразованы в соответствующие объекты базы данных. В табл. 6.7 представлены классы, которые могут использоваться с этой целью. Значения NULL языка SQL при прямом и обратном преобразовании соответствуют объекту NULL языка Python, который обозначается как None.

Таблица 6.7. Объекты и конструкторы типов

Объект типа	Описание
Date(<i>yr, mo, dy</i>)	Объект для значения даты
Time(<i>hr, min, sec</i>)	Объект для значения времени
Timestamp(<i>yr, mo, dy, hr, min, sec</i>)	Объект для значения отметки времени
DateFromTicks(<i>ticks</i>)	Объект даты, для представления которого используется количество секунд с начала эпохи
TimeFromTicks(<i>ticks</i>)	Объект времени, для представления которого используется количество секунд с начала эпохи
TimestampFromTicks(<i>ticks</i>)	Объект отметки времени, для представления которого используется количество секунд с начала эпохи
Binary(<i>string</i>)	Объект для строкового значения binary(long)
STRING	Объект, описывающий строковые столбцы, например VARCHAR
BINARY	Объект, описывающий столбцы (long) binary, например RAW, BLOB
NUMBER	Объект, описывающий числовые столбцы
DATETIME	Объект, описывающий столбцы date/time
ROWID	Объект, описывающий столбцы идентификаторов строк

Изменения в API в связи с переходом от одной версии к другой

При пересмотре DB-API в связи с переходом от версии 1.0 (1996 год) к версии 2.0 (1999 год) было внесено несколько важных изменений.

- Из API удален обязательный модуль dbi.
- Обновлено объекты типа.

- Добавлены новые атрибуты для создания лучших привязок к базе данных.
- Переопределена семантика `callproc()` и изменено возвращаемое значение `execute()`.
- Проведено преобразование в исключения на основе классов.

После публикации версии 2.0 в 2002 году были добавлены некоторые дополнительные, необязательные расширения DB-API, о которых было сказано выше. С того времени, как была опубликована последняя версия API, в нем не произошли какие-либо другие существенные изменения. В списке рассылки DB-SIG этот API продолжает обсуждаться. Среди тем, привлекавших особое внимание в последние пять лет, было обсуждение возможности выпуска следующей версии DB-API, условно именуемой DB-API 3.0. Рассматриваемые предложения включают следующее.

- Применение такого возвращаемого значения для `nextset()`, которое было бы более удобным при наличии нового результирующего набора.
- Переход от типа данных `float` к `Decimal`.
- Повышение гибкости и разнообразие поддержки стилей параметров.
- Применение подготовленных инструкций или кеширования инструкций.
- Уточнение транзакционной модели.
- Определение роли API при решении проблем переносимости.
- Добавление возможностей проверки модулей.

Если у читателя есть свое мнение по поводу этого API или его будущего, он может принять участие в работе группы и присоединиться к обсуждению. Ниже приведены некоторые ссылки, которые могут оказаться полезными.

- <http://python.org/topics/database>
- <http://linuxjournal.com/article/2605> (outdated but historical)
- <http://wiki.python.org/moin/DbApi3>

6.2.5. Реляционные базы данных

Выше в этой главе приведен большой объем предварительных сведений, но не дан ответ на очень важный вопрос: какими интерфейсами к системам баз данных можно воспользоваться в сценарии Python? Этот вопрос аналогичен тому, на каких платформах может применяться Python. И в том и в другом случае ответ состоит в следующем: Python охватывает практически все интерфейсы и все платформы. Ниже приведен достаточно полный (но не исчерпывающий) список интерфейсов.

Коммерческие реляционные СУБД

- IBM Informix
- Sybase
- Oracle
- Microsoft SQL Server
- IBM DB2

- SAP
- Embarcadero Interbase
- Ingres

Реляционные СУБД с открытым исходным кодом

- MySQL
- PostgreSQL
- SQLite
- Gadfly

API для баз данных

- JDBC
- ODBC

Нереляционные базы данных

- MongoDB
- Redis
- Cassandra
- SimpleDB
- Tokyo Cabinet
- CouchDB
- Bigtable (через API хранилища данных ядра приложений Google)

Чтобы найти более уточненный (но не обязательно самый последний) список поддерживаемых баз данных, перейдите на следующий веб-сайт:

<http://wiki.python.org/moin/DatabaseInterfaces>

6.2.6. Базы данных и Python: адаптеры

Для каждой из поддерживаемых баз данных предусматривается один или несколько адаптеров, которые позволяют подключиться к целевой системе баз данных из сценария Python. Для некоторых баз данных, таких как Sybase, SAP, Oracle и SQL Server, количество доступных адаптеров больше по сравнению с другими. Таким образом, крайне важно определить, какой из доступных адаптеров в наибольшей степени соответствует конкретным потребностям. В связи с рассмотрением каждого доступного варианта необходимо найти ответы на следующие вопросы: насколько приемлемой является производительность адаптера, можно ли с успехом воспользоваться его документацией и (или) веб-сайтом, поддерживает ли его активное сообщество пользователей, каково общее качество и стабильность драйвера, и т.д. Необходимо учитывать, что большинство адаптеров поддерживает лишь базовые функции, без которых нельзя подключиться к базе данных. Интерес в основном представляют функции, превосходящие возможности базовых. Следует помнить, что в программе

часто приходится предусматривать применение кода более высокого уровня, связанного с многопоточковой поддержкой и управлением потоками, организовать поддержку пулов соединений с базой данных и т.д.

Если программист стремится избежать излишней работы и избавиться от необходимости заниматься каждой мелочью, например, если он предпочитает свести к минимуму объем применяемого кода SQL или количество решаемых задач администрирования базы данных, то ему целесообразно применить объектно-реляционные преобразователи, которые рассматриваются ниже.

Рассмотрим несколько примеров применения модулей адаптеров для обеспечения взаимодействия с реляционной базой данных. При этом основная сложность заключается в том, чтобы успешно установить соединение. После того как это сделано и появилась возможность использовать объекты, атрибуты и методы объектов DB-API, основной код становится примерно одинаковым, независимо от того, какой адаптер и какая реляционная СУБД используются.

6.2.7. Примеры применения адаптеров баз данных

Прежде всего рассмотрим некоторые образцы кода, которые показывают, как создать базу данных, объявить таблицу и приступить к ее использованию. Будут представлены примеры, в которых показаны адаптеры MySQL, PostgreSQL и SQLite.

MySQL

В этом примере будет использоваться база данных MySQL, а также адаптер для MySQL на языке Python, который приобрел широкую известность. Речь идет об адаптере MySQLdb, называемом также MySQL-python. Мы рассмотрим наряду с этим еще один адаптер MySQL, именуемый как MySQL Connector/Python, а затем перейдем к рассмотрению версии Python 3. В связи с описанием примеров кода, приведенных ниже, будут также затрагиваться примеры ошибок (созданных преднамеренно), чтобы дать понять, каких последствий можно ожидать и для каких ошибок может потребоваться создать обработчик.

Как правило, сначала необходимо зарегистрироваться в качестве администратора для создания базы данных и предоставления разрешений, после этого снова войти в систему в качестве обычного пользователя, как показано в следующем примере:

```
>>> import MySQLdb
>>> cxn = MySQLdb.connect(user='root')
>>> cxn.query('DROP DATABASE test')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  _mysql_exceptions.OperationalError: (1008, "Can't drop, database 'test'; database
doesn't exist")
>>> cxn.query('CREATE DATABASE test')
>>> cxn.query("GRANT ALL ON test.* to '@'localhost'")
>>> cxn.commit()
>>> cxn.close()
```

В предыдущем коде курсор не использовался. Для некоторых адаптеров предусмотрены объекты Connection, позволяющие выполнять запросы SQL с помощью метода query(), но не для всех. Мы рекомендуем либо не пользоваться этим вариантом, либо проверять применяемый адаптер для достижения полной уверенности в том, что указанная возможность существует.

Применение метода `commit()` оказалось необязательным, поскольку в базе данных MySQL по умолчанию включена автоматическая фиксация. Затем происходит повторное подключение к новой базе данных в качестве обычного пользователя, создание таблицы и выполнение обычных запросов и команд с помощью языка SQL для осуществления намеченного задания с помощью сценария Python. На этот раз применяются курсоры и связанный с ними метод `execute()`.

Следующий ряд операций взаимодействия с базой данных показывает, как создать таблицу. Попытка повторного создания таблицы (без предварительного удаления существующей) приводит к ошибке:

```
>>> cxn = MySQLdb.connect(db='test')
>>> cur = cxn.cursor()
>>> cur.execute('CREATE TABLE users(login VARCHAR(8), userid INT)')
0L
```

Теперь вставим несколько строк в базу данных и выполним к ней запрос:

```
>>> cur.execute("INSERT INTO users VALUES('john', 7000)")
1L
>>> cur.execute("INSERT INTO users VALUES('jane', 7001)")
1L
>>> cur.execute("INSERT INTO users VALUES('bob', 7200)")
1L

>>> cur.execute("SELECT * FROM users WHERE login LIKE 'j%'")
2L
>>> for data in cur.fetchall():
...     print '%s\t%s' % data
...
john      7000
jane      7001
```

Последний ряд инструкций показывает, как обновить таблицу с помощью операций обновления или удаления строк:

```
>>> cur.execute("UPDATE users SET userid=7100 WHERE userid=7001")
1L
>>> cur.execute("SELECT * FROM users")
3L
>>> for data in cur.fetchall():
...     print '%s\t%s' % data
...
john      7000
jane      7100
bob       7200
>>> cur.execute('DELETE FROM users WHERE login="bob"')
1L
>>> cur.execute('DROP TABLE users')
0L
>>> cur.close()
>>> cxn.commit()
>>> cxn.close()
```

MySQL — одна из наиболее широко применяемых в мире баз данных с открытым исходным кодом, поэтому не удивительно, что для нее предусмотрен адаптер Python.

PostgreSQL

Еще одной весьма распространенной базой данных с открытым исходным кодом является PostgreSQL. В отличие от MySQL, для Postgres предусмотрено не меньше трех адаптеров Python: `psycopg`, `PyPgSQL` и `PyGreSQL`. В свое время был также четвертый адаптер, `PyPy`, который теперь не поддерживается, после того, как его код был объединен с кодом `PyGreSQL` в 2003 году. Каждый из трех оставшихся адаптеров имеет свои особенности, преимущества и недостатки, поэтому рекомендуется приложить достаточные усилия для определения того, какой из них для вас подходит.

Необходимо отметить, что разработка `PyPgSQL` не ведется столь активно, как прежде, а для `PyGreSQL` наиболее современная версия (4.0) была выпущена в 2009 году, хотя в этой книге будут приведены примеры применения всех трех адаптеров. Это снижение активности по двум направлениям из трех явно показывает, что единственным лидером среди адаптеров PostgreSQL остается `psycopg`, поэтому ему будет посвящен последний пример применения адаптеров, приведенный в данной книге. В настоящее время выпущена вторая версия адаптера `psycopg`, поэтому, несмотря на использование в наших примерах модуля `psycopg` версии 1, который все еще доступен для загрузки, вместо него следует использовать `psycopg2`.

Положительным фактором является то, что во всех версиях адаптеров интерфейсы являются настолько похожими, что можно, допустим, создать приложение для сравнения производительности работы между ними тремя (если вас действительно интересует этот показатель). Ниже представлен код установки, предназначенный для получения объекта `Connection` для каждого адаптера.

Psycopg

```
>>> import psycopg
>>> cxn = psycopg.connect(user='pgsql')
```

PyPgSQL

```
>>> from pyPgSQL import PgSQL
>>> cxn = PgSQL.connect(user='pgsql')
```

PyGreSQL

```
>>> import pgdb
>>> cxn = pgdb.connect(user='pgsql')
```

Далее следует некоторый общий код, который применим для всех трех адаптеров:

```
>>> cur = cxn.cursor()
>>> cur.execute('SELECT * FROM pg_database')
>>> rows = cur.fetchall()
>>> for i in rows:
...     print i
>>> cur.close()
>>> cxn.commit()
>>> cxn.close()
```

Наконец, можно рассмотреть вывод, полученный при использовании каждого адаптера, обращая внимание на некоторые отличия.

PyPgSQL

```
sales
templatel
template0
```

psycopg

```
('sales', 1, 0, 0, 1, 17140, '140626', '3221366099', '', None, None)
('templatel', 1, 0, 1, 1, 17140, '462', '462', '', None, '{pgsql=C*T*/pgsql}')
('template0', 1, 0, 1, 0, 17140, '462', '462', '', None, '{pgsql=C*T*/pgsql}')
```

PyGreSQL

```
['sales', 1, 0, False, True, 17140L, '140626', '3221366099', '', None, None]
['templatel', 1, 0, True, True, 17140L, '462', '462', '', None, '{pgsql=C*T*/pgsql}']
['template0', 1, 0, True, False, 17140L, '462', '462', '', None, '{pgsql=C*T*/pgsql}']
SQLite
```

В чрезвычайно простых приложениях для создания системы постоянного хранения, как правило, достаточно использовать обычные файлы, но при разработке более сложных приложений, управляемых данными, невозможно обойтись без полнофункциональной реляционной базы данных. База данных SQLite предназначена для систем промежуточного уровня и фактически представляет собой гибрид, в котором совместно применяются файловая система и реляционная база данных. База данных SQLite является чрезвычайно простой и быстродействующей, к тому же при работе с ней не требуется применение сервера, а администрирование сводится к минимуму или вообще не требуется.

2.5

Популярность базы данных SQLite быстро растет, и она доступна на многих платформах. Введение адаптера базы данных `rusqlite` в версию Python 2.5 в качестве модуля `sqlite3` знаменует важный момент, когда впервые адаптер базы данных был включен в стандартную библиотеку Python для применения во всех выпусках.

Этот адаптер был включен в состав программного обеспечения Python не только благодаря своему предпочтительному положению по сравнению с другими базами данных и адаптерам, но и по той причине, что он является простым, позволяет использовать файлы (или оперативную память) в качестве своего опорного хранилища (как и модули DBM), не требует применения сервера и не нуждается в лицензировании. Базу данных SQLite можно было бы рассматривать просто как альтернативу другим аналогичным решениям по обеспечению постоянного хранения данных, поддерживаемым языком Python, но случилось так, что эта база данных имеет интерфейс SQL.

Наличие подобного модуля в стандартной библиотеке означает, что появилась возможность проводить ускоренную разработку приложений для баз данных

на языке Python с помощью SQLite, а затем переносить эти приложения на более мощную реляционную СУБД, такую как MySQL, PostgreSQL, Oracle или SQL Server, для применения на производстве, если в этом состоит ваша задача. Если же в дальнейшем не потребуется вся эта мощь, то вполне можно ограничиться применением sqlite3.

Тем не менее, несмотря на включение этого адаптера базы данных в стандартную библиотеку, все еще остается необходимость самому загрузить фактически применяемое программное обеспечение базы данных. После установки указанного программного обеспечения остается лишь запустить интерпретатор Python (и импортировать адаптер), чтобы получить непосредственный доступ к данным:

```
>>> import sqlite3
>>> cxn = sqlite3.connect('sqlite_test/test')
>>> cur = cxn.cursor()
>>> cur.execute('CREATE TABLE users(login VARCHAR(8),
    userid INTEGER)')
>>> cur.execute('INSERT INTO users VALUES("john", 100)')
>>> cur.execute('INSERT INTO users VALUES("jane", 110)')
>>> cur.execute('SELECT * FROM users')
>>> for eachUser in cur.fetchall():
...     print eachUser
...
(u'john', 100)
(u'jane', 110)
>>> cur.execute('DROP TABLE users')
<sqlite3.Cursor object at 0x3d4320>
>>> cur.close()
>>> cxn.commit()
>>> cxn.close()
```

После рассмотрения приведенных выше небольших примеров можно перейти к более сложным приложениям. В следующую очередь мы рассмотрим приложение, аналогичное приведенному выше для MySQL, но позволяющее выполнить несколько дополнительных операций.

- Создание базы данных (в случае необходимости).
- Создание таблицы.
- Вставка строк в таблицу.
- Обновление строк в таблице.
- Удаление строк из таблицы.
- Уничтожение таблицы.

Для этого примера будут использоваться еще две базы данных с открытым исходным кодом. В последнее время очень широкое распространение находит база данных SQLite — весьма небольшая, простая, но чрезвычайно быстродействующая база данных, поддерживающая все наиболее широко применяемые функции для работы с базами данных. Второй базой данных, рассматриваемой в этом примере, является Gadfly, которая представляет собой главным образом совместимую с SQL реляционную СУБД, написанную полностью на языке Python. (Для некоторых из наиболее важных структур данных предусмотрены модули C, но база данных Gadfly может работать и без них, хотя и медленнее.)

Прежде чем перейти к коду, сделаем несколько замечаний. И для SQLite, и для Gadfly необходимо указывать местонахождение хранимых файлов базы данных (для MySQL предусмотрены заданные по умолчанию каталоги, поэтому такая информация не требуется). Наиболее современная версия Gadfly еще не полностью совместима с DB-API 2.0, поэтому не поддерживает некоторые функциональные средства; наиболее важным из них является атрибут курсора `rowcount`, который используется в рассматриваемом примере.

6.2.8. Пример приложения на основе адаптера базы данных

В следующем примере будет показано, как использовать язык Python для получения доступа к базе данных. Чтобы обеспечить разнообразие и предоставить читателю для изучения максимально возможный объем кода, мы добавили поддержку для трех различных систем баз данных: Gadfly, SQLite и MySQL. Для того чтобы задача стала еще более интересной, вначале представим весь исходный код для версии Python 2.x без построчного описания.

Приложение действует точно по такому же принципу, как было описано в отдельных пунктах списков в предыдущем подразделе. От читателя требуется понять, как реализуются функции приложения, без его полного объяснения; отметим лишь, что нужно начать с функции `main()`, которая приведена в конце листинга. (В целях упрощения мы предусматриваем только регистрацию в качестве пользователя `root` даже в такой полноценной системе, как MySQL, которая имеет архитектуру “клиент-сервер”, хотя такая организация работы не рекомендуется для производственного приложения.) Ниже приведен исходный код рассматриваемого приложения, получившего название `ushuffle_db.py`.

```
#!/usr/bin/env python

import os
from random import randrange as rand

COLSIZ = 10
FIELDS = ('login', 'userid', 'projid')
RDBMSs = {'s': 'sqlite', 'm': 'mysql', 'g': 'gadfly'}
DBNAME = 'test'
DBUSER = 'root'
DB_EXC = None
NAMELEN = 16

tformat = lambda s: str(s).title().ljust(COLSIZ)
cformat = lambda s: s.upper().ljust(COLSIZ)

def setup():
    return RDBMSs[raw_input('
Choose a database system:

(M) ySQL
(G) adfly
(S) QLite

Enter choice: ').strip().lower()[0]]

def connect(db):
    global DB_EXC
```

```

dbDir = '%s_%s' % (db, DBNAME)

if db == 'sqlite':
    try:
        import sqlite3
    except ImportError:
        try:
            from pysqlite2 import dbapi2 as sqlite3
        except ImportError:
            return None

    DB_EXC = sqlite3
    if not os.path.isdir(dbDir):
        os.mkdir(dbDir)
    conn = sqlite3.connect(os.path.join(dbDir, DBNAME))

elif db == 'mysql':
    try:
        import MySQLdb
        import _mysql_exceptions as DB_EXC
    except ImportError:
        return None

    try:
        conn = MySQLdb.connect(db=DBNAME)
    except DB_EXC.OperationalError:
        try:
            conn = MySQLdb.connect(user=DBUSER)
            conn.query('CREATE DATABASE %s' % DBNAME)
            conn.commit()
            conn.close()
            conn = MySQLdb.connect(db=DBNAME)
        except DB_EXC.OperationalError:
            return None

elif db == 'gadfly':
    try:
        from gadfly import gadfly
        DB_EXC = gadfly
    except ImportError:
        return None

    try:
        conn = gadfly(DBNAME, dbDir)
    except IOError:
        conn = gadfly()
        if not os.path.isdir(dbDir):
            os.mkdir(dbDir)
        conn.startup(DBNAME, dbDir)

else:
    return None
return conn

def create(cur):
    try:
        cur.execute('''
            CREATE TABLE users (
                login VARCHAR(%d),

```

```

        userid INTEGER,
        projid INTEGER)
    ''' % NAMELEN)
except DB_EXC.OperationalError:
    drop(cur)
    create(cur)

drop = lambda cur: cur.execute('DROP TABLE users')

NAMES = (
    ('aaron', 8312), ('angela', 7603), ('dave', 7306),
    ('davina', 7902), ('elliott', 7911), ('ernie', 7410),
    ('jess', 7912), ('jim', 7512), ('larry', 7311),
    ('leslie', 7808), ('melissa', 8602), ('pat', 7711),

    ('serena', 7003), ('stan', 7607), ('faye', 6812),
    ('amy', 7209), ('mona', 7404), ('jennifer', 7608),
)

def randName():
    pick = set(NAMES)
    while pick:
        yield pick.pop()

def insert(cur, db):
    if db == 'sqlite':
        cur.executemany("INSERT INTO users VALUES(?, ?, ?)",
            [(who, uid, rand(1,5)) for who, uid in randName()])
    elif db == 'gadfly':
        for who, uid in randName():
            cur.execute("INSERT INTO users VALUES(?, ?, ?)",
                (who, uid, rand(1,5)))
    elif db == 'mysql':
        cur.executemany("INSERT INTO users VALUES(%s, %s, %s)",
            [(who, uid, rand(1,5)) for who, uid in randName()])

getRC = lambda cur: cur.rowcount if hasattr(cur, 'rowcount') else -1

def update(cur):
    fr = rand(1,5)
    to = rand(1,5)
    cur.execute(
        "UPDATE users SET projid=%d WHERE projid=%d" % (to, fr))
    return fr, to, getRC(cur)

def delete(cur):
    rm = rand(1,5)
    cur.execute('DELETE FROM users WHERE projid=%d' % rm)
    return rm, getRC(cur)

def dbDump(cur):
    cur.execute('SELECT * FROM users')
    print '\n%s' % ''.join(map(cformat, FIELDS))
    for data in cur.fetchall():
        print ''.join(map(tformat, data))

def main():
    db = setup()

```

```

print '*** Connect to %r database' % db
cxn = connect(db)
if not cxn:
    print 'ERROR: %r not supported or unreachable, exiting' % db
    return
cur = cxn.cursor()

print '\n*** Create users table (drop old one if appl.)'
create(cur)

print '\n*** Insert names into table'
insert(cur, db)
dbDump(cur)

print '\n*** Move users to a random group'
fr, to, num = update(cur)
print '\t(%d users moved) from (%d) to (%d)' % (num, fr, to)
dbDump(cur)

print '\n*** Randomly delete group'
rm, num = delete(cur)
print '\t(group %d; %d users removed)' % (rm, num)
dbDump(cur)

print '\n*** Drop users table'
drop(cur)
print '\n*** Close cxns'
cur.close()
cxn.commit()
cxn.close()

if __name__ == '__main__':
    main()

```

Смеем заверить читателя, что это приложение вполне работоспособное. Его можно загрузить с веб-сайта данной книги, чтобы проверить его работу на практике. Однако, прежде чем приступить к изучению этого приложения, необходимо выполнить некоторую подготовительную работу. Тем не менее на данном этапе мы также не приводим построчное описание.

3.x

Не следует беспокоиться; мы подробно опишем другой пример, а данный пример предназначен для демонстрации еще одного варианта переноса приложения в версию Python 3, который показывает, как можно создавать сценарии, выполняемые и в Python 2, и в Python 3, применяя один и тот же файл исходного кода с расширением .py и не прибегая к формальному преобразованию в другую версию с помощью таких инструментов, как 2to3 и 3to2. Приложение, оформленное таким образом, будет полностью рассмотрено в примере 6.1. Кроме того, следует отметить, что атрибуты, представленные в этом примере, мы будем переносить и повторно использовать во всех оставшихся примерах данной главы, причем такой перенос будет происходить и в примеры с объектно-реляционными преобразователями, и в примеры с нереляционными базами данных.

Перенос в версию Python 3

В главе книги *Core Python Language Fundamentals*, посвященной описанию передовой практики, приведен целый ряд полезных рекомендаций по переносу приложений в другие версии, но сейчас мы хотим поделиться некоторыми конкретными советами и реализовать их при использовании сценария `ushuffle_db.py`.

При переносе между версиями Python 2 и 3 одной из важных проблем становится то, что инструкция `print`, применявшаяся в версии Python 2, становится в Python 3 встроенной функцией (built-in function — BIF). Вместо непосредственного использования того или другого можно также заменить оператор печати так называемым посредником с помощью функции `distutils.log.warn()`; по крайней мере, такая возможность была предусмотрена во время написания данной книги. При этом применяемая конструкция является одинаковой в Python 2 и 3, иными словами, при переходе между версиями не требуются какие-либо изменения. Кроме того, чтобы при изучении кода не возникала путаница, переименуем эту функцию в рассматриваемом приложении в `printf()`, поскольку в языке C/C++ уже давно применяются аналоги `print/print()`, предназначенные для печати. См. также относящееся к этой теме упражнение, приведенное в конце данной главы.

Второй совет относится к встроенной функции `raw_input()` версии Python 2. В версии Python 3 имя этой функции изменено на `input()`. Сложившаяся ситуация дополнительно усложнена в связи с тем фактом, что в версии Python 2 была предусмотрена своя функция `input()`, которая оказалась источником нарушения безопасности и была удалена из состава языка. Иными словами, вместо нее стала применяться функция `raw_input()`, которая переименована в `input()` в версии Python 3. Следуя по намеченному ранее пути, т.е. применяя имена аналогичных функций из языка C/C++, назовем эту функцию в нашем приложении как `scanf()`.

Следующий совет должен послужить напоминанием об изменениях в синтаксисе обработки исключительных ситуаций. Эта тема подробно рассматривается в главе "Ошибки и исключения" книг *Core Python Language Fundamentals* и *Core Python Programming*. Более подробные сведения об этом обновлении можно получить в указанных главах, но на данный момент наиболее важное изменение, о котором необходимо знать, состоит в следующем.

Старый формат: `except Exception, instance`

Новый формат: `except Exception as instance`

Это имеет значение, только если возникает необходимость сохранить экземпляр исключения, в связи с тем, что требуется выяснить причину исключения. Если эти данные вас не интересуют или не предусмотрено их использование, то можно пропустить этап сохранения экземпляра исключения. Вполне допустимо использование следующей конструкции: `except Exception`.

В таком случае в обеих версиях, Python 2 и Python 3, применяется одинаковый синтаксис. В предыдущих изданиях данной книги использовалась конструкция `except Exception, e`. В настоящем издании часть этой конструкции `", e"` была полностью удалена, так как было решено не заменять ее на `"as e"` для упрощения переноса с одной версии на другую.

Наконец, последнее изменение, которое должно быть сделано, относится непосредственно к рассматриваемому примеру, в отличие от других изменений, которые укладываются в рамки общих рекомендаций по переносу. Ко времени написания этой книги основной адаптер MySQL-Python на языке C, который чаще называют по имени его пакета, `MySQLdb`, еще не был перенесен в версию Python 3. Тем не менее

предусмотрен еще один адаптер MySQL, который носит название MySQL Connector/Python и имеет имя пакета `mysql.connector`.

Адаптер MySQL Connector/Python реализует клиентский протокол MySQL исключительно на языке Python, поэтому нет необходимости применять какие-либо библиотеки MySQL или откомпилированные модули; а лучше всего то, что этот адаптер уже перенесен в версию Python 3. Почему это для нас так важно? С его помощью пользователи Python 3 могут получить доступ к базам данных MySQL, и этого достаточно!

В ходе внесения всех этих изменений и дополнений в сценарий `ushuffle_db.py` стало ясно, что проще рассматривать универсальную версию этого приложения, `ushuffle_dbU.py`, которая приведена в примере 6.1.

Пример 6.1. Применение адаптера базы данных (`ushuffle_dbU.py`)

В этом сценарии выполняются некоторые несложные операции с использованием ряда баз данных (MySQL, SQLite, Gadgetfly). Он работает и в версии Python 2, и в версии Python 3 без каких-либо изменений, а его компоненты будут повторно использоваться в следующих разделах данной главы.

```

1  #!/usr/bin/env python
2
3  from distutils.log import warn as printf
4  import os
5  from random import randrange as rand
6
7  if isinstance(__builtins__, dict) and 'raw_input' in __builtins__:
8      scanf = raw_input
9  elif hasattr(__builtins__, 'raw_input'):
10     scanf = raw_input
11  else:
12     scanf = input
13
14  COLSIZ = 10
15  FIELDS = ('login', 'userid', 'projid')
16  RDBMSs = {'s': 'sqlite', 'm': 'mysql', 'g': 'gadgetfly'}
17  DBNAME = 'test'
18  DBUSER = 'root'
19  DB_EXC = None
20  NAMELEN = 16
21
22  tformat = lambda s: str(s).title().ljust(COLSIZ)
23  cformat = lambda s: s.upper().ljust(COLSIZ)
24
25  def setup():
26     return RDBMSs[raw_input('
27 Choose a database system:
28
29 (M)ySQL
30 (G)adfly
31 (S)QLite
32
33 Enter choice: ').strip().lower()[0]]
34
35  def connect(db, DBNAME):
36     global DB_EXC

```

```
37     dbDir = '%s_%s' % (db, DBNAME)
38
39     if db == 'sqlite':
40         try:
41             import sqlite3
42         except ImportError:
43             try:
44                 from pysqlite2 import dbapi2 as sqlite3
45             except ImportError:
46                 return None
47
48         DB_EXC = sqlite3
49         if not os.path.isdir(dbDir):
50             os.mkdir(dbDir)
51         cxn = sqlite3.connect(os.path.join(dbDir, DBNAME))
52
53     elif db == 'mysql':
54         try:
55             import MySQLdb
56             import _mysql_exceptions as DB_EXC
57
58         try:
59             cxn = MySQLdb.connect(db=DBNAME)
60         except DB_EXC.OperationalError:
61             try:
62                 cxn = MySQLdb.connect(user=DBUSER)
63                 cxn.query('CREATE DATABASE %s' % DBNAME)
64                 cxn.commit()
65                 cxn.close()
66                 cxn = MySQLdb.connect(db=DBNAME)
67             except DB_EXC.OperationalError:
68                 return None
69         except ImportError:
70             try:
71                 import mysql.connector
72                 import mysql.connector.errors as DB_EXC
73             try:
74                 cxn = mysql.connector.Connect(**{
75                     'database': DBNAME,
76                     'user': DBUSER,
77                 })
78             except DB_EXC.InterfaceError:
79                 return None
80         except ImportError:
81             return None
82
83     elif db == 'gadfly':
84         try:
85             from gadfly import gadfly
86             DB_EXC = gadfly
87         except ImportError:
88             return None
89
90         try:
91             cxn = gadfly(DBNAME, dbDir)
92         except IOError:
93             cxn = gadfly()
94             if not os.path.isdir(dbDir):
```

```

95         os.mkdir(dbDir)
96         cxn.startup(DENAME, dbDir)
97     else:
98         return None
99     return cxn
100
101 def create(cur):
102     try:
103         cur.execute('''
104             CREATE TABLE users (
105                 login VARCHAR(%d),
106                 userid INTEGER,
107                 projid INTEGER)
108             ''' % NAMELEN)
109     except DB_EXC.OperationalError, e:
110         drop(cur)
111         create(cur)
112
113 drop = lambda cur: cur.execute('DROP TABLE users')
114
115 NAMES = (
116     ('aaron', 8312), ('angela', 7603), ('dave', 7306),
117     ('davina', 7902), ('elliott', 7911), ('ernie', 7410),
118     ('jess', 7912), ('jim', 7512), ('larry', 7311),
119     ('leslie', 7808), ('melissa', 8602), ('pat', 7711),
120     ('serena', 7003), ('stan', 7607), ('faye', 6812),
121     ('amy', 7209), ('mona', 7404), ('jennifer', 7608),
122 )
123
124 def randName():
125     pick = set(NAMES)
126     while pick:
127         yield pick.pop()
128
129 def insert(cur, db):
130     if db == 'sqlite':
131         cur.executemany("INSERT INTO users VALUES(?, ?, ?)",
132             [(who, uid, rand(1,5)) for who, uid in randName()])
133     elif db == 'gadfly':
134         for who, uid in randName():
135             cur.execute("INSERT INTO users VALUES(?, ?, ?)",
136                 (who, uid, rand(1,5)))
137     elif db == 'mysql':
138         cur.executemany("INSERT INTO users VALUES(%s, %s, %s)",
139             [(who, uid, rand(1,5)) for who, uid in randName()])
140
141 getRC = lambda cur: cur.rowcount if hasattr(cur, 'rowcount') else -1
142
143 def update(cur):
144     fr = rand(1,5)
145     to = rand(1,5)
146     cur.execute(
147         "UPDATE users SET projid=%d WHERE projid=%d" % (to, fr))
148     return fr, to, getRC(cur)
149
150 def delete(cur):
151     rm = rand(1,5)
152     cur.execute('DELETE FROM users WHERE projid=%d' % rm)

```

```

153 return rm, getRC(cur)
154
155 def dbDump(cur):
156     cur.execute('SELECT * FROM users')
157     printf('\n%s' % ''.join(map(cformat, FIELDS)))
158     for data in cur.fetchall():
159         printf(''.join(map(tformat, data)))
160
161 def main():
162     db = setup()
163     printf('*** Connect to %r database' % db)
164     cxn = connect(db)
165     if not cxn:
166         printf('ERROR: %r not supported or unreachable, exit' % db)
167         return
168     cur = cxn.cursor()
169
170     printf('\n*** Creating users table')
171     create(cur)
172
173     printf('\n*** Inserting names into table')
174     insert(cur, db)
175     dbDump(cur)
176
177     printf('\n*** Randomly moving folks')
178     fr, to, num = update(cur)
179     printf('\t(%d users moved) from (%d) to (%d)' % (num, fr, to))
180     dbDump(cur)
181
182     printf('\n*** Randomly choosing group')
183     rm, num = delete(cur)
184     printf('\t(group #%d; %d users removed)' % (rm, num))
185     dbDump(cur)
186
187     printf('\n*** Dropping users table')
188     drop(cur)
189     printf('\n*** Close cxns')
190     cur.close()
191     cxn.commit()
192     cxn.close()
193
194 if __name__ == '__main__':
195     main()

```

Построчное объяснение

Строки 1–32

В первой части этого сценария осуществляется импорт необходимых модулей, создаются некоторые глобальные константы (размер столбца для отображения и набор поддерживаемых баз данных) и применяются функции `tformat()`, `cformat()` и `setup()`.

Вслед за инструкциями `import` расположен код, заслуживающий дополнительных пояснений (строки 7–12). В нем осуществляется поиск необходимой функции, для которой создается псевдоним `scanf()`, выбранный для использования в качестве

назначенной нами пользовательской функции для ввода данных в командной строке. Описать назначение ключевых слов `elif` и `else` несложно: с их помощью производится дополнительная проверка для определения того, предусмотрена ли в используемой версии встроенная функция `raw_input()`. Если эта функция существует, то работа ведется в версии Python 2 (или 1) и необходимо использовать именно ее. В противном случае для выполнения сценария применяется версия Python 3, поэтому следует использовать новое имя — `input()`.

Еще одним источником усложнения служит инструкция `if`. В нашем приложении единственным модулем является `__builtin__`. Если происходит импорт этого модуля (как в версии Python 2), то имя `__builtin__` соответствует словарю, `dict`. Применяемое условное выражение по существу указывает, что если был выполнен импорт модуля `__builtin__`, то следует проверить, содержится ли в этом словаре имя `raw_input`; в противном случае `__builtin__` представляет собой модуль, поэтому в инструкции `if` происходит переход к ветвям `elif` и `else`. Остается лишь надеяться, что это описание достаточно понятное!

Что касается функций `tformat()` и `cformat()`, то первая представляет собой строку формата для отображения заголовков; например, `tformat` означает средство форматирования регистра заголовка. (В этом состоит лишь удобный способ извлечения из базы данных имен, которые могут полностью состоять из строчных букв (как в рассматриваемом примере), иметь согласно правилам прописную первую букву, состоять полностью из прописных букв и т.д.; таким образом производится унификация форматов всех имен. Последняя функция имеет имя, которое указывает на ее назначение как на средство форматирования CAPS.) Остается лишь осуществить выборку имен всех столбцов и преобразовать полученные строки в заголовки столбцов путем вызова метода `str.upper()`.

Оба средства форматирования выравнивают формируемые ими выходные данные по левому краю и ограничивают ширину этих данных десятью символами, поскольку не предполагается, что для данных потребуется большая ширина, по крайней мере, указанное требование реализовано в применяемом нами образце данных. Во всяком случае, если читателю потребуется задать какое-то другое значение, то ему достаточно установить такую величину `COLSIZ`, которая подходит для применяемых им данных. Проще написать данный код с использованием конструкций `lambda`, а не традиционных функций, хотя читатель, безусловно, может применить тот вариант, который его больше устраивает.

Можно возразить, что слишком большие усилия затрачиваются на всю эту подготовку, притом что функция `scanf()` в составе `setup()` всего лишь выводит для пользователя приглашение выбрать реляционную СУБД, предназначенную для использования в каждом конкретном сеансе выполнения этого сценария (или в тех сценариях, которые будут созданы на его основе в последней части данной главы). Однако замысел этого кода состоит в том, чтобы показать удобные конструкции, которые могут быть применены пользователем при решении других задач. Этот сценарий предназначен скорее для использования в экспериментах, а не на производстве.

В данной книге уже рассматривались пользовательские функции вывода; как было указано выше, можно воспользоваться функцией `distutils.log.warn()` для переключения между инструкцией `print`, применяемой в версии Python 2, и функцией `print()` для Python 3. В рассматриваемом приложении необходимая функция импортируется как `printf()` (строка 3).

Большинство объявлений констант фактически не требуют пояснений. Дело обстоит иначе лишь по отношению к переменной `DB_EXC`, имя которой является

сокращением от DataBase EXception (исключение базы данных). Эта переменная будет в конечном итоге присвоена модулю исключения базы данных, относящемуся к конкретной системе баз данных, которую пользователь выберет как применяемую для этого приложения. Иными словами, если пользователь выберет MySQL, то DB_EXC примет значение `_mysql_exceptions` и т.д. Если бы данное приложение создавалось как в большей степени объектно-ориентированное, то можно было бы предусмотреть класс, в котором эта переменная является просто атрибутом экземпляра, таким как `self.db_exc_module`.

Строки 35–99

В данном сценарии многое из того, что требуется для обеспечения единообразного доступа к базе данных, предусмотрено в функции `connect()`. В начале каждого раздела (под “разделом” здесь подразумевается каждая конструкция `if`, относящаяся к базе данных) предпринимается попытка загрузить соответствующие модули базы данных. Если подходящий модуль не обнаруживается, возвращается значение `None`, указывающее на то, что система баз данных не поддерживается.

Весь прочий код, выполняемый после установления соединения, не зависит от базы данных и адаптера и должен действовать правильно применительно к любой базе данных, с которой установлено соединение. (Единственным исключением в рассматриваемом сценарии является функция `insert()`.) Заслуживает внимания то, что во всех трех подразделах этого участка кода допустимое соединение должно быть возвращено как значение `схп`.

Если выбрана база данных SQLite, предпринимается попытка загрузить адаптер базы данных. Вначале будет сделана попытка загрузить модуль `sqlite3` из стандартной библиотеки (Python 2.5 и последующие версии). Если эта операция оканчивается неудачей, осуществляется поиск пакета `pysqlite2` сторонних разработчиков. Такое действие предпринимается для поддержки версий 2.4.x и более старых систем с установленным адаптером `pysqlite`. Если обнаруживается та или другая возможность создания соединения, то проверяется, существует ли указанный каталог, поскольку эта база данных действует по принципу файловой системы. (Может быть также выбран вариант с созданием базы данных в оперативной памяти, для чего необходимо подставить `:memory:` в качестве имени файла.) После выполнения вызова `connect()` к базе данных SQLite используется существующая база данных или создается новая, если заданное значение пути приводит в каталог, в котором отсутствует база данных.

В базе данных MySQL для файлов базы данных используются каталоги, заданные по умолчанию, поэтому пользователь не обязан указывать расположение этих файлов. Наиболее широко применяемым адаптером MySQL является пакет `MySQLdb`, поэтому попытаемся в первую очередь импортировать именно его. Как и в случае с базой данных SQLite, для MySQL также предусмотрен “план B”, для которого хорошо подходит пакет `mysql.connector`, поскольку он является совместимым и с версией Python 2, и с версией Python 3. Если не обнаруживается ни тот ни другой пакет, это означает, что база данных MySQL не поддерживается, и возвращается значение `None`.

Последней базой данных, поддерживаемой рассматриваемым приложением, является Gadfly. (Ко времени написания настоящей книги база данных Gadfly была в основном, но не полностью совместимой с DB-API, что учитывается в этом приложении.) Для ее запуска используется механизм, аналогичный применяемому для базы данных SQLite: для начала выполнения применяется каталог, в котором должны находиться файлы базы данных. Если дела обстоят именно так, то проблем не возникает. В противном случае необходимо пройти по обходному маршруту, чтобы запустить

новую базу данных. (Неизвестно, почему разработчики базы данных Gadfly приняли такое решение. Возможно, функциональные средства запуска `startup()` должны были быть непосредственно внесены в конструктор `gadfly.gadfly()`.)

Строки 101–113

Функция `create()` создает новую таблицу `users` в рассматриваемой базе данных. Если возникает ошибка, то почти всегда по той причине, что таблица уже существует. Если дело обстоит именно так, то таблица уничтожается и создается повторно путем рекурсивного вызова этой функции. Этот код несет в себе предпосылки нарушения в работе, поскольку, если создание таблицы заново все равно оканчивается неудачей, возникает бесконечная рекурсия, ограничиваемая только тем, когда произойдет исчерпание приложением пределов свободной памяти. Этот недостаток будет устранен в одном из упражнений в конце данной главы.

Для удаления таблицы из базы данных применяется однострочная функция `drop()`, оформленная как лямбда-функция с помощью ключевого слова `lambda`.

Строки 115–127

Отличительной особенностью следующих блоков кода является наличие набора констант `NAMES` и идентификаторов пользователей, за которыми следует генератор `randName()`. Набор `NAMES` — это кортеж, который должен быть преобразован в данные типа `set` для использования в функции `randName()`, поскольку в этом генераторе происходит модификация кортежей с удалением каждый раз по одному имени, которое происходит до тех пор, пока еще остаются не удаленные имена. Такую организацию работы принято называть деструктивной; она часто используется в рассматриваемом приложении, поэтому лучше задать `NAMES` как канонический источник данных и просто копировать его содержимое в другую структуру данных, уничтожая ее после каждого использования генератора.

Строки 129–139

Кроме указанного выше, местом расположения кода, зависящего от базы данных, является функция `insert()`. Различия в этой функции обусловлены тем, что в каждой базе данных вставка данных происходит немного иначе по сравнению с другими базами данных. Например, адаптеры для SQLite и MySQL являются совместимыми с DB-API, поэтому в обоих из них объекты курсора имеют функцию `executemany()`, тогда как адаптер для Gadfly таковым не является, а это означает, что вставка строк должна осуществляться одна за другой.

Еще одно различие обусловлено тем, что для SQLite и Gadfly применяется стиль параметров `qmark`, а для MySQL — `format`. По этой причине строки формата также выглядят по-разному. Впрочем, внимательный анализ показывает, что сами параметры создаются во многом одинаково.

Применяемый для этого код является таковым: каждой паре “имя–идентификатор пользователя” ставится в соответствие отдельная проектная группа (которая задана с помощью идентификатора проекта, или `projid`). Выбор идентификатора проекта осуществляется случайным образом из четырех различных групп (`randrange(1, 5)`).

Строка 141

Эта единственная строка представляет условное выражение (которая трактуется следующим образом: трехместная операция Python), которое возвращает значение

rowcount для последней операции (соответствующее количеству измененных строк), а если объект курсора не поддерживает этот атрибут (т.е. не является совместимым с DB-API), возвращается значение 1.

2.5

Условные выражения были впервые введены в версии Python 2.5, поэтому, если используется версия 2.4.x или предшествующая ей, необходимо будет выполнить обратное преобразование в конструкцию в старом стиле:

```
getRC = lambda cur: (hasattr(cur, 'rowcount') \
    and [cur.rowcount] or [-1])[0]
```

Если на первый взгляд эта строка кода кажется непонятной, не следует беспокоиться. Вам потребуется просмотреть документ с вопросами и ответами (FAQ), чтобы узнать, почему применяется такая конструкция, и понять, по какой причине в конечном итоге в язык Python были введены условные выражения в версии 2.5. Если вы сумеете разобраться в этом, можете считать, что овладели надежными знаниями об объектах Python и их логических значениях.

Строки 143–153

Функции `update()` и `delete()` применяются для выборки объектов из одной группы случайным образом. Если операцией является обновление, то объекты перемещаются из текущей группы в другую (также выбранную случайным образом), а в случае удаления происходит полное удаление объектов.

Строки 155–159

Функция `dbDump()` извлекает все строки из базы данных, форматирует их для печати и отображает для пользователя. Для форматирования вывода применяются функции `cformat()` (для отображения заголовков столбцов) и `tformat()` (для форматирования каждой строки с данными о пользователе).

Прежде всего следует отметить, что выборка данных осуществляется после применения оператора `SELECT` с помощью метода `fetchall()`. Таким образом, при обработке в цикле данных о каждом пользователе берется содержимое трех столбцов (`login`, `userid`, `projid`) и передается в функцию `tformat()` с помощью функции `map()` для представления этих данных в виде строк (если это преобразование еще не выполнено), затем данные форматируются как предназначенные для вывода в заголовке, после чего форматируется полная строка, предназначенная для заполнения столбцов `COLSIZ` с выравниванием влево (правая часть заполняется пробелами).

Строки 161–195

Все эти действия выполняются в рамках функции `main()`. В ней осуществляют отдельные вызовы каждой описанной выше функции, от которой зависит работа сценария (при условии, что не произошло преждевременное завершение работы по той причине, что не найден адаптер базы данных или не удалось создать соединение, как показано в строках 164–166). Основная часть приведенного здесь кода не нуждается в особых пояснениях, тем более потому, что в этой части программы находятся операторы вывода. В этой последней части кода происходит завершение работы с курсором и соединением.

6.3. Объектно-реляционные преобразователи

Как было указано в предыдущем разделе, в наши дни программист может воспользоваться широким разнообразием различных систем баз данных, и для большинства из них имеются интерфейсы на языке Python, с помощью которых эти базы данных можно успешно поставить себе на службу. Единственным недостатком, связанным с использованием этих систем, является необходимость владения языком SQL. Поэтому программисты, которые с большей уверенностью могут манипулировать объектами Python, а не запросами SQL, но все равно желают использовать реляционную базу данных в качестве серверной части в своем приложении для работы с данными, могут предпочесть использование объектно-реляционных преобразователей.

6.3.1. Применение объектов вместо запросов SQL

Создатели объектно-реляционных преобразователей возвели несколько дополнительных уровней абстракции над уровнем языка SQL и реализовали объекты Python, которыми можно манипулировать для выполнения тех же задач, но не формируя строки на языке SQL. Некоторые системы обеспечивают большую гибкость, если при работе с ними применяются определенные операторы SQL, но чаще всего удастся почти полностью избавиться от необходимости применять какой-либо обычный код SQL.

Таблицы базы данных магическим образом преобразуются в классы Python, в которых столбцы и характеристики данных представлены как атрибуты, а для выполнения операций с базой данных применяются методы. Настройка приложения на работу с объектно-реляционным преобразователем в определенной степени напоминает подготовку к использованию стандартного адаптера базы данных. При использовании объектно-реляционных преобразователей управление значительной частью операций осуществляется не самим программистом, а программным обеспечением, поэтому фактически реализация многих функций становится сложнее или требует большего количества строк кода, чем при непосредственном использовании адаптера. Тем не менее можно надеяться на то, что программист благодаря использованию объектно-реляционного преобразователя сумеет добиться повышения производительности своего труда, поэтому небольшое увеличение объема работы будет вполне оправданным.

6.3.2. Язык Python и объектно-реляционные преобразователи

В число наиболее широко применяемых в наши дни объектно-реляционных преобразователей Python входят SQLAlchemy (<http://sqlalchemy.org>) и SQLAlchemy (<http://sqlalchemy.org>) и SQLAlchemy (<http://sqlalchemy.org>). В данной главе будут приведены примеры применения того и другого, поскольку применяемые в них системы немного отличаются из-за несовпадений в подходах, но после освоения работы с ними вам будет гораздо проще переходить от одного объектно-реляционного преобразователя к другому.

Среди других примеров объектно-реляционных преобразователей Python можно назвать Storm, PyDO/PyDO2, PDO, Dejavu, PDO, Durus, QLine и ForgetSQL. Более крупные системы на основе веб-интерфейса могут также иметь собственные компоненты объектно-реляционных преобразователей, такие как WebWare MiddleKit и API базы данных Django. Однако следует учитывать, что даже широко применяемые другими программные средства могут оказаться не самими лучшими для вашего приложения. И наоборот, для вашего приложения может идеально подойти ПО, которое лишь упоминается, но не рассматривается подробно в данной книге.

Установка и настройка

Ни SQLAlchemy, ни SQLAlchemy не входят в стандартную библиотеку, поэтому данное программное обеспечение необходимо загрузить и установить самостоятельно. (Как правило, с этой задачей удастся легко справиться с помощью инструмента `pip` или `easy_install`.)

2.5

Во времени написания этой книги все пакеты программ, описанные в данной главе, были доступны в версии Python 2, а для версии Python 3 были приспособлены только пакет SQLAlchemy, база данных SQLite и адаптер MySQL Connector/Python. Пакет `sqlite3` вошел в состав стандартной библиотеки, начиная с Python 2.5 и последующих версий, а также Python 3.x, поэтому не нужно предпринимать никаких действий, если не используется версия 2.4 или предшествующая ей.

3.x

Если установка должна быть выполнена на компьютере, где имеется только версия Python 3, то потребуется вначале загрузить пакет `Distribute` (который включает программу `easy_install`). Откройте веб-браузер (или вызовите на выполнение команду `curl`, если она установлена), загрузите инсталляционный файл (который находится по адресу http://python-distribute.org/distribute_setup.py) и установите пакет SQLAlchemy с помощью программы `easy_install`. Ниже показано, как может выглядеть весь этот процесс на персональном компьютере с операционной системой Windows.

```
C:\WINDOWS\Temp>C:\Python32\python distribute_setup.py
Extracting in c:\docume~1\wesley\locals~1\temp\tmp8mccdr
Now working in c:\docume~1\wesley\locals~1\temp\tmp8mccdr\distribute-0.6.21
Installing Distribute
warning: no files found matching 'Makefile' under directory 'docs'
warning: no files found matching 'indexsidebar.html' under directory 'docs'
creating build
creating build\src
:
Installing easy_install-3.2.exe script to C:\python32\Scripts
```

```
Installed c:\python32\lib\site-packages\distribute-0.6.21-py3.2.egg
Processing dependencies for distribute==0.6.21
Finished processing dependencies for distribute==0.6.21
After install bootstrap.
Creating C:\python32\Lib\site-packages\setuptools-0.6c11-py3.2.egg-info
Creating C:\python32\Lib\site-packages\setuptools.pth
```

```
C:\WINDOWS\Temp>
C:\WINDOWS\Temp>C:\Python32\Scripts\easy_install sqlalchemy
Searching for sqlalchemy
Reading http://pypi.python.org/simple/sqlalchemy/
Reading http://www.sqlalchemy.org
Best match: SQLAlchemy 0.7.2
Downloading http://pypi.python.org/packages/source/S/SQLAlchemy/SQLAlchemy-0.7.2.tar.gz
#md5=b84a26ae2e5de6f518d7069b29bf8f72
:
Adding sqlalchemy 0.7.2 to easy-install.pth file
Installed c:\python32\lib\site-packages\sqlalchemy-0.7.2-py3.2.egg
Processing dependencies for sqlalchemy
Finished processing dependencies for sqlalchemy
```

6.3.3. Пример базы данных с описанием должностей сотрудников

Рассмотрим, как перенести наше приложение, в котором осуществляется перестановка пользователей `ushuffle_db.py`, в среду SQLAlchemy и SQLAlchemy. И в том и в другом случае в качестве сервера базы данных применяется MySQL. Заслуживает внимания то, что объекты реализованы с помощью классов, поскольку при использовании объектно-реляционных преобразователей необходимо обеспечить работу с объектами, в противоположность тому, что в адаптере базы данных применяется бесформатный код SQL. В обоих примерах осуществляется импорт набора имен `NAMES` и средства случайного выбора имен из сценария `ushuffle_db.py`. Это сделано для предотвращения повсеместного копирования и вставки одного и того же кода, поскольку целесообразнее обеспечить повторное использование кода.

6.3.4. SQLAlchemy

Начнем с пакета SQLAlchemy, поскольку его интерфейс в большей степени напоминает интерфейс для работы с SQL, чем SQLAlchemy. Интерфейс SQLAlchemy проще, в большей степени соответствует стилю сценариев Python и является более быстродействующим, тогда как интерфейс SQLAlchemy позволяет добиться действительно качественной абстракции, с проникновением в мир объектов, а также обеспечивает большую гибкость при использовании бесформатного кода SQL, если в этом возникает необходимость.

Как показывают примеры 6.2 и 6.3, сценарии, полученные в результате переноса созданных ранее приложений для перестановки пользователей с применением обоих объектно-реляционных преобразователей, весьма напоминают друг друга с точки зрения установки, организации доступа и общего количества строк кода. Кроме того, в них происходит заимствование одинаковых наборов функций и констант из сценария `ushuffle_db.py`.

Пример 6.2. Объектно-реляционный преобразователь SQLAlchemy (`ushuffle_sad.py`)

Отличительной особенностью этого приложения для перестановки пользователей, совместимого с версиями Python 2.x и Python 3.x, является то, что в нем используется объектно-реляционный преобразователь SQLAlchemy в сочетании с базой данных MySQL или SQLite, которая применяется в качестве серверной части.

```

1  #!/usr/bin/env python
2
3  from distutils.log import warn as printf
4  from os.path import dirname
5  from random import randrange as rand
6  from sqlalchemy import Column, Integer, String, create_engine, exc, orm
7  from sqlalchemy.ext.declarative import declarative_base
8  from ushuffle_dbU import DBNAME, NAMELEN, randName,
   FIELDS, tformat, cformat, setup
9
10 DSNs = {
11     'mysql': 'mysql://root@localhost/%s' % DBNAME,
12     'sqlite': 'sqlite:///memory:',
13 }
14
15 Base = declarative_base()
```

```
16 class Users(Base):
17     __tablename__ = 'users'
18     login = Column(String(NAMELEN))
19     userid = Column(Integer, primary_key=True)
20     projid = Column(Integer)
21     def __str__(self):
22         return ''.join(map(tformat,
23                             (self.login, self.userid, self.projid)))
24
25 class SQLAlchemyTest(object):
26     def __init__(self, dsn):
27         try:
28             eng = create_engine(dsn)
29         except ImportError:
30             raise RuntimeError()
31
32         try:
33             eng.connect()
34         except exc.OperationalError:
35             eng = create_engine(dirname(dsn))
36             eng.execute('CREATE DATABASE %s' % DBNAME).close()
37             eng = create_engine(dsn)
38
39         Session = orm.sessionmaker(bind=eng)
40         self.ses = Session()
41         self.users = Users.__table__
42         self.eng = self.users.metadata.bind = eng
43
44     def insert(self):
45         self.ses.add_all(
46             Users(login=who, userid=userid, projid=rand(1,5)) \
47             for who, userid in randName()
48         )
49         self.ses.commit()
50
51     def update(self):
52         fr = rand(1,5)
53         to = rand(1,5)
54         i = -1
55         users = self.ses.query(
56             Users).filter_by(projid=fr).all()
57         for i, user in enumerate(users):
58             user.projid = to
59         self.ses.commit()
60         return fr, to, i+1
61
62     def delete(self):
63         rm = rand(1,5)
64         i = -1
65         users = self.ses.query(
66             Users).filter_by(projid=rm).all()
67         for i, user in enumerate(users):
68             self.ses.delete(user)
69         self.ses.commit()
70         return rm, i+1
71
72     def dbDump(self):
73         printf('\n%s' % ''.join(map(cformat, FIELDS)))
74         users = self.ses.query(Users).all()
```

```
75     for user in users:
76         printf(user)
77     self.ses.commit()
78
79     def __getattr__(self, attr):    # удаление или создание
80         return getattr(self.users, attr)
81
82     def finish(self):
83         self.ses.connection().close()
84
85 def main():
86     printf('*** Connect to %r database' % DBNAME)
87     db = setup()
88     if db not in DSNS:
89         printf('\nERROR: %r not supported, exit' % db)
90         return
91
92     try:
93         orm = SQLAlchemyTest(DSNS[db])
94     except RuntimeError:
95         printf('\nERROR: %r not supported, exit' % db)
96         return
97
98     printf('\n*** Create users table (drop old one if appl.)')
99     orm.drop(checkfirst=True)
100    orm.create()
101
102    printf('\n*** Insert names into table')
103    orm.insert()
104    orm.dbDump()
105
106    printf('\n*** Move users to a random group')
107    fr, to, num = orm.update()
108    printf('\t(%d users moved) from (%d) to (%d)' % (num, fr, to))
109    orm.dbDump()
110
111    printf('\n*** Randomly delete group')
112    rm, num = orm.delete()
113    printf('\t(group #%d; %d users removed)' % (rm, num))
114    orm.dbDump()
115
116    printf('\n*** Drop users table')
117    orm.drop()
118    printf('\n*** Close cxns')
119    orm.finish()
120
121 if __name__ == '__main__':
122     main()
```

Построчное объяснение

Строки 1–13

Как обычно, начнем описание с рассмотрения инструкций импорта модулей и определения констант. Согласно рекомендациям по стилю для сценариев на языке Python, вначале осуществляется импорт стандартных библиотечных модулей

(`distutils`, `os.path`, `random`), за этими инструкциями следуют инструкции импорта сторонних или внешних модулей (`sqlalchemy`) и, наконец, производится импорт модулей, локальных по отношению к данному приложению (`ushuffle_dbU`). В рассматриваемом приложении последняя операция импорта позволяет ввести в программу большинство необходимых констант и вспомогательных функций.

Еще в одной константе заданы имена источников базы данных (Database Source Name — DSN), которые можно рассматривать как указатель URI для соединения с базой данных. Приложение, которое рассматривалось в предыдущих изданиях настоящей книги, поддерживало только MySQL, а в этой книге удалось ввести в состав используемых средств базу данных SQLite. В приложении `ushuffle_dbU.py`, которое рассматривалось выше в данной главе, для работы с базой данных SQLite использовалась файловая система. В данном примере применяется версия, предусматривающая размещение данных в оперативной памяти (строка 12).



Шаблон Active Record

Active Record — это шаблон проектирования программного обеспечения (http://en.wikipedia.org/wiki/Active_record_pattern), который привязывает манипуляции объектами к эквивалентным действиям в базе данных. Объекты объектно-реляционного преобразователя по существу представляют строки базы данных, поэтому при создании объекта в базу данных автоматически записывается строка, представляющая данные этого объекта. При обновлении объекта то же происходит с соответствующей строкой. Аналогичным образом при удалении объекта удаляется соответствующая ему строка в базе данных.

Объектно-реляционный преобразователь SQLAlchemy сразу после его создания не был оснащен декларативным уровнем наподобие Active Record, который позволял бы упростить работу с ним. Вместо этого в нем применялся шаблон Data Mapper, в котором объекты не обладают способностью вносить изменения в саму базу данных. Скорее можно считать, что в этом шаблоне объекты поддерживали действия, которые мог вызывать пользователь для осуществления необходимых изменений. Практика показывает, что объектно-реляционные преобразователи действительно позволяют избавиться от необходимости применять для работы с базой данных исключительно код SQL, но разработчики все равно обязаны подготавливать эквивалентные операции с базой данных для осуществления добавлений, обновлений и удалений.

Стремление продолжить развитие интерфейсов, подобных Active Record, привело к реализации таких проектов, как ActiveRecord и TurboEntity. В конечном итоге вместо этих двух проектов был разработан Elixir (<http://elixir.ematia.de>), который занял место наиболее широко применяемого декларативного уровня для SQLAlchemy. Некоторые разработчики находят, что Elixir во многом напоминает известные средства Rails, тогда как другие отмечают, что этот проект чрезмерно упрощен и в нем применяется такой высокий уровень абстракции, на котором стали недоступными слишком многие функциональные средства.

В конечном итоге для объектно-реляционного преобразователя SQLAlchemy был создан собственный декларативный уровень, разработчики которого также придерживались шаблона Active Record. В нем применяется относительно небольшой объем кода, он является несложным и вполне позволяет справляться с работой, поэтому используется в следующем примере как более удобный для первого знакомства с подобным программным обеспечением. Тем не менее, если читатель придет к выводу, что этот декларативный уровень является слишком упрощенным, то он может по-прежнему использовать `__table__` object в качестве более традиционного средства доступа.

Строки 15–23

Применение декларативного уровня SQLAlchemy демонстрируется в следующем блоке кода. Работа с этим декларативным уровнем сводится к тому, что определяются объекты, манипулирование которыми приводит к выполнению эквивалентных операций в базе данных. Как следует из предыдущего примечания, подобные надстройки могут не обладать таким же широким набором средств, как инструменты сторонних разработчиков, но для рассматриваемого здесь простого примера слишком широкие возможности не требуются.

Для использования указанного декларативного уровня необходимо импортировать модуль `sqlalchemy.ext.declarative_base` (строка 7) и использовать его для создания класса `Base` (строка 15), который затем применяется для создания подклассов данных (строка 16).

Следующая часть определения класса содержит атрибут `__tablename__`, представляющий собой имя таблицы базы данных, на которую отображается этот класс. Еще один вариант состоит в том, что можно явно определить объект `sqlalchemy.Table` более низкого уровня, и в таком случае вместо класса должен применяться псевдоним для `__table__`. В этом приложении принят промежуточный подход, в котором для доступа к строкам главным образом используются объекты, а сама таблица (строка 41) применяется для выполнения действий на уровне таблицы (создания и удаления).

За этим следуют атрибуты `column` (для ознакомления со всеми допустимыми типами данных обратитесь к документации). Наконец, приведено определение метода `__str__()`, который возвращает предназначенное для восприятия человеком строковое представление данных. В этом приложении к выходным данным применяется форматирование (с помощью функции `tformat()`), поэтому мы не рекомендуем непосредственно использовать его на практике. Если бы этот код потребовалось повторно использовать в другом приложении, то пришлось бы преодолевать определенные сложности, связанные с необходимостью модифицировать применяемый способ форматирования вывода. Гораздо удобнее вместо этого создать подкласс этого класса и внести изменения в метод `__str__()` дочернего класса. Кроме того, SQLAlchemy поддерживает наследование таблиц.

Строки 25–42

Инициализатор класса, как и метод `ushuffle_dbU.connect()`, выполняет все от него зависящее для обеспечения доступа к базе данных, а затем сохраняет информацию о созданном соединении. В первую очередь предпринимается попытка использовать DSN для создания обработчика, предназначенного для работы с базой данных. Обработчик выполняет роль основного диспетчера базы данных. Иногда в целях отладки возникает необходимость просматривать код SQL, сформированный объектно-реляционным преобразователем. Для этого достаточно задать параметр `echo`, например, с помощью команды `create_engine('sqlite:///memory:', echo=True)`.

Неудачное завершение попытки создания обработчика (строки 29-30) означает, что SQLAlchemy не поддерживает выбранную базу данных. При этом обычно возникает исключение `ImportError`, связанное с тем, что не удастся найти необходимый адаптер. В этом случае происходит возврат функции `setup()` для формирования сообщения, передаваемого пользователю.

Если же создание обработчика выполняется успешно, то на следующем этапе предпринимается попытка подключения к базе данных. При этом неудачное

завершение обычно означает, что сама база данных (или ее сервер) доступна, но, как в данном случае, не существует база данных, предназначенная для хранения данных, поэтому предпринимается попытка ее создать и снова установить соединение (строки 34–37). Необходимо отметить, что на всякий случай, чтобы соединение было установлено успешно (строка 35), происходит удаление имени базы данных с помощью команды `os.path.dirname()`, а остальная часть DSN остается неизменной.

Это единственное место, где можно ознакомиться с реально применяемым кодом SQL (строка 36), поскольку применяемые при этом действия по существу относятся к категории администрирования базы данных, а не эксплуатации приложения. Все прочие операции с базой данных осуществляются в рамках таблицы путем манипуляции с объектами или вызова табличного метода базы данных с помощью делегирования (дополнительные сведения по этой теме приведены ниже, в описании строк 44–70).

В последнем разделе кода (строки 39–42) создается объект сеанса, предназначенный для управления отдельными транзакционными объектами, охватывающими одну или несколько операций с базами данных, которые должны быть все зафиксированы, чтобы можно было осуществить запись данных. После этого происходит сохранение объекта сеанса, пользовательской таблицы и обработчика как атрибутов экземпляра. Выполнение дополнительной привязки обработчика к метаданным таблицы (строка 42) означает, что к данному обработчику привязаны все операции с этой таблицей. (Привязка может быть также выполнена к другим обработчикам или соединениям.)

Строки 44–70

Далее следуют три метода, которые представляют основные функциональные средства базы данных, касающиеся вставки строк (строки 44–49), обновления строк (строки 51–60) и удаления строк (строки 62–70). Для вставки применяется метод `session.add_all()`, который принимает в качестве параметра итерационный объект и создает ряд операций вставки. По окончании этой работы можно принять решение о том, должна ли быть выполнена операция фиксации, как в данном примере (строка 49), или отката.

Оба метода, `update()` и `delete()`, поддерживают выполнение запросов в сеансе и позволяют использовать метод `query.filter_by()` для поиска. При обновлении происходит случайным образом выбор элементов из одной группы товаров (`fr`) и перемещение их в другой проект путем замены их идентификаторов другими значениями (`to`). Счетчик (`i`) отслеживает значение `rowcount`, которое показывает количество строк с внесенными изменениями в данные о пользователях. Как удаление рассматривается выбор случайным образом теоретического проекта компании по его идентификатору (`tm`), который был отменен, из-за чего произошло увольнение сотрудников компании. После выполнения обеих операций происходит фиксация их результатов с помощью объекта сеанса.

Следует учитывать, что существуют эквивалентные методы `update()` и `delete()` объекта запроса, которые не используются в рассматриваемом приложении. Применение этих методов способствовало бы сокращению объема необходимого кода, поскольку выполняемые с их помощью операции являются массовыми, а после каждого вызова метода на выполнение возвращается значение `rowcount`. Преобразование сценария `ushuffle_sad.py` с целью перехода к использованию этих методов предложено в качестве одного из упражнений в конце данной главы.

Ниже приведены некоторые из наиболее широко применяемых методов запроса.

- `filter_by()`. Извлечение значений с указанием конкретных значений столбца с помощью ключевых параметров.
- `filter()`. Этот метод является аналогичным `filter_by()`, но более гибким, поскольку в качестве параметра может быть задано выражение. Например, вызов `query.filter_by(userid=1)` равнозначен вызову `query.filter(Users.userid==1)`.
- `order_by()`. Аналогичен директиве ORDER BY языка SQL. По умолчанию сортировка производится по возрастанию. Для того чтобы обеспечить сортировку по убыванию, необходимо импортировать `sqlalchemy.desc()`.
- `limit()`. Аналогичен директиве LIMIT языка SQL.
- `offset()`. Аналогичен директиве OFFSET языка SQL.
- `all()`. Возврат всех объектов, согласованных в запросе.
- `one()`. Возврат только одного (следующего) объекта, который согласуется в запросе.
- `first()`. Возврат первого объекта, который согласуется в запросе.
- `join()`. Создание инструкции JOIN языка SQL по заданным желаемым условиям операции соединения.
- `update()`. Массовое обновление строк.
- `delete()`. Массовое удаление строк.

Применение большинства из этих методов приводит к созданию еще одного объекта Query, поэтому вызовы этих методов можно соединять в цепочку, например `query.order_by(desc(Users.userid)).limit(5).offset(5)`.

Если есть необходимость в использовании параметров LIMIT и OFFSET, то стилю языка Python в большей степени соответствует подход, при котором берется объект запроса и к нему применяется срез, например, как при использовании инструкции `query.order_by(Users.userid)[10:20]` для получения второй группы из десяти пользователей из набора данных о пользователях, отсортированного по идентификаторам.

Для ознакомления с методами Query обратитесь к документации по адресу <http://www.sqlalchemy.org/docs/orm/query.html#sqlalchemy.orm.query.Query>. Что касается операций соединения JOIN, то их описание составляет отдельную крупную тему, поэтому ознакомьтесь с дополнительной и более конкретной информацией по адресу <http://www.sqlalchemy.org/docs/orm/tutorial.html#ormtutorial-joins>. Читатель получит возможность провести эксперименты с некоторыми из этих методов, выполняя упражнения в конце данной главы.

Выше в этой главе рассматривалось в основном только применение запросов, поэтому речь шла об операциях уровня строки. Перейдем к изучению операций создания и удаления таблиц. В частности, определим, должны ли для этого применяться функции, подобные следующим:

```
def drop(self):
    self.users.drop()
```

В данном случае принято решение снова использовать делегирование (с вводным описанием этой темы можно ознакомиться в главе по объектно-ориентированному программированию книги *Core Python Language Fundamentals* или *Core Python Programming*). В текущем контексте под делегированием подразумевается получение атрибутов, недостающих в одном экземпляре, из другого объекта в экземпляре (`self.users`), в котором они имеются; признаком делегирования является применение инструкций `__getattr__()`, `self.users.create()`, `self.users.drop()` и т.д. (строки 79-80, 98-99, 116).

Строки 72–77

Для отображения выходных результатов на экране должным образом применяется метод `dbDump()`. Он извлекает строки из базы данных и применяет к полученным данным форматирование, по такому же принципу, как и эквивалентный ему метод в сценарии `ushuffle_dbU.py`. И действительно, оба этих метода почти идентичны.

Строки 79–83

Согласно приведенному выше краткому описанию делегирования, применение метода `__getattr__()` позволяет исключить необходимость в создании методов `drop()` и `create()`, поскольку эти методы, так или иначе, просто вызывали бы соответствующий метод `drop()` или `create()`, относящийся к таблице. Необходимость во введении каких-либо дополнительных функциональных средств отсутствует, поэтому нет смысла создавать еще пару функций, которые будут требовать поддержки. Следует еще раз отметить, что метод `__getattr__()` вызывается только в таких случаях, когда поиск атрибута оканчивается неудачей. (В отличие от него метод `__getattribute__()` вызывается в любом случае.)

Если после вызова `orm.drop()` не удастся найти требуемый метод, то происходит вызов `getattr(orm, 'drop')`. В таком случае вызывается `__getattr__()` и имя атрибута делегируется для применения в `self.users`. Интерпретатор находит, что в объекте `self.users` имеется атрибут `drop`, и передает этот вызов для обработки с помощью данного объекта: `self.users.drop()`.

Последним методом является метод `finish()`, который выполняет заключительные действия по очистке перед закрытием соединения. В данном случае этот метод мог быть оформлен как лямбда-функция, но от такого решения мы отказались, поскольку при очистке курсоров, закрытии соединений и т.д. нельзя обойтись одной инструкцией.

Строки 85–122

Ввод приложения в действие происходит с помощью функции `main()`. При этом создается объект `SQLAlchemyTest`, который используется во всех операциях с базой данных. Сам сценарий совпадает с тем, который применялся в исходном приложении `ushuffle_dbU.py`. Следует отметить, что параметр `db` базы данных является необязательным и не служит достижению какой-либо цели в данном примере `ushuffle_sad.py` или в следующей версии, предназначенной для работы с объектно-реляционным преобразователем `SQLObject`, `ushuffle_so.py`. Это “заглушка” (placeholder), которой можно воспользоваться, чтобы ввести поддержку других реляционных СУБД в этих приложениях (см. упражнения в конце главы).

После выполнения данного сценария может быть получен вывод, который на персональном компьютере с операционной системой Windows выглядит следующим образом:

```
C:\>python ushuffle_sad.py
*** Connect to 'test' database

Choose a database system:

(M) ySQL
(G) adfly
(S) QLite

Enter choice: s

*** Create users table (drop old one if appl.)
```

```
*** Insert names into table
```

LOGIN	USERID	PROJID
Faye	6812	2
Serena	7003	4
Amy	7209	2
Dave	7306	3
Larry	7311	2
Mona	7404	2
Ernie	7410	1
Jim	7512	2
Angela	7603	1
Stan	7607	2
Jennifer	7608	4
Pat	7711	2
Leslie	7808	3
Davina	7902	3
Elliot	7911	4
Jess	7912	2
Aaron	8312	3
Melissa	8602	1

```
*** Move users to a random group
      (3 users moved) from (1) to (3)
```

LOGIN	USERID	PROJID
Faye	6812	2
Serena	7003	4
Amy	7209	2
Dave	7306	3
Larry	7311	2
Mona	7404	2
Ernie	7410	3
Jim	7512	2
Angela	7603	3
Stan	7607	2
Jennifer	7608	4
Pat	7711	2
Leslie	7808	3

Davina	7902	3
Elliot	7911	4
Jess	7912	2
Aaron	8312	3
Melissa	8602	3

```
*** Randomly delete group
    (group #3; 7 users removed)
```

LOGIN	USERID	PROJID
Faye	6812	2
Serena	7003	4
Amy	7209	2
Larry	7311	2
Mona	7404	2
Jim	7512	2
Stan	7607	2
Jennifer	7608	4
Pat	7711	2
Elliot	7911	4
Jess	7912	2

```
*** Drop users table
```

```
*** Close cxns
C:\>
```

Способы доступа: явный, классический и с применением объектно-реляционного преобразователя

Как было сказано выше, для рассматриваемого примера было решено использовать декларативный уровень объектно-реляционного преобразователя SQLAlchemy. Однако для лучшего ознакомления с данной темой целесообразно также привести более “явную” форму сценария `ushuffle_sad.py` (его название расшифровывается так: User shuffle SQLAlchemy declarative — перестановка пользователей с помощью декларативного уровня ORM SQLAlchemy). Назовем эту версию `ushuffle_sae.py` (User shuffle SQLAlchemy explicit — перестановка пользователей с помощью явной формы ORM SQLAlchemy). Вполне очевидно, что эти две версии весьма напоминают друг друга.

Построчное объяснение здесь не приведено, поскольку сценарии `ushuffle_sad.py` и `ushuffle_sae.py` мало в чем отличаются друг от друга, но с этим описанием можно ознакомиться по адресу <http://corepython.com>. Мы решили не вносить существенных изменений по сравнению с предыдущими изданиями и вместе с тем предоставить возможность сравнить явную форму с декларативной. Кроме того, за время после выхода предыдущего издания настоящей книги программное обеспечение объектно-реляционного преобразователя SQLAlchemy достигло полной зрелости, поэтому мы стремились также отразить современное состояние этого средства. Сценарий `ushuffle_sae.py` приведен ниже.

```
#!/usr/bin/env python
```

```
from distutils.log import warn as printf
from os.path import dirname
from random import randrange as rand
```

```

from sqlalchemy import Column, Integer, String, create_engine,
    exc, orm, MetaData, Table
from sqlalchemy.ext.declarative import declarative_base
from ushuffle_dbU import DBNAME, NAMELEN, randName, FIELDS,
    tformat, cformat, setup

DSNs = {
    'mysql': 'mysql://root@localhost/%s' % DBNAME,
    'sqlite': 'sqlite:///memory:',
}

class SQLAlchemyTest(object):
    def __init__(self, dsn):
        try:
            eng = create_engine(dsn)

        except ImportError, e:
            raise RuntimeError()

        try:
            cxn = eng.connect()
        except exc.OperationalError:
            try:
                eng = create_engine(dirname(dsn))
                eng.execute('CREATE DATABASE %s' % DBNAME).close()
                eng = create_engine(dsn)
                cxn = eng.connect()
            except exc.OperationalError:
                raise RuntimeError()

        metadata = MetaData()
        self.eng = metadata.bind = eng
        try:
            users = Table('users', metadata, autoload=True)
        except exc.NoSuchTableError:
            users = Table('users', metadata,
                Column('login', String(NAMELEN)),
                Column('userid', Integer),
                Column('projid', Integer),
            )

        self.cxn = cxn
        self.users = users

    def insert(self):
        d = [dict(zip(FIELDS, [who, uid, rand(1,5)])) \
            for who, uid in randName()]
        return self.users.insert().execute(*d).rowcount

    def update(self):
        users = self.users
        fr = rand(1,5)
        to = rand(1,5)
        return (fr, to,
            users.update(users.c.projid==fr).execute(
                projid=to).rowcount)

    def delete(self):

```

```

    users = self.users
    rm = rand(1,5)
    return (rm,
            users.delete(users.c.projid==rm).execute().rowcount)

def dbDump(self):
    printf('\n%s' % ''.join(map(cformat, FIELDS)))
    users = self.users.select().execute()

    for user in users.fetchall():
        printf(''.join(map(tformat, (user.login,
                                     user.userid, user.projid))))

def __getattr__(self, attr):
    return getattr(self.users, attr)

def finish(self):
    self.cxn.close()

def main():
    printf('*** Connect to %r database' % DBNAME)
    db = setup()
    if db not in DSNs:
        printf('\nERROR: %r not supported, exit' % db)
        return

    try:
        orm = SQLAlchemyTest(DSNs[db])
    except RuntimeError:
        printf('\nERROR: %r not supported, exit' % db)
        return

    printf('\n*** Create users table (drop old one if appl.)')
    orm.drop(checkfirst=True)
    orm.create()

    printf('\n*** Insert names into table')
    orm.insert()
    orm.dbDump()

    printf('\n*** Move users to a random group')
    fr, to, num = orm.update()
    printf('\t(%d users moved) from (%d) to (%d)' % (num, fr, to))
    orm.dbDump()

    printf('\n*** Randomly delete group')
    rm, num = orm.delete()
    printf('\t(group #%d; %d users removed)' % (rm, num))
    orm.dbDump()

    printf('\n*** Drop users table')
    orm.drop()
    printf('\n*** Close cxns')
    orm.finish()

if __name__ == '__main__':
    main()

```

Наиболее заметные и важные различия между сценариями `ushuffle_sad.py` и `ushuffle_sae.py` состоят в следующем.

- Вместо декларативного объекта `Base` создается объект `Table`.
- Принято решение не использовать объекты сеанса `Session`; вместо этого применяются отдельные блоки инструкций, с автоматической фиксацией, без применения транзакций и т.д.
- Для всего взаимодействия с базой данных применяется объект `Table`, а не объекты `Session Query`.

Для того чтобы можно было убедиться в отсутствии связи между сеансами и явными операциями, в конце главы предложено выполнить упражнение по включению объектов `Session` в сценарий `ushuffle_sae.py`. Выше приведено достаточно полное описание `SQLAlchemy`, а теперь перейдем к рассмотрению объектно-реляционного преобразователя `SQLObject`, который представляет собой аналогичный инструмент.

SQLObject

`SQLObject` представляет собой первый объектно-реляционный преобразователь Python, который нашел широкое распространение. Фактически он был создан десять лет тому назад! Ян Бикинг (Ian Bicking), его разработчик, выпустил первую альфа-версию в октябре 2002 года (объектно-реляционный преобразователь `SQLAlchemy` не был известен до февраля 2006 года). Ко времени написания данной книги программное обеспечение `SQLObject` было предусмотрено только для версии Python 2.

Как уже было сказано выше, объектно-реляционный преобразователь `SQLObject` является в большей степени объектно-ориентированным (и более соответствующим стилю Python). В `SQLObject` на ранних этапах развития уже был реализован шаблон `Active Record` для обеспечения неявного доступа объектов к базе данных. С другой стороны, это программное обеспечение не предоставляет широких возможностей использования бесформатного кода SQL для выполнения более произвольных или специализированных запросов. Многие пользователи утверждают, что `SQLAlchemy` проще в изучении, но мы предлагаем читателю выработать собственное суждение на этот счет. Рассмотрим сценарий `ushuffle_so.py` в примере 6.3, который представляет собой результат переноса сценариев `ushuffle_dbU.py` и `ushuffle_sad.py` в среду `SQLObject`.

Пример 6.3. Применение объектно-реляционного преобразователя `SQLObject` (`ushuffle_so.py`)

Это совместимое с версиями Python 2.x и Python 3.x приложение по перестановке пользователей отличается тем, что в нем объектно-реляционный преобразователь `SQLObject` применяется наряду с базой данных `MySQL` или `SQLite`, которая служит в качестве серверной части.

```
1  #!/usr/bin/env python
2
3  from distutils.log import warn as printf
4  from os.path import dirname
5  from random import randrange as rand
6  from sqlobject import *
7  from ushuffle_dbU import DBNAME, NAMELEN, randName,
```



```

FIELDS, tformat, cformat, setup
8
9  DSNs = {
10     'mysql': 'mysql://root@localhost/%s' % DBNAME,
11     'sqlite': 'sqlite:///memory:',
12 }
13
14 class Users(SQLObject):
15     login = StringCol(length=NAMELEN)
16     userid = IntCol()
17     projid = IntCol()
18     def __str__(self):
19         return ''.join(map(tformat,
20                             (self.login, self.userid, self.projid)))
21
22 class SQLObjectTest(object):
23     def __init__(self, dsn):
24         try:
25             cxn = connectionForURI(dsn)
26         except ImportError:
27             raise RuntimeError()
28         try:
29             cxn.releaseConnection(cxn.getConnection())
30         except dberrors.OperationalError:
31             cxn = connectionForURI(dirname(dsn))
32             cxn.query("CREATE DATABASE %s" % dbName)
33             cxn = connectionForURI(dsn)
34         self.cxn = sqlhub.processConnection = cxn
35
36     def insert(self):
37         for who, userid in randName():
38             Users(login=who, userid=userid, projid=rand(1,5))
39
40     def update(self):
41         fr = rand(1,5)
42         to = rand(1,5)
43         i = -1
44         users = Users.selectBy(projid=fr)
45         for i, user in enumerate(users):
46             user.projid = to
47         return fr, to, i+1
48
49     def delete(self):
50         rm = rand(1,5)
51         users = Users.selectBy(projid=rm)
52         i = -1
53         for i, user in enumerate(users):
54             user.destroySelf()
55         return rm, i+1
56
57     def dbDump(self):
58         printf('\n%s' % ''.join(map(cformat, FIELDS)))
59         for user in Users.select():
60             printf(user)
61
62     def finish(self):
63         self.cxn.close()
64

```

```

65  def main():
66      printf('*** Connect to %r database' % DBNAME)
67      db = setup()
68      if db not in DSNs:
69          printf('\nERROR: %r not supported, exit' % db)
70          return
71
72      try:
73          orm = SQLAlchemyTest(DSNs[db])
74      except RuntimeError:
75          printf('\nERROR: %r not supported, exit' % db)
76          return
77
78      printf('\n*** Create users table (drop old one if appl.)')
79      Users.dropTable(True)
80      Users.createTable()
81
82      printf('\n*** Insert names into table')
83      orm.insert()
84      orm.dbDump()
85
86      printf('\n*** Move users to a random group')
87      fr, to, num = orm.update()
88      printf('\t(%d users moved) from (%d) to (%d)' % (num, fr, to))
89      orm.dbDump()
90
91      printf('\n*** Randomly delete group')
92      rm, num = orm.delete()
93      printf('\t(group #%d; %d users removed)' % (rm, num))
94      orm.dbDump()
95
96      printf('\n*** Drop users table')
97      Users.dropTable()
98      printf('\n*** Close cxns')
99      orm.finish()
100
101  if __name__ == '__main__':
102      main()

```

Построчное объяснение

Строки 1–12

Операции импорта и объявления констант для этого модуля практически идентичны их аналогам в сценарии `ushuffle_sad.py`, если не считать того, что вместо SQLAlchemy используется SQLAlchemy.

Строки 14–20

Таблица `Users` служит для расширения класса `SQLObject.SQLObject`. Мы определяем такие же столбцы, как и прежде, а также предоставляем метод `__str__()` для отображения вывода.

Строки 22–34

Конструктор для применяемого класса выполняет все от него зависящее для обеспечения доступа к базе данных и возвращает соединение с базой данных, как и в предыдущем примере, относящемся к SQLAlchemy. Аналогичным образом, только здесь предоставляется возможность ознакомиться с реально применяемым кодом SQL. Описание работы кода приведено ниже, включая то, какие действия осуществляются в связи с возникновением ошибок.

- Предпринимается попытка установить соединение с существующей таблицей (строка 29); если эта попытка завершается успешно, то данный этап считается выполненным. Такая организация работы предусмотрена для того, чтобы не возникали исключения, связанные с неудачным завершением проверок наличия доступного адаптера реляционного СУБД и работающего сервера, а на следующем этапе — проверки существования базы данных.
- В противном случае создается таблица; если эта попытка завершается успешно, то данный этап считается выполненным (строки 31–33).
- После успешного выполнения указанных действий объект соединения сохраняется в переменной `self.cnx`.

Строки 36–55

В этих строках приведены инструкции, применяемые для выполнения операций с базой данных. Предусмотрены операции вставки (строки 36–38), обновления (строки 40–47) и удаления (строки 49–55). Эти операции аналогичны своим эквивалентам для SQLAlchemy.



Уголок хакера. Функция `insert()`, которая сведена к одной (хотя и длинной) строке Python

Мы имеем возможность свести определение метода `insert()` к однострочной инструкции, хотя и более сложной для понимания:

```
[Users(**dict(zip(FIELDS, (who, userid, rand(1,5)))))) \
    for who, userid in randName()]
```

Не следует поощрять создание кода, который является более сложным для восприятия, а также кода, выполняемого явно с применением усовершенствованных операций со списком. Тем не менее, приходится считаться с тем, что существующее решение имеет один недостаток: оно требует создания новых объектов путем явного именования столбцов как ключевых параметров. Если же используется объект `FIELDS`, то отпадает необходимость определять имена столбцов, следовательно, вносить слишком много исправлений в код в случае изменения имен столбцов, особенно если объект `FIELDS` определен в некотором модуле конфигурации (а не приложения).

Строки 57–63

Этот блок начинается (как и следовало ожидать) с того же метода `dbDump()`, который осуществляет выборку строк из базы данных и выводит полученные данные в удобном формате на экран. Метод `finish()` (строки 62–63) применяется для закрытия соединения. В данном примере, в отличие от примера с SQLAlchemy, нельзя

было использовать делегирование для уничтожения таблицы, поскольку здесь метод, который мог быть делегирован, именуется `dropTable()`, а не `drop()`.

Строки 65–102

Это снова функция `main()`. Она действует точно так же, как в сценарии `ushuffle_sad.py`. Кроме того, строительными блоками, на основе которых может быть добавлена поддержка других реляционных СУБД в этих приложениях, становятся параметр `db` и константы DSN (см. упражнения в конце главы).

Ниже показан вывод, полученный при выполнении сценария `ushuffle_so.py` (который, как и следовало ожидать, почти идентичен выводу сценариев `ushuffle_dbU.py` и `ushuffle_sa?.py`).

```
$ python ushuffle_so.py
*** Connect to 'test' database

Choose a database system:

(M) MySQL
(G) adfly
(S) QLite

Enter choice: s

*** Create users table (drop old one if appl.)

*** Insert names into table
```

LOGIN	USERID	PROJID
Jess	7912	2
Ernie	7410	1
Melissa	8602	1
Serena	7003	1
Angela	7603	1
Aaron	8312	4
Elliot	7911	3
Jennifer	7608	1
Leslie	7808	4
Mona	7404	4
Larry	7311	1
Davina	7902	3
Stan	7607	4
Jim	7512	2
Pat	7711	1
Amy	7209	2
Faye	6812	1
Dave	7306	4

```

*** Move users to a random group
      (5 users moved) from (4) to (2)
```

LOGIN	USERID	PROJID
Jess	7912	2
Ernie	7410	1
Melissa	8602	1
Serena	7003	1
Angela	7603	1

Aaron	8312	2
Elliot	7911	3
Jennifer	7608	1
Leslie	7808	2
Mona	7404	2
Larry	7311	1
Davina	7902	3
Stan	7607	2
Jim	7512	2
Pat	7711	1
Amy	7209	2
Faye	6812	1
Dave	7306	2

```
*** Randomly delete group
      (group #3; 2 users removed)
```

LOGIN	USERID	PROJID
Jess	7912	2
Ernie	7410	1
Melissa	8602	1
Serena	7003	1
Angela	7603	1
Aaron	8312	2
Jennifer	7608	1
Leslie	7808	2
Mona	7404	2
Larry	7311	1
Stan	7607	2
Jim	7512	2
Pat	7711	1
Amy	7209	2
Faye	6812	1
Dave	7306	2

```
*** Drop users table
```

```
*** Close cxns
$
```

6.4. Нереляционные базы данных

В начале данной главы приведено краткое описание языка SQL и дан общий обзор реляционных баз данных. Затем было показано, как обеспечить обмен данными с системами подобных типов, а также представлены общие сведения о переносе программ в версию Python 3. За этими разделами последовали разделы, посвященные описанию объектно-реляционных преобразователей, а также связанных с ними возможностей для пользователей обойтись без применения языка SQL за счет перехода к более объектно-ориентированному подходу. Однако следует помнить, что незаметно для пользователя и SQLAlchemy, и SQLAlchemy формируют от его имени код SQL. Теперь перейдем к заключительному разделу настоящей главы, в котором по-прежнему рассматриваются объекты, но речь о реляционных базах данных уже не идет.

6.4.1. Введение в NoSQL

Новейшие тенденции развития Интернета и социальных сетей привели к тому, что формирование данных, подлежащих сохранению, стало происходить с такой интенсивностью и в таких объемах, с которыми не могут справиться реляционные базы данных. Достаточно представить себе, в каких масштабах происходит выработка новых данных в процессе работы сети Facebook или Twitter. Например, разработчики игр для Facebook или приложений, обрабатывающих потоковые данные Twitter, могут сталкиваться с такими прикладными программами, для которых требуется обеспечивать сохранение на постоянной основе потока данных со скоростью, равной миллионам строк или объектов в час. Стремление решить указанную проблему масштабируемости привело к созданию, взрывообразному росту и развертыванию нереляционных баз данных, или так называемых баз данных NoSQL.

В связи с развитием таких баз данных появился широкий спектр программных продуктов, но они предоставляют разные возможности. Даже к самой отдельно взятой категории нереляционных (non-relational, или non-rel) программных продуктов относятся объектные базы данных; хранилища “ключ–значение”; хранилища документов (или хранилища данных); графические базы данных; табличные базы данных; базы данных с организацией по столбцам, с расширяемыми записями, с широкими столбцами; многозначные базы данных и т.д. В конце настоящей главы будут приведены некоторые ссылки, с которых можно начать изучение тематики NoSQL. Ко времени написания данной книги одной из наиболее широко применяемых нереляционных баз данных для хранения документов была MongoDB.

6.4.2. База данных MongoDB

В последнее время база данных MongoDB находит все более и более широкую область распространения. Эта база данных имеет большое количество пользователей, для нее подготовлен значительный объем документации, организовано сообщество и предусмотрена профессиональная поддержка. Еще одним признаком ее всеобщего признания является регулярное проведение посвященных ей конференций. На главном веб-сайте разработчиков этой базы данных можно найти сведения о том, что работу с ней проводят многие важные компании, включая Craigslist, Shutterfly, foursquare, bit.ly, SourceForge и др. Сведения об этих и многих других компаниях можно найти по адресу <http://www.mongodb.org/display/DOCS/Production+Deployments>. База данных MongoDB хорошо подходит для использования при вступительном ознакомлении с базами данных NoSQL и хранилищами документов не только потому, что у нее широкий круг пользователей, но и по многим другим причинам. Как любопытный факт отметим, что система хранения документов базы данных MongoDB написана на языке C++.

Общее сравнение хранилищ документов (MongoDB, CouchDB, Riak, Amazon SimpleDB) с другими нереляционными базами данных показывает, что хранилища документов занимают промежуточное место между простыми хранилищами “ключ–значение”, такими как Redis, Voldemort, Amazon Dynamo и т.д., с одной стороны, и хранилищами данных, организованных по столбцам, наподобие Cassandra, Google Bigtable и Hbase, с другой. Хранилища документов немного напоминают приложения, не поддерживающие схемы, производные по отношению к реляционным базам данных, более простые и менее ограниченные по сравнению с системами хранения по столбцам, но более гибкие, чем обычные хранилища “ключ–значение”. Такие

хранилища, как правило, сохраняют данные в виде объектов JSON (JavaScript Object Notation), что позволяет определять типы данных, такие как строки, числа, списки, а также формировать иерархические структуры.

Необходимо учитывать, что для описания MongoDB (и NoSQL) применяется немного иная терминология по сравнению с реляционными системами баз данных. Например, в них вместо строк и столбцов рассматриваются документы и коллекции. Для того чтобы было проще разобраться в том, как изменилась терминология, можно ознакомиться со схемой соответствия SQL и Mongo по адресу <http://www.mongodb.org/display/DOCS/SQL+to+Mongo+Mapping+Chart>.

В частности, база данных MongoDB может применяться для хранения полезных данных JSON (документов), которые могут рассматриваться как отдельный словарь Python, после сериализации с преобразованием в двоичный код, обычно именуемый как формат BSON. Если не касаться механизма хранения MongoDB, то разработчики могут рассматривать эту базу данных как хранилище JSON, которое, в свою очередь, напоминает словарь Python, и из этого можно исходить при создании на ее основе программного обеспечения. База данных MongoDB нашла достаточно широкое распространение, поэтому для нее предусмотрены адаптеры, обеспечивающие работу со многими платформами, включая Python.

6.4.3. Адаптер PyMongo: MongoDB и Python

Количество адаптеров MongoDB, предусмотренных для сопряжения со средой Python, достаточно велико, но наиболее тщательно разработанным из них является PyMongo. Другие адаптеры либо относятся к категории более упрощенных, либо имеют специальное назначение. Чтобы получить список всех пакетов Python, предназначенных для работы с MongoDB, можно выполнить поиск по ключевому слову `mongo` на сайте Cheeseshop (<http://pypi.python.org>). По желанию читатель может попытаться воспользоваться любым из них, но в примере, приведенном в данной главе, используется PyMongo.

Кроме всего прочего, важным преимуществом пакета `pymongo` является то, что он был перенесен в версию Python 3. Согласно подходу, который уже использовался ранее в этой главе, мы представим только одно приложение Python, работающее и в версии Python 2, и в версии Python 3, с учетом того, какой интерпретатор применяется для выполнения сценария, а это, в свою очередь, требует применения версии `pymongo`, инсталлированной должным образом.

Для описания процесса установки будет отведено лишь несколько строк, поскольку данная книга имеет другое назначение; тем не менее, достаточно указать, что можно перейти по адресу mongodb.org для загрузки MongoDB и воспользоваться средством `easy_install` или `pip`, чтобы установить PyMongo и (или) PyMongo3. (Примечание. Автор не испытал каких-либо затруднений при получении `pymongo3` на своем компьютере Mac, а что касается Windows, то в процессе установки возникли ошибки.) Независимо от того, будет ли установлен тот или другой пакет (или оба пакета), вызов этого программного обеспечения в коде должен выглядеть одинаково: `import pymongo`.

Для того чтобы можно было убедиться в том, что база данных MongoDB установлена и работает правильно, ознакомьтесь с кратким руководством по адресу <http://www.mongodb.org/display/DOCS/Quickstart>, а для выполнения аналогичной проверки по отношению к PyMongo проверьте, успешно ли выполняется импорт пакета `pymongo`. Для ознакомления с тем, как используется программное обеспечение

MongoDB в сценарии на языке Python, выполните упражнения из учебника PyMongo по адресу <http://api.mongodb.org/python/current/tutorial.html>.

Теперь приступим к переносу созданного ранее приложения для перестановки пользователей (`ushuffle_*.py`), которое рассматривалось на протяжении всей данной главы, чтобы можно было использовать MongoDB в качестве системы постоянного хранения данных. Заслуживает внимания то, что текущее приложение по своей организации во многом напоминает приложения, разработанные для SQLAlchemy и SQLAlchemy, но важна даже не эта особенность, а то, что работа с MongoDB требует гораздо меньше издержек по сравнению с типичной реляционной системой баз данных, такой как MySQL. В примере 6.4 представлен сценарий `ushuffle_mongo.py`, совместимый с версиями Python 2 и Python 3, за которым следует построчное описание.

Пример 6.4. Применение базы данных MongoDB (`ushuffle_mongo.py`)

Это — приложение для перестановки пользователей, совместимое с Python 2.x и Python 3.x, в котором применяются MongoDB и PyMongo.

```

1  #!/usr/bin/env python
2
3  from distutils.log import warn as printf
4  from random import randrange as rand
5  from pymongo import Connection, errors
6  from ushuffle_dbU import DBNAME, randName, FIELDS,
   tformat, cformat
7
8  COLLECTION = 'users'
9
10 class MongoTest(object):
11     def __init__(self):
12         try:
13             cxn = Connection()
14         except errors.AutoReconnect:
15             raise RuntimeError()
16         self.db = cxn[DBNAME]
17         self.users = self.db[COLLECTION]
18
19     def insert(self):
20         self.users.insert(
21             dict(login=who, userid=uid, projid=rand(1,5)) \
22             for who, uid in randName())
23
24     def update(self):
25         fr = rand(1,5)
26         to = rand(1,5)
27         i = -1
28         for i, user in enumerate(self.users.find({'projid': fr})):
29             self.users.update(user,
30                               {'$set': {'projid': to}})
31         return fr, to, i+1
32
33     def delete(self):
34         rm = rand(1,5)
35         i = -1
36         for i, user in enumerate(self.users.find({'projid': rm})):
37             self.users.remove(user)

```



```
38         return rm, i+1
39
40     def dbDump(self):
41         printf('\n%s' % ''.join(map(cformat, FIELDS)))
42         for user in self.users.find():
43             printf(''.join(map(tformat,
44                               (user[k] for k in FIELDS))))
45
46     def finish(self):
47         self.db.connection.disconnect()
48
49 def main():
50     printf('*** Connect to %r database' % DBNAME)
51     try:
52         mongo = MongoTest()
53     except RuntimeError:
54         printf('\nERROR: MongoDB server unreachable, exit')
55         return
56
57     printf('\n*** Insert names into table')
58     mongo.insert()
59     mongo.dbDump()
60
61     printf('\n*** Move users to a random group')
62     fr, to, num = mongo.update()
63     printf('\t(%d users moved) from (%d) to (%d)' % (num, fr, to))
64     mongo.dbDump()
65
66     printf('\n*** Randomly delete group')
67     rm, num = mongo.delete()
68     printf('\t(group #%d; %d users removed)' % (rm, num))
69     mongo.dbDump()
70
71     printf('\n*** Drop users table')
72     mongo.db.drop_collection(COLLECTION)
73     printf('\n*** Close cxns')
74     mongo.finish()
75
76 if __name__ == '__main__':
77     main()
```

Построчное объяснение

Строки 1–8

Основная строка импорта предназначена для ввода в действие объекта Connection пакета PyMongo и исключений пакета (errors). Все прочие описания были приведены выше в данной главе. По аналогии с примерами для объектно-реляционных преобразователей, здесь снова заимствуются большинство констант и общих функций из созданного ранее приложения `ushuffle_dbU.py`. Последняя инструкция задает имя для применяемой коллекции (table).

Строки 10–17

В первой части инициализатора для рассматриваемого класса `MongoTest` создается соединение и активизируется исключение, если не удастся установить связь с сервером (строки 12–15). Следующие две строки вполне могут ускользнуть от внимания читателя, поскольку они внешне выглядят как простые присваивания, но по существу в этих строках создается база данных или повторно используется существующая (строка 16), а также создается или повторно используется существующая коллекция `users`, которая может рассматриваться как своего рода аналог таблицы базы данных.

Структура таблиц такова, что в них определяются столбцы, затем создаются строки для каждой записи, тогда как к схемам коллекций не предъявляется никаких требований; в коллекции лишь выполняется привязка отдельных документов к каждой записи. Следует отметить, что в этой части кода в глаза бросается отсутствие определения класса модели данных. Каждая запись определяет сама себя, иначе говоря, в коллекцию попадает любая сохраняемая запись, какой бы она ни была.

Строки 19–22

Метод `insert()` служит для добавления значений к коллекции `MongoDB`. Коллекция состоит из документов. Документ можно рассматривать как отдельную запись в форме словаря Python. Словари создаются с использованием относящейся к документам фабричной функции `dict()` для каждой записи, а затем перенаправляются в метод `insert()` коллекции через выражение генератора.

Строки 24–31

Метод `update()` действует по такому же принципу, как было описано ранее в этой главе. Различие состоит в методе `update()` коллекции, который предоставляет разработчикам большие возможности по сравнению с типичной системой баз данных. В данном случае (строки 29–30) используется директива `$set` базы данных `MongoDB`, которая явно обновляет существующее значение.

Каждая директива `MongoDB` представляет операцию модификатора, которая является одновременно высокоэффективной, полезной и удобной для разработчика при обновлении существующих значений. Кроме `$set`, предусмотрены также операции увеличения значения поля на определенную величину, удаления поля (пары “ключ–значение”), добавления и удаления значений в массиве и т.д.

Рассмотрим, что происходит перед этим. Прежде чем выполнить обновление, необходимо передать запрос на получение данных обо всех пользователях в системе (строка 28), чтобы найти пользователей с идентификатором проекта (`projid`), соответствующем группе, в которой должно быть произведено обновление. Для этого используется метод `find()` коллекции, которому передаются условия поиска. Тем самым происходит замена инструкции `SELECT` языка `SQL`.

Предусмотрена также возможность использовать метод `Collection.update()` для изменения нескольких документов; для этого достаточно лишь задать значение True флага `multi`. Единственным недостатком рассматриваемого подхода является то, что он в настоящее время не обеспечивает возврат общего количества измененных документов.

Для ознакомления с более сложными запросами по сравнению с тем, что используется в нашем простом сценарии и содержит только одно условие, см. соответствующий раздел в официальной документации по адресу <http://www.mongodb.org/display/DOCS/Advanced+Queries>.

Строки 33–38

В методе `delete()` повторно используется тот же запрос, что и в методе `update()`. После получения данных обо всех пользователях, которые соответствуют запросу, происходит удаление их по одному с помощью метода `remove()` (строки 36-37) и возврат результатов. Если данные об общем количестве удаленных документов не представляют интереса, то можно применить просто отдельный вызов метода `self.users.remove()`, который удаляет из коллекции все документы, соответствующие условиям.

Строки 40–44

В запросе, выполняемом в методе `dbDump()`, условия не заданы (строка 42), поэтому происходит возврат всех записей пользователей в коллекции, за ними следуют данные, которые форматируются в виде строки и отображаются для пользователя приложения (строки 43-44).

Строки 46-47

Последний метод, определяемый и вызываемый в ходе выполнения приложения, разрывает соединение с сервером MongoDB.

Строки 49–77

Ведущая функция `main()` не требует пояснений и действует точно по такому же принципу, как и в предыдущих приложениях, рассматриваемых в настоящей главе: подключение к серверу базы данных и выполнение подготовительной работы; вставка пользователей в коллекцию (`table`) и формирование дампа содержимого базы данных; перемещение пользователей из одного проекта в другой (и дамп содержимого); удаление всей группы (и дамп содержимого); удаление всей коллекции; отключение.

На этом описание средств поддержки нереляционных баз данных в языке Python завершается, но для читателя это описание должно стать лишь отправной точкой. Как было указано в начале этого раздела, многие функциональные возможности NoSQL заслуживают внимания, поэтому необходимо изучить каждую из них и даже, возможно, подготовить для них прототипы, чтобы определить наиболее подходящий инструмент для выполнения конкретного задания. В следующем разделе приведены некоторые ссылки, с помощью которых можно найти дополнительные сведения по данной теме.

6.4.4. Резюме

Хотелось бы надеяться, что настоящая глава послужит хорошим введением в тематику использования реляционных баз данных в языке Python. После того как обнаруживается, что в приложении для работы с данными нельзя обойтись простыми и даже специализированными файлами, такими как файлы DBM, файлы в формате `pickle` и т.д., перед разработчиком встает сложная задача осуществления выбора среди многих вариантов средств хранения данных. К ним относится весьма значительное количество реляционных СУБД, в том числе реализованных полностью на языке Python, при использовании которых не возникает необходимость устанавливать, поддерживать или управлять реальной системой баз данных.

В следующем разделе приведена информация о многих адаптерах Python, а также о системах баз данных и объектно-реляционных преобразователях. Кроме того, в

последнее время сообществом пользователей средств хранения данных были разработаны нереляционные базы данных, применимые в тех ситуациях, когда не удается масштабировать реляционные базы данных до такого уровня, который требуется для конкретного приложения.

При изучении этой темы рекомендуется также просматривать страницы группы по интересам DB-SIG, кроме того, веб-страницы и списки рассылки для всех систем, заслуживающих внимания. Как и во всех других областях разработки программного обеспечения, при работе на языке Python упрощается и изучение тематики, и проведение экспериментов.

6.5. Справочная информация

В табл. 6.8 перечислено большинство предоставляемых для общего доступа баз данных, наряду с рабочими модулями и пакетами Python, которые могут служить в качестве адаптеров к этим системам баз данных. Необходимо учитывать, что не все адаптеры являются совместимыми с DB-API.

Таблица 6.8. Модули/пакеты и веб-сайты, относящиеся к тематике баз данных

Имя	Источник информации в Интернете
Реляционные базы данных	
Gadfly	gadfly.sf.net
MySQL	mysql.com или mysql.org
MySQLdb (другое название MySQL-python)	sf.net/projects/mysql-python
MySQL Connector/Python	launchpad.net/myconnpy
PostgreSQL	postgresql.org
psycopg	initd.org/psycopg
PyPgSQL	pypgsql.sf.net
PyGreSQL	pygresql.org
SQLite	sqlite.org
pysqlite	trac.edgewall.org/wiki/PySqlite
sqlite3 ^a	docs.python.org/library/sqlite3
2.5	
APSW	code.google.com/p/apsw
MaxDB (SAP)	maxdb.sap.com
sdb.dbapi	maxdb.sap.com/doc/7_7/46/702811f2042d87e1000000a1553f6/content.htm
sdb.sql	maxdb.sap.com/doc/7_7/46/71b2a816ae0284e1000000a1553f6/content.htm
sapdb	sapdb.org/sapdbPython.html
Firebird (InterBase)	firebirdsql.org
KInterbasDB	firebirdsql.org/en/python-driver
SQL Server	microsoft.com/sql
pymssql	code.google.com/p/pymssql (требуется FreeTDS [freetds.org])
adodbapi	adodbapi.sf.net

Имя	Источник информации в Интернете
Sybase	sybase.com
sybase	www.object-craft.com.au/projects/sybase
Oracle	oracle.com
cx_Oracle	cx-oracle.sf.net
DCOracle2	zope.org/Members/matt/dco2 (более старая версия, только для Oracle8)
Ingres	ingres.com
Ingres DBI	community.actian.com/wiki/Ingres_Python_Development_Center
ingmod	www.informatik.uni-rostock.de/~hme/software/
Хранилища документов NoSQL	
MongoDB	mongodb.org
PyMongo	pypi.python.org/pypi/pymongo. Документация по адресу api.mongodb.org/python/current
PyMongo3	pypi.python.org/pypi/pymongo3
Другие адаптеры	api.mongodb.org/python/current/tools.html
CouchDB	couchdb.apache.org
couchdb-python	code.google.com/p/couchdb-python. Документация по адресу packages.python.org/CouchDB
Объектно-реляционные преобразователи	
SQLObject	sqlobject.org
SQLObject2	sqlobject.org/2
SQLAlchemy	sqlalchemy.org
Storm	storm.canonical.com
PyDO/PyDO2	skunkweb.sf.net/pydo.html

^a Адаптер `pysqlite` впервые введен в версии Python 2.5 в качестве модуля `sqlite3`.

В дополнение к ссылкам, относящимся к базам данных, модулям и пакетам, ниже приведен еще один набор оперативных ссылок, которые заслуживают внимания.

Python и базы данных

- wiki.python.org/moin/DatabaseProgramming
- wiki.python.org/moin/DatabaseInterfaces

Форматы базы данных, структуры и шаблоны разработки

- en.wikipedia.org/wiki/DSN
- www.martinfowler.com/eaCatalog/dataMapper.html
- en.wikipedia.org/wiki/Active_record_pattern
- blog.mongodb.org/post/114440717/bson

Нереляционные базы данных

- en.wikipedia.org/wiki/Nosql
- nosql-database.org/
- www.mongodb.org/display/DOCS/MongoDB,+CouchDB,+MySQL+Compare+Grid

6.6. Упражнения

Базы данных

- 6.1. *API базы данных.* Что такое DB-API языка Python? В чем преимущества этого стандарта? Почему его следует (или не следует) использовать?
- 6.2. *API базы данных.* Опишите различия между стилями параметров модулей баз данных (см. атрибут модуля `paramstyle`).
- 6.3. *Объекты курсора.* В чем состоят различия между методами `execute*()` курсора?
- 6.4. *Объекты курсора.* В чем состоят различия между методами `fetch*()` курсора?
- 6.5. *Адаптеры баз данных.* Внимательно изучите применяемые вами реляционную СУБД и относящийся к ней модуль Python. Совместимы ли они с DB-API? Какие дополнительные возможности предоставляет этот модуль, которые являются вспомогательными и не требуются согласно DB-API?
- 6.6. *Объекты типа.* Изучите применяемые вами базу данных и адаптер DB-API с точки зрения предусмотренных в них объектов `Type`, а затем напишите небольшой сценарий, в котором используется по крайней мере один из этих объектов.
- 6.7. *Применение рефакторинга.* В функции `ushuffle_dbU.create()` существующая таблица уничтожается, а затем повторно создается с помощью рекурсивного вызова `create()`. Такая организация работы является небезопасной, поскольку после (еще одного) неудачного завершения попытки повторного создания таблицы возникает бесконечная рекурсия. Исправьте этот недостаток, подготовив более практичное решение, не требующее вновь копировать запрос создания (`cur.execute()`) в обработчике исключений. **Дополнительное задание.** Предусмотрите повторение попыток воссоздания таблицы не больше трех раз, после чего должен происходить возврат в вызывающий код с сообщением об ошибке.
- 6.8. *База данных и язык HTML.* Возьмите любую существующую таблицу базы данных и воспользуйтесь своими знаниями о программировании для веб, чтобы создать обработчик, который выводит содержимое этой таблицы в виде кода HTML для браузеров.
- 6.9. *Программирование для веб и базы данных.* Воспользуйтесь примером сценария перестановки пользователей (`ushuffle_db.py`) и создайте для него веб-интерфейс.

- 6.10. *Программирование графического интерфейса пользователя и базы данных.* Воспользуйтесь примером сценария перестановки пользователей (`ushuffle_db.py`) и создайте для него графический интерфейс пользователя.
- 6.11. *Класс портфеля ценных бумаг.* Создайте приложение, которое управляет портфелем ценных бумаг для нескольких пользователей. Воспользуйтесь в качестве серверной части реляционной базой данных и предусмотрите пользовательский интерфейс на основе веб. Вы можете воспользоваться классом базы данных для ценных бумаг из главы по объектно-ориентированному программированию книги *Core Python Language Fundamentals* или *Core Python Programming*.
- 6.12. *Отладка и рефакторинг.* В функциях `update()` и `remove()` имеется небольшой недостаток: метод `update()` может перемещать пользователей в составе одной группы в ту же группу. Внесите такое изменение, чтобы случайно выбранная целевая группа отличалась от группы, из которой перемещается пользователь. Аналогичным образом в методе `remove()` может предприниматься попытка удалить пользователя из группы, не имеющей членов (поскольку они либо не существуют, либо были перемещены с помощью метода `update()`).

Объектно-реляционные преобразователи

- 6.13. *Класс портфеля ценных бумаг.* Создайте альтернативное решение по отношению к сценарию портфеля ценных бумаг (упражнение 6.11) с использованием объектно-реляционного преобразователя, исключив возможность непосредственной записи данных в реляционную СУБД.
- 6.14. *Отладка и рефакторинг.* Перенесите свое решение упражнения 6.13 в примеры для SQLAlchemy и SQLAlchemy.
- 6.15. *Поддержка различных реляционных СУБД.* Воспользуйтесь приложением SQLAlchemy (`ushuffle_sad.py`) или SQLAlchemy (`ushuffle_so.py`), которое в настоящее время поддерживает MySQL или SQLite, и добавьте еще одну реляционную базу данных по своему выбору.

Для следующих четырех упражнений возьмите за основу сценарий `ushuffle_dbU.py`, отличительной особенностью которого является то, что в нем определен код, находящийся ближе к началу сценария (строки 7–12), служит для определения того, какая функция должна использоваться для получения ввода данных пользователем из командной строки.

- 6.16. *Импорт и язык Python.* Еще раз рассмотрим код, о котором шла речь. Для чего требуется проверка того, представляет ли собой `__builtins__` словарь `dict`, а не модуль?
- 6.17. *Перенос в версию Python 3.* Применение метода `distutils.log.warn()` не может служить идеальной заменой для `print/print()`. Докажите это. Предоставьте фрагменты кода, которые показывают, в каких случаях `warn()` не является совместимым с `print()`.
- 6.18. *Перенос в версию Python 3.* Некоторые пользователи считают, что метод `print()` можно использовать в версии Python 2, как и в версии Python 3. Докажите их неправоту. **Подсказка.** Упражнение от самого Гвидо: `print(x, y)`.

- 6.19. Язык Python.** Предположим, что решено использовать `print()` в Python 3, а `distutils.log.warn()` в Python 2, но желательно воспользоваться именем `printf()`. В чем ошибка в приведенном ниже коде?

```
from distutils.log import warn
if hasattr(__builtins__, 'print'):
    printf = print
else:
    printf = warn
```

- 6.20. Исключения.** Если в сценарии `ushuffle_sad.py` соединение с сервером устанавливалось с использованием назначенного имени базы данных, то исключение (`exc.OperationalError`) указывало, что таблица не существует, поэтому приходилось возвращаться назад и сначала создавать базу данных, а затем еще раз пытаться выполнить подключение к базе данных. Но это не единственный источник ошибок: если используется база данных MySQL и остановлен сам сервер, активизируется такое же исключение. В этом случае попытка выполнения инструкции `CREATE DATABASE` также оканчивается неудачей. Добавьте еще один обработчик, позволяющий справиться с этой ситуацией, который возвращал бы в код, где осуществляется попытка создать экземпляр, исключение `RuntimeError`.

- 6.21. SQLAlchemy.** Дополните функцию `ushuffle_sad.dbDump()` путем добавления нового заданного по умолчанию параметра `newest5`, который принимает значение по умолчанию `False`. Если передается значение `True`, то вместо отображения всех пользователей должны происходить сортировка списка в обратном направлении по значению `Users.userid` и отображение пяти первых строк, представляющих сотрудников, которые были приняты на работу последними. Поместите этот специальный вызов в функции `main()` непосредственно после вызова `orm.insert()` и `orm.dbDump()`.

а) Используйте методы `Query.limit()` и `offset()`.

б) Используйте вместо этого синтаксис создания срезов языка Python.

После таких обновлений вывод должен выглядеть примерно так:

```
...
Jess      7912      4
Aaron     8312      3
Melissa   8602      2
```

*** Top 5 newest employees

LOGIN	USERID	PROJID
Melissa	8602	2
Aaron	8312	3
Jess	7912	4
Elliot	7911	3
Davina	7902	3

```
*** Move users to a random group
    (4 users moved) from (3) to (1)
```

LOGIN	USERID	PROJID
Faye	6812	4
Serena	7003	2
Amy	7209	1

- 6.22. *SQLAlchemy*. Внесите изменения в сценарий `ushuffle_sad.update()` для использования метода `Query update()` после перехода вниз на 5 строк кода. Примените модуль `timeit` для определения того, является ли этот вариант более быстрым по сравнению с исходным.
- 6.23. *SQLAlchemy*. То же, что и в упражнении 6.22, но для `ushuffle_sad.delete()` используйте метод `Query delete()`.
- 6.24. *SQLAlchemy*. В этой явно недекларативной версии `ushuffle_sad.py`, `ushuffle_sae.py` исключено использование декларативного уровня, а также сеансов. Безусловно, применение модели `Active Record` не является столь обязательным, но в целом концепция сеансов `Session` не так уж плоха. Внесите в сценарии `ushuffle_sae.py` изменения в весь код, применяемый для выполнения операций в базе данных, чтобы в нем (совместно) использовался объект `Session`, как в декларативной версии `ushuffle_sad.py`.
- 6.25. *Модели данных Django*. По аналогии с тем, что реализовано в примерах для *SQLAlchemy* или *SQLObject*, возьмите за основу класс модели данных `Users` и создайте эквивалентный сценарий с использованием объектно-реляционного преобразователя `Django`. Для этого вам может потребоваться заранее ознакомиться с главой 11, “Платформы для веб. Django”.
- 6.26. *Объектно-реляционный преобразователь Storm*. Перенесите приложение `ushuffle_s*.py` в среду объектно-реляционного преобразователя `Storm`.

Нереляционные базы данных (NoSQL)

- 6.27. *NoSQL*. Назовите некоторые причины, по которым нереляционные базы данных находят все более широкое распространение. Какие возможности, выходящие за рамки традиционных реляционных баз данных, они предлагают?
- 6.28. *NoSQL*. Разработано по меньшей мере четыре различных типа нереляционных баз данных. Дайте определение каждого из этих основных типов и назовите наиболее широко известные проекты в каждой категории. Особо отметьте те из них, для которых имеется по крайней мере один адаптер `Python`.
- 6.29. *CouchDB*. `CouchDB` — это еще одно хранилище документов, которое часто сравнивают с `MongoDB`. Ознакомьтесь с некоторыми приведенными в Интернете сравнениями, ссылки на которые можно найти в заключительном разделе настоящей главы, а затем загрузите и установите `CouchDB`. Преобразуйте сценарий `ushuffle_mongo.py` в совместимый с `CouchDB` сценарий `ushuffle_couch.py`.

ГЛАВА

7

Программирование приложений для работы с Microsoft Office

В этой главе...

- Введение
- Программирование клиентов COM на языке Python
- Вступительные примеры
- Промежуточные примеры
- Соответствующие модули/пакеты

Независимо от того, что нужно сделать, всегда имеется ограничительный фактор, который определяет, насколько быстро и качественно удастся это сделать. Вы должны изучить свою задачу и определить лежащий в ее основе ограничительный фактор, или ограничение. Затем необходимо сосредоточить всю свою энергию на прохождении этого узкого места.

Брайен Трейси (Brian Tracy), март 2001 г.
(из книги Eat That Frog, 2001, Berrett-Koehler)

Эта глава не похожа на большую часть основного содержания данной книги, поскольку мы не сосредоточиваемся на разработке сетевых приложений, графических интерфейсов пользователя, веб-приложений или приложений с командной строкой и используем язык Python для решения совсем другой задачи: управления собственным программным обеспечением, а именно приложениями Microsoft Office с помощью средств программирования клиентов для службы COM (Component Object Model).

7.1. Введение

Независимо от того, нравится ли это разработчикам или нет, им часто приходится организовывать взаимодействие своих приложений с программным обеспечением на основе операционной системы Windows. При этом, каков бы ни был объем решаемых задач, значительную помощь может оказать язык Python, будь то разовая работа или повседневно осуществляемая деятельность.

В настоящей главе рассматриваются средства программирования клиентов COM на языке Python, которые могут использоваться для управления и взаимодействия с приложениями Microsoft Office, такими как Word, Excel, PowerPoint и Outlook. COM — это служба, с помощью которой приложения, функционирующие на персональном компьютере, могут взаимодействовать друг с другом. В данной главе рассматриваются клиентские программы COM, в задачу которых может входить взаимодействие с такими известными приложениями, как набор программ Office.

Обычно принято, что для написания клиентов COM применяются два очень мощных, но весьма непохожих друг на друга инструмента: Microsoft Visual Basic (VB) или Visual Basic for Applications (VBA), либо (Visual) C++. Кроме того, в качестве удобной замены для этих инструментов при программировании COM часто рассматривается язык Python, поскольку он является более мощным, чем VB, а также более выразительным и менее трудоемким по сравнению с языком C++, применяемым в качестве средства разработки.

Еще более новыми инструментами являются IronPython, .NET и VSTO. С их помощью также можно разрабатывать приложения, взаимодействующие с набором программ Office, но при этом в основе организации функционирования программ по-прежнему лежит интерфейс COM. Это означает, что и в данном случае остается применимым материал, приведенный в данной главе, даже если используются некоторые из более усовершенствованных инструментов.

Глава предназначена, с одной стороны, для разработчиков COM, которые хотят понять, как применяется Python в том мире, который для них хорошо знаком, а с другой, — для программистов на языке Python, желающим узнать, как создают клиенты COM для автоматизации задач наподобие формирования электронных

таблиц Excel, создания образцов писем в виде документов Word, подготовки слайдов для презентации с помощью PowerPoint, отправки электронной почты через Outlook и т.д. В этой главе не обсуждаются принципы или понятия COM, кроме того, отсутствует обоснование необходимости применения COM. К тому же в главе не приведено описание COM+, ATL, IDL, MFC, DCOM, ADO, .NET, IronPython, VSTO и других инструментов.

Вместо этого приобщение к программированию клиентов COM будет происходить в форме описания того, как использовать язык Python для обеспечения взаимодействия с приложениями Office.

7.2. Программирование клиентов COM на языке Python

Одно из наиболее полезных дел, которые может совершить в своей повседневной рабочей среде программист, работающий на языке Python, состоит в том, чтобы интегрировать в эту среду средства поддержки для приложений Windows. Часто оказывается, что очень удобно считывать данные с помощью приложений Windows, а также осуществлять запись данных. Пусть даже среда Windows не применяется в самом подразделении разработчиков, но весьма велики шансы, что приложениями этой среды пользуются управленческие структуры и другие группы проектировщиков. Для взаимодействия с приложениями Windows в собственной среде этих приложений программисты могут воспользоваться расширениями Windows Extensions for Python Марка Хэммонда (Mark Hammond).

Программные средства Windows охватывают весьма обширную сферу приложений, причем доступ для большей части этой сферы можно получить с помощью пакета расширений Windows Extensions for Python. Этот пакет расширений предоставляет такой набор функций, как API (Applications Programming Interface) Windows, средства порождения процессов, поддержка базовых классов Microsoft (Microsoft Foundation Classes — MFC), разработка графического интерфейса пользователя, многопоточное программирование Windows, службы, удаленный доступ, каналы, серверное программирование COM и события. Однако далее в этой главе будет рассматриваться только одна часть этой сферы приложений Windows: программирование клиентов COM.

7.2.1. Программирование клиентов COM

Стандарт COM (для которого предусмотрено также маркетинговое название ActiveX) может применяться для взаимодействия с такими инструментами, как Outlook и Excel. Как показывает практика, программисты испытывают особое удовлетворение благодаря тому, что у них есть возможность управлять собственными приложениями Office непосредственно из своего кода на языке Python.

Следует особо отметить, что при обсуждении использования объектов COM, например, для запуска приложения и предоставления стороннему коду доступа к методам и данным этих приложений, применяется термин *программирование клиента COM*. *Программирование сервера COM* — это реализация объектов COM, к которым клиенты могут получить доступ.



Язык Python и программирование клиентов Microsoft COM

Реализация Python на 32-разрядной платформе Windows предоставляет возможность подключения к объектам COM, в основе которых лежит технология поддержки интерфейсов Microsoft, обеспечивающая взаимодействие объектов независимо от языка или формата данных. В этом разделе будет показано, что сочетание Python и COM (вернее, клиентских средств этого интерфейса) предоставляет уникальную возможность создания сценариев, способных непосредственно взаимодействовать с такими приложениями Microsoft Office, как Word, Excel, PowerPoint и Outlook.

7.2.2. Вводные сведения

Для практического освоения этого раздела необходимо использовать персональный компьютер (или виртуальную машину персонального компьютера в любой системе), на котором функционирует 32- или 64-разрядная версия Windows. Кроме того, необходимо установить на компьютере (по меньшей мере) версию .NET 2.0, Python и расширения Extensions Python for Windows. (Эти расширения представлены по адресу <http://pywin32.sf.net>.) Наконец, необходимо иметь установленные приложения Microsoft (одно или несколько), на которых можно было бы опробовать представленные примеры. Разработка может осуществляться с помощью командной строки или с применением интегрированной среды разработки PythonWin, которая входит в состав дистрибутива Extensions.

Автор должен признаться, что он не является экспертом в части интерфейса COM или разработчиком программного обеспечения Microsoft, но вместе с тем обладает достаточной квалификацией для успешной демонстрации использования Python для управления приложениями Office. Вполне естественно, что приведенные здесь примеры могут быть существенно улучшены. Обращаемся к вам с просьбой писать и отправлять в наш адрес любые комментарии, предложения или улучшенные варианты сценариев, которые вы посчитали бы достойными для представления широкому кругу пользователей.

Последняя часть главы состоит из демонстрационных сценариев, с которых можно начать освоение программирования для каждого из основных приложений Office; в завершение главы приведено несколько промежуточных примеров. Прежде чем приступать к описанию примеров, необходимо подчеркнуть, что выполнение всех клиентских приложений COM осуществляется в виде примерно одинаковых шагов. Типичный способ организации взаимодействия с приложениями Office состоит в следующем.

1. Запуск приложения.
2. Создание документа требуемого типа, над которым будет осуществляться работа (или загрузка существующего документа).
3. Выполнение операции, после которой приложение становится видимым (если это необходимо).
4. Осуществление всех необходимых операций над документом.
5. Сохранение или уничтожение документа.
6. Завершение работы.

После этого краткого вступления приступим к рассмотрению некоторых сценариев. В следующем разделе приведен ряд сценариев, применяемых для управления различными приложениями Microsoft. Во всех этих сценариях осуществляется импорт модуля `win32com.client`, а также нескольких модулей Tk, применяемых для управления запуском (и завершением) каждого приложения. Кроме того, как и в главе 5, применяется расширение файла `.pyw` для подавления ненужного командного окна DOS.

7.3. Вступительные примеры

В этом разделе рассматриваются несложные примеры, которые могут послужить отправной точкой для приобщения к разработкам для четырех основных приложений Office: Excel, Word, PowerPoint и Outlook.

7.3.1. Программа Excel

Первый пример представляет собой демонстрацию использования программы Excel. Среди всех приложений набора Office в наибольшей степени удобным для программирования является программа Excel. С помощью Excel можно обеспечить выполнение чрезвычайно эффективных операций наполнения документов данными и обработки данных, что позволяет воспользоваться преимуществами электронных таблиц, а также организовать данные в наглядном формате, пригодном для печати. Весьма перспективной является также возможность считывать данные из электронных таблиц и обрабатывать эти данные, используя всю мощь такого полнофункционального языка программирования, как Python. Более сложный пример будет представлен в конце данного раздела, но для начала рассмотрим пример 7.1.

Пример 7.1. Пример применения Excel (`excel.pyw`)

В данном сценарии происходит запуск Excel, после чего в ячейки электронной таблицы записываются данные.

```
1  #!/usr/bin/env python
2
3  from Tkinter import Tk
4  from time import sleep
5  from tkMessageBox import showwarning
6  import win32com.client as win32
7
8  warn = lambda app: showwarning(app, 'Exit?')
9  RANGE = range(3, 8)
10
11 def excel():
12     app = 'Excel'
13     xl = win32.gencache.EnsureDispatch('%s.Application' % app)
14     ss = xl.Workbooks.Add()
15     sh = ss.ActiveSheet
16     xl.Visible = True
17     sleep(1)
18
19     sh.Cells(1,1).Value = 'Python-to-%s Demo' % app
20     sleep(1)
```

```

21 for i in RANGE:
22     sh.Cells(i,1).Value = 'Line %d' % i
23     sleep(1)
24 sh.Cells(i+2,1).Value = "Th-th-th-that's all folks!"
25
26 warn(app)
27 ss.Close(False)
28 xl.Application.Quit()
29
30 if __name__=='__main__':
31     Tk().withdraw()
32     excel()

```

Построчное объяснение

Строки 1–6, 31

Импорт модулей Tkinter и tkMessageBox осуществляется исключительно для того, чтобы можно было открыть окно сообщения showwarning после завершения демонстрации. Окно Tk верхнего уровня закрывается с помощью withdraw() для его подавления (строка 31) перед вызовом диалогового окна (строка 26). Если инициализация окна верхнего уровня не будет выполнена заранее, то данное окно будет создано автоматически; закрытие этого окна не произойдет, и на экране останется ненужный компонент изображения.

Строки 11–17

После запуска кода Excel (*диспетчеризации*) происходит добавление книги (электронной таблицы, содержащей листы, в которых происходит запись данных; листы организованы как вкладки в рабочей книге); затем в сценарии происходит захват дескриптора активного листа (таковым является лист, отображаемый на экране). Для тех, кто недостаточно знаком с этой терминологией, отметим, что может возникнуть путаница, если не учитывать, что электронные таблицы (spreadsheet) прежде всего состоят из листов (sheet).



Статическая и динамическая диспетчеризация

В строке 13 выполняется операция, которую принято называть *статической диспетчеризацией*. Для этого перед запуском сценария необходимо выполнить программу Makerу из интерфейса приложения PythonWin. (Запустите эту интегрированную среду разработки, выберите Tools, COM Makepy utility, а затем укажите соответствующую прикладную объектную библиотеку.) Эта служебная программа создает и кеширует объекты, необходимые для приложения. Если такая подготовительная работа не будет выполнена, то возникнет необходимость в построении объектов и атрибутов во время выполнения; такая операция именуется *динамической диспетчеризацией*. Для того чтобы ограничиться использованием динамической диспетчеризации, можно применять регулярную функцию Dispatch():

```
xl = win32com.client.Dispatch('%s.Application' % app)
```

Флаг `Visible` необходимо присвоить значение `True`, чтобы приложение Office стало видимым на рабочем столе; приостановка выполняется для того, чтобы можно было видеть каждый шаг демонстрации (строка 16).

Строки 19–24

В части сценария, касающейся приложения, происходит запись названия демонстрации в первой (левой верхней) ячейке, которая имеет номер (A1), или (1, 1). После этого пропускается одна строка и записываются строки “Line N”, где N — число от 3 до 7, причем происходит приостановка на одну секунду перед каждой следующей строкой, чтобы можно было наглядно видеть происходящие обновления. (Если бы не было этой задержки, то обновления ячеек осуществлялись бы слишком быстро для слежения за ними. Именно по этой причине во всем сценарии применяются вызовы `sleep()`.)

Строки 26–32

После демонстрации открывается диалоговое окно с предупреждением, которое указывает, что после ознакомления с выводом можно выйти из программы. Электронная таблица закрывается без сохранения, поскольку задан атрибут `ss.Close([SaveChanges=False])`, и происходит выход из приложения. Наконец, в части “main” сценария инициализируется среда Tk и выполняется основная часть приложения.

Выполнение этого сценария приводит к появлению окна приложения Excel, которое должно выглядеть примерно так, как показано на рис. 7.1.

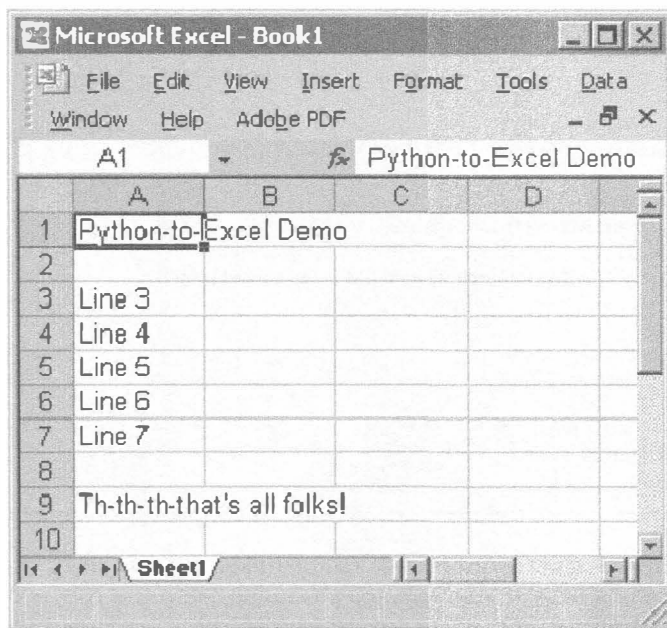


Рис. 7.1. Демонстрационный сценарий управления приложением Excel из сценария Python (`excel.pyw`)

7.3.2. Программа Word

Для следующей демонстрации требуется приложение Word. В мире программирования не принято непосредственно использовать текстовый процессор Word как средство работы с документами по аналогии с базами данных, поскольку это приложение не позволяет поддерживать большой объем данных. Тем не менее Word вполне может применяться для подготовки образцов писем. В примере 7.2 документ создается по принципу записи одной строки текста за другой.

Пример 7.2. Пример применения Word (word.pyw)

В этом сценарии происходит запуск приложения Word и запись данных в документ.

```

1  #!/usr/bin/env python
2
3  from Tkinter import Tk
4  from time import sleep
5  from tkMessageBox import showwarning
6  import win32com.client as win32
7
8  warn = lambda app: showwarning(app, 'Exit?')
9  RANGE = range(3, 8)
10
11 def word():
12     app = 'Word'
13     word = win32.gencache.EnsureDispatch('%s.Application' % app)
14     doc = word.Documents.Add()
15     word.Visible = True
16     sleep(1)
17
18     rng = doc.Range(0,0)
19     rng.InsertAfter('Python-to-%s Test\r\n\r\n' % app)
20     sleep(1)
21     for i in RANGE:
22         rng.InsertAfter('Line %d\r\n' % i)
23         sleep(1)
24     rng.InsertAfter("\r\nTh-th-th-that's all folks!\r\n")
25
26     warn(app)
27     doc.Close(False)
28     word.Application.Quit()
29
30 if __name__ == '__main__':
31     Tk().withdraw()
32     word()

```

Пример для программы Word весьма напоминает аналогичный сценарий, приведенный в примере для Excel. Единственное различие состоит в том, что вместо записи в ячейки происходит вставка строк в “текстовую область” документа и перемещение указателя мыши на следующую строку после каждой операции записи. Необходимо также отдельно выполнять вставку символов завершения строки, т.е. символов возврата каретки и перевода строки, (\r\n).

После выполнения этого сценария экран с результатами может выглядеть примерно так, как показано на рис. 7.2.

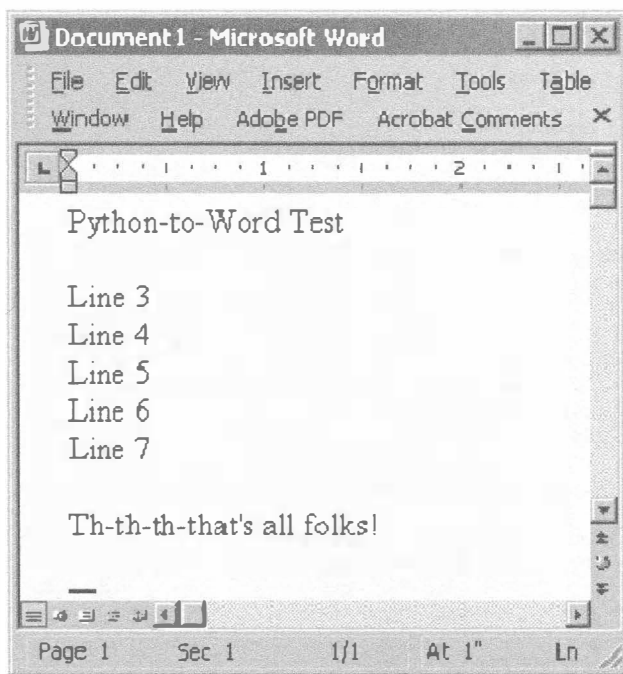


Рис. 7.2. Демонстрационный сценарий управления приложением Word из сценария Python (word.pyw)

7.3.3. Программа PowerPoint

На первый взгляд не совсем очевидно, для чего может потребоваться применение интерфейса к приложению PowerPoint, но не следует забывать, что иногда возникает необходимость срочно подготовить презентацию, допустим, для конференции. В таком случае можно, например, подготовить текстовый файл с маркированными списками во время переезда к месту проведения события, а затем воспользоваться несколькими свободными часами для написания сценария, который интерпретирует файл и автоматически формирует ряд слайдов. После этого можно дополнительно украсить свои слайды, добавляя фон, анимацию и т.д., поскольку все это позволяет интерфейс COM. Ситуация может также сложиться таким образом, что потребуется автоматически сформировать или изменить новые либо существующие презентации. В таком случае можно создать сценарий COM, выполняемый с помощью сценария командного интерпретатора, с помощью которого можно было бы сформировать и откорректировать каждую презентацию. Итак, предстоящий для изучения материал может оказаться не лишним, поэтому перейдем к рассмотрению примера 7.3, в котором показан сценарий работы с приложением PowerPoint.

Пример 7.3. Пример применения PowerPoint (ppoint.pyw)

В этом сценарии происходит запуск PowerPoint, после чего формируется представление данных в виде фигур на слайде.

```
1  #!/usr/bin/env python
2
3  from Tkinter import Tk
4  from time import sleep
5  from tkMessageBox import showwarning
6  import win32com.client as win32
7
8  warn = lambda app: showwarning(app, 'Exit?')
9  RANGE = range(3, 8)
10
11 def ppoint():
12     app = 'PowerPoint'
13     ppoint = win32.gencache.EnsureDispatch('%s.Application' % app)
14     pres = ppoint.Presentations.Add()
15     ppoint.Visible = True
16
17     sl = pres.Slides.Add(1, win32.constants.ppLayoutText)
18     sleep(1)
19     sla = sl.Shapes[0].TextFrame.TextRange
20     sla.Text = 'Python-to-%s Demo' % app
21     sleep(1)
22     slb = sl.Shapes[1].TextFrame.TextRange
23     for i in RANGE:
24         slb.InsertAfter("Line %d\r\n" % i)
25         sleep(1)
26     slb.InsertAfter("\r\nTh-th-th-that's all folks!")
27
28     warn(app)
29     pres.Close()
30     ppoint.Quit()
31
32 if __name__ == '__main__':
33     Tk().withdraw()
34     ppoint()
```

В этом случае снова нельзя не заметить аналогий с обеими предыдущими демонстрациями, для Excel и Word. Отличительной особенностью является то, что в приложении PowerPoint запись данных осуществляется в другие объекты. Работать с приложением PowerPoint немного сложнее, поскольку в нем вместо отдельного активного листа или документа применяются презентации, состоящие из нескольких слайдов, а каждый слайд может иметь отличную от других компоновку. (В недавно выпущенных версиях PowerPoint предусмотрено до 30 различных компоновок!) Действия, которые могут быть выполнены на слайде, зависят от выбранной компоновки.

В рассматриваемом примере используется компоновка, которая включает только название и текст (строка 17), и происходит заполнение основного названия (строки 19-20), формирование фигур Shape[0] или Shape(1) (следует учитывать, что последовательности Python начинаются с индекса 0, а в программном обеспечении Microsoft

в качестве начального индекса применяется 1), подготовка текстовой части (строки 22–26) и формирование фигур Shape[1] или Shape(2). Для того чтобы выяснить, какие константы должны использоваться, необходимо ознакомиться со списком всех доступных констант. Например, константа `ppLayoutText` определена как имеющая значение 2 (целочисленное), константа `ppLayoutTitle` равна 1 и т.д. Определения констант можно найти во многих книгах по программированию в среде Microsoft VB/Office или получить в Интернете, выполнив поиск по именам констант. С помощью модуля `win32.constants` можно также использовать просто целочисленные значения констант, не обращая к их именам.

Снимок экрана с приложением PowerPoint показан на рис. 7.3.

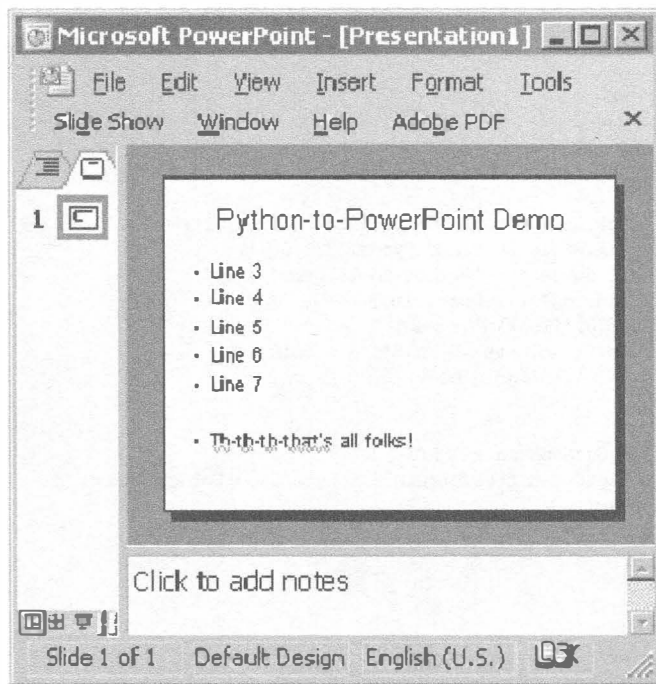


Рис. 7.3. Демонстрационный сценарий управления приложением PowerPoint из сценария Python (`ppoint.py`)

7.3.4. Программа Outlook

Наконец, рассмотрим демонстрационный сценарий для программы Outlook, в котором используется еще больше констант, чем в сценарии для программы PowerPoint. Приложение Outlook представляет собой столь же распространенный и универсальный инструмент, как и Excel, поэтому вполне оправдано стремление организовать с ним работу с помощью языка Python. В программе Python всегда можно найти возможность для работы с адресами электронной почты, сообщениями и другими данными, которыми достаточно легко управлять. В примере 7.4 рассматривается сценарий работы с приложением Outlook, в котором осуществляется немного больше функций по сравнению с предыдущими примерами.

Пример 7.4. Пример применения Outlook (olook.pyw)

В этом сценарии происходит запуск приложения Outlook, создается новое сообщение, осуществляется его передача, после чего пользователю предоставляется возможность открыть и рассмотреть не только само сообщение, но и интерфейс Outbox.

```

1  #!/usr/bin/env python
2
3  from Tkinter import Tk
4  from time import sleep
5  from tkMessageBox import showwarning
6  import win32com.client as win32
7
8  warn = lambda app: showwarning(app, 'Exit?')
9  RANGE = range(3, 8)
10
11 def outlook():
12     app = 'Outlook'
13     olook = win32.gencache.EnsureDispatch('%s.Application' % app)
14
15     mail = olook.CreateItem(win32.constants.olMailItem)
16     recip = mail.Recipients.Add('you@127.0.0.1')
17     subj = mail.Subject = 'Python-to-%s Demo' % app
18     body = ["Line %d" % i for i in RANGE]
19     body.insert(0, '%s\r\n' % subj)
20     body.append("\r\nTh-th-th-that's all folks!")
21     mail.Body = '\r\n'.join(body)
22     mail.Send()
23
24     ns = olook.GetNamespace("MAPI")
25     obox = ns.GetDefaultFolder(win32.constants.olFolderOutbox)
26     obox.Display()
27     obox.Items.Item(1).Display()
28
29     warn(app)
30     olook.Quit()
31
32 if __name__ == '__main__':
33     Tk().withdraw()
34     outlook()

```

В данном примере Outlook используется для отправки электронной почты самому себе. Для успешного выполнения этого демонстрационного примера необходимо отключить доступ к сети, чтобы не выполнялась действительная отправка сообщения. В таком случае сообщение остается в папке Outbox, и его можно просмотреть (а после просмотра при желании удалить). После запуска Outlook создается новое почтовое сообщение, в котором заполнены такие обязательные поля, как получатель, тема и текст (строки 15–21). Затем вызывается метод send() (строка 22) для помещения письма в очередь сообщений в почтовом ящике Outbox, откуда это электронное письмо должно быть перемещено в папку Sent Mail (Отправленные) после его передачи почтовому серверу.

Как и в приложении PowerPoint, в программе Outlook применяется много констант; для сообщений электронной почты используется константа olMailItem (со значением 0). К другим часто применяемым константам Outlook относятся

olAppointmentItem (1), olContactItem (2) и olTaskItem (3). Разумеется, на этом список применяемых констант далеко не исчерпывается, поэтому при необходимости в дополнительных константах читатель может просмотреть книгу по программированию для VB/Office или выполнить поиск констант и их значений в Интернете.

В следующем разделе (строки 24–27) используется еще одна константа, olFolderOutbox (4), для открытия папки Outbox и подготовки ее для отображения. Необходимо также найти последний по времени объект в этой папке (рассчитывая на то, что таковым является только что созданное нами письмо) и сформировать его отображение. Другие широко применяемые константы, относящиеся к папкам, включают: olFolderInbox (6), olFolderCalendar (9), olFolderContacts (10), olFolderDrafts (16), olFolderSentMail (5) и olFolderTasks (13). При использовании динамической диспетчеризации может оказаться так, что потребуется указывать числовые значения констант вместо имен (см. предыдущее примечание).

На рис. 7.4 приведен снимок экрана, где показано только окно сообщения.

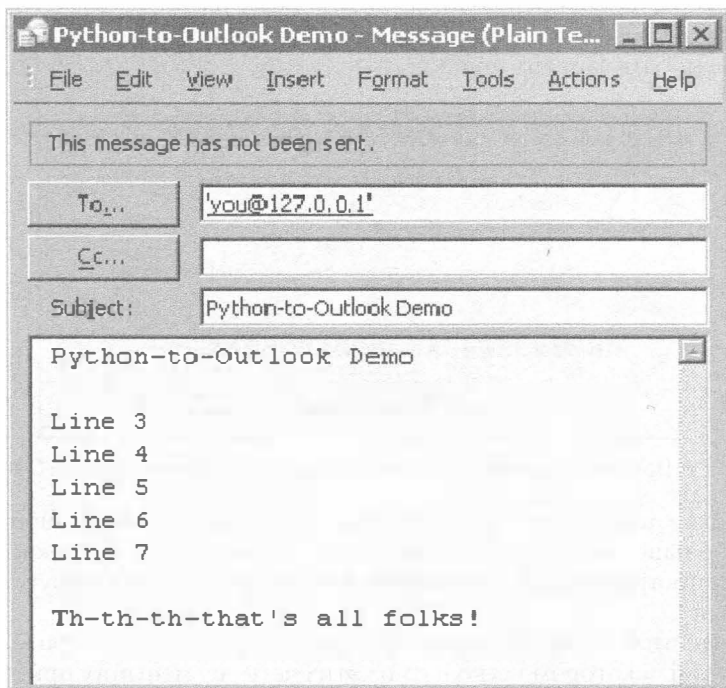


Рис. 7.4. Пример выполнения демонстрационного сценария управления приложением Outlook из сценария Python (olook.pyw)

Прежде чем продолжить рассмотрение данной темы, необходимо отметить, что на практике приложение Outlook оказалось весьма уязвимым к атакам многих типов, поэтому корпорацией Microsoft в это приложение были встроены определенные средства защиты, которые ограничивают доступ к адресной книге и регламентируют возможность отправлять электронную почту от имени пользователя. При попытке доступа к данным программы Outlook открывается окно, показанное на рис. 7.5, с помощью которого можно явно предоставить разрешение для доступа внешней программе.

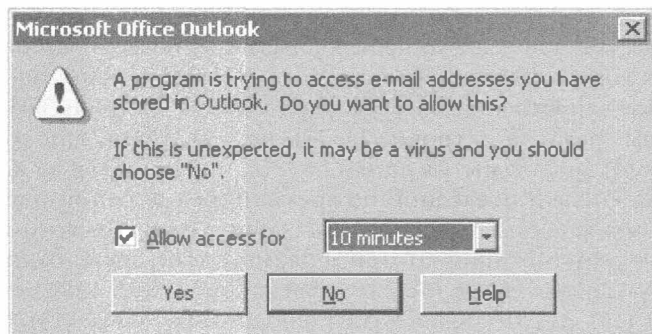


Рис. 7.5. Предупреждение о попытке доступа к адресной книге Outlook

Затем, при попытке отправить сообщение из внешней программы, открывается диалоговое окно с предупреждением, показанным на рис. 7.6. Необходимо подождать до завершения отсчета таймера, поскольку лишь после этого предоставляется возможность выбрать вариант “Да”.

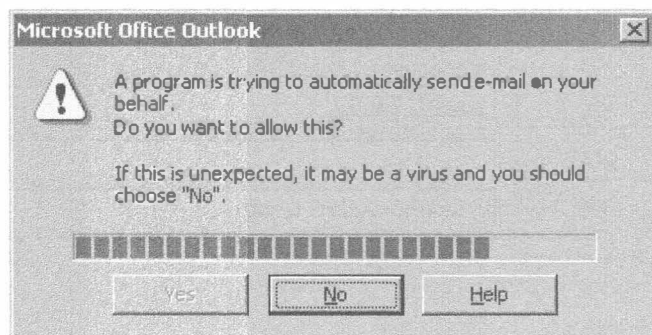


Рис. 7.6. Предупреждение о попытке передачи электронной почты Outlook

После прохождения всех этих проверок безопасности какие-либо затруднения встречаться больше не должны. Обойти эти проверки можно также с помощью специального программного обеспечения, но его необходимо загружать и устанавливать отдельно.

На веб-сайте этой книги по адресу <http://corepython.com> приведен альтернативный сценарий, в котором несколько из этих четырех меньших приложений объединены в одно приложение, которое позволяет пользователю выбрать демонстрационный пример для ознакомления.

7.4. Промежуточные примеры

Примеры, которые рассматривались выше в настоящей главе, были предназначены для обеспечения возможности приступить к использованию языка Python для управления продуктами Microsoft Office. Теперь рассмотрим несколько практически применимых приложений, в том числе те, которые автор регулярно использует в своей работе.

7.4.1. Програма Excel

В этом примере в дополнение к материалу данной главы используется часть содержимого главы 13. В настоящей главе многое взято из сценария `stock.py`, приведенного в примере 13.1, в котором используется служба Yahoo! Finance для запроса данных о котировках акций. Пример 7.5 показывает, как можно объединить сценарий из примера получения котировок акций с нашим демонстрационным сценарием Excel; в конечном итоге должно быть получено приложение, которое загружает котировки акций из Интернета и вставляет их непосредственно в документ Excel, в связи с чем отпадает необходимость создавать или использовать в качестве промежуточных файлы в формате CSV.

Пример 7.5. Пример получения котировок акций и применения приложения Excel (`estock.pyw`)

В этом сценарии загружаются котировки акций из Yahoo! и происходит запись данных в документ Excel.

```

1  #!/usr/bin/env python
2
3  from Tkinter import Tk
4  from time import sleep, ctime
5  from tkMessageBox import showwarning
6  from urllib import urlopen
7  import win32com.client as win32
8
9  warn = lambda app: showwarning(app, 'Exit?')
10 RANGE = range(3, 8)
11 TICKS = ('YHOO', 'GOOG', 'EBAY', 'AMZN')
12 COLS = ('TICKER', 'PRICE', 'CHG', '%AGE')
13 URL = 'http://quote.yahoo.com/d/quotes.csv?s=%s&f=s1lclp2'
14
15 def excel():
16     app = 'Excel'
17     xl = win32.gencache.EnsureDispatch('%s.Application' % app)
18     ss = xl.Workbooks.Add()
19     sh = ss.ActiveSheet
20     xl.Visible = True
21     sleep(1)
22
23     sh.Cells(1, 1).Value = 'Python-to-%s Stock Quote Demo' % app
24     sleep(1)
25     sh.Cells(3, 1).Value = 'Prices quoted as of: %s' % ctime()
26     sleep(1)
27     for i in range(4):
28         sh.Cells(5, i+1).Value = COLS[i]
29     sleep(1)
30     sh.Range(sh.Cells(5, 1), sh.Cells(5, 4)).Font.Bold = True
31     sleep(1)
32     row = 6
33
34     u = urlopen(URL % ', '.join(TICKS))
35     for data in u:
36         tick, price, chg, per = data.split(',')
37         sh.Cells(row, 1).Value = eval(tick)

```



```

38     sh.Cells(row, 2).Value = ('%.2f' % round(float(price), 2))
39     sh.Cells(row, 3).Value = chg
40     sh.Cells(row, 4).Value = eval(per.rstrip())
41     row += 1
42     sleep(1)
43     u.close()
44
45     warn(app)
46     ss.Close(False)
47     xl.Application.Quit()
48
49 if __name__ == '__main__':
50     Tk().withdraw()
51     excel()

```

Построчное объяснение

Строки 1–13

Рассматривается несложный сценарий, взятый из главы 13, в котором поддерживается процесс получения котировок акций с узла службы Yahoo! Finance. В данной главе мы берем основной компонент из указанного сценария и встраиваем его в пример, обеспечивающий получение необходимых данных и передачу их в электронную таблицу Excel.

Строки 15–32

В первой части основного приложения происходит запуск программы Excel (строки 17–21), как и в предыдущих примерах. Затем в ячейки записываются название и отметка времени (строки 23–29), а также заголовки столбцов, к тексту которых затем применяется стиль с полужирным шрифтом (строка 30). Остальные ячейки отводятся для записи фактических данных котировок акций, начиная со строки таблицы 6 (строка 32).

Строки 34–43

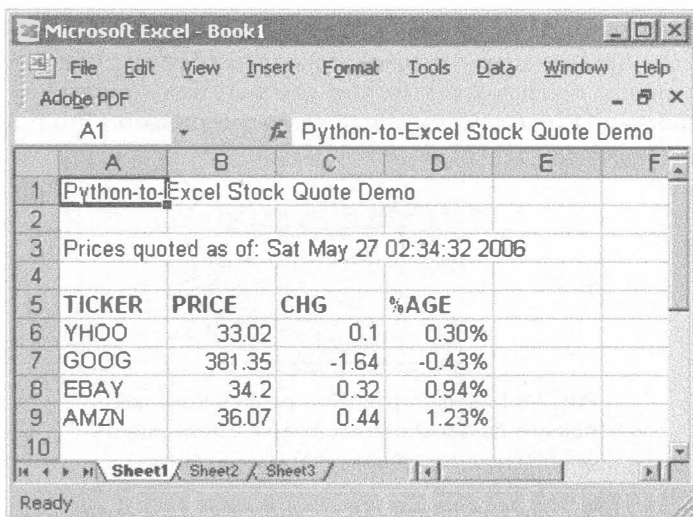
Как и прежде, открывается указатель URL (строка 34), но, вместо того чтобы просто передать данные в стандартное устройство вывода, происходит заполнение ячеек электронной таблицы так, что каждый раз происходит запись данных в один столбец, а для каждой компании отводится одна строка таблицы (строки 35–42).

Строки 45–51

В оставшейся части сценария применяется код, который уже рассматривался ранее. На рис. 7.7 показано окно с реальными данными после выполнения нашего сценария.

Следует учитывать, что столбцы данных теряют свое исходное форматирование как числовых строк, поскольку в Excel эти данные сохраняются как числа с использованием заданного по умолчанию формата ячейки. В частности, потеряно форматирование дробных чисел с двумя знаками после десятичной точки. Например, отображается 34.2, несмотря на то, что из сценария Python передано значение 34.20. Что же касается изменений в столбце с данными на момент предыдущего закрытия, то потеряны не только десятичные позиции, но и знак “плюс” (+), который указывает на

изменение котировок в большую сторону. (Сравните вывод, отображаемый в документе Excel, с выводом из исходной текстовой версии, которая приведена в примере 13.1, `stock.py`. Эти недостатки должны быть устранены в упражнении в конце этой главы.)



	A	B	C	D	E	F
1	Python-to-Excel Stock Quote Demo					
2						
3	Prices quoted as of: Sat May 27 02:34:32 2006					
4						
5	TICKER	PRICE	CHG	%AGE		
6	YHOO	33.02	0.1	0.30%		
7	GOOG	381.35	-1.64	-0.43%		
8	EBAY	34.2	0.32	0.94%		
9	AMZN	36.07	0.44	1.23%		
10						

Рис. 7.7. Пример вывода демонстрационного сценария управления приложением Excel из сценария получения котировок акций на языке Python (`estock.pyw`)

7.4.2. Программа Outlook

Прежде всего мы стремились предоставить читателям примеры сценариев для работы с приложением Outlook, которые показывают, как манипулировать данными адресной книги, отправлять и получать электронную почту. Однако с учетом всех проблем защиты, возникающих при работе с приложением Outlook, было решено не рассматривать эти функции и вместе с тем предоставить читателям весьма полезный пример.

Начнем с того, что программистам, которые постоянно трудятся над созданием приложений с интерфейсом командной строки, часто приходится использовать текстовые редакторы того или иного типа для упрощения своей работы. По поводу выбора наиболее подходящих текстовых редакторов существуют разные мнения, но, не углубляясь в дискуссии по этому поводу, отметим, что эти инструменты включают Emacs, vi (или его современный преемник vim, или gVim) и пр. Если пользователям этих инструментов приходится редактировать ответ на письмо по электронной почте в диалоговом окне Outlook, то они бывают вынуждены вместо привычной среды используемого редактора переходить в отнюдь не такую удобную среду. Для выхода из этой ситуации можно воспользоваться языком Python.

Рассмотрим следующий простой сценарий, в основу которого лег первоначальный замысел, предложенный Джоном Класса (John Klassa) в 2001 году: перед подготовкой ответа на сообщение электронной почты в приложении Outlook запускается предпочтительный редактор пользователя, который содержит текст ответа на письмо по электронной почте в текущем диалоговом окне редактирования; пользователь получает возможность выполнить все остальные операции редактирования в своем редакторе, а затем выйти из этой программы. Вслед за этим происходит замена

содержимого диалогового окна Outlook тем текстом, в котором только что были внесены изменения. Пользователю остается лишь щелкнуть на кнопке Send (Отправить).

Этот инструмент можно вызвать из командной строки. В данной книге он именуется как `outlook_edit.pyw`. Расширение `.pyw` используется для указания на то, что приложение должно выполняться не в терминальном режиме. Иначе говоря, при выполнении этого приложения с графическим интерфейсом пользователя вывод на экран окна терминала подавляется. Прежде чем перейти к описанию кода, рассмотрим, как он работает. После запуска приложения открывается его простой графический интерфейс пользователя, как показано на рис. 7.8.

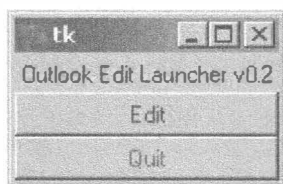


Рис. 7.8. Панель управления редактором электронной почты Outlook с графическим интерфейсом пользователя (`outlook_edit.pyw`)

Если при просмотре электронной почты обнаруживается такое письмо, на которое необходимо ответить, можно щелкнуть на кнопке Reply (Ответить), чтобы вызвать на экран раскрывающееся окно, полностью аналогичное приведенному на рис. 7.9 (разумеется, за исключением содержимого).

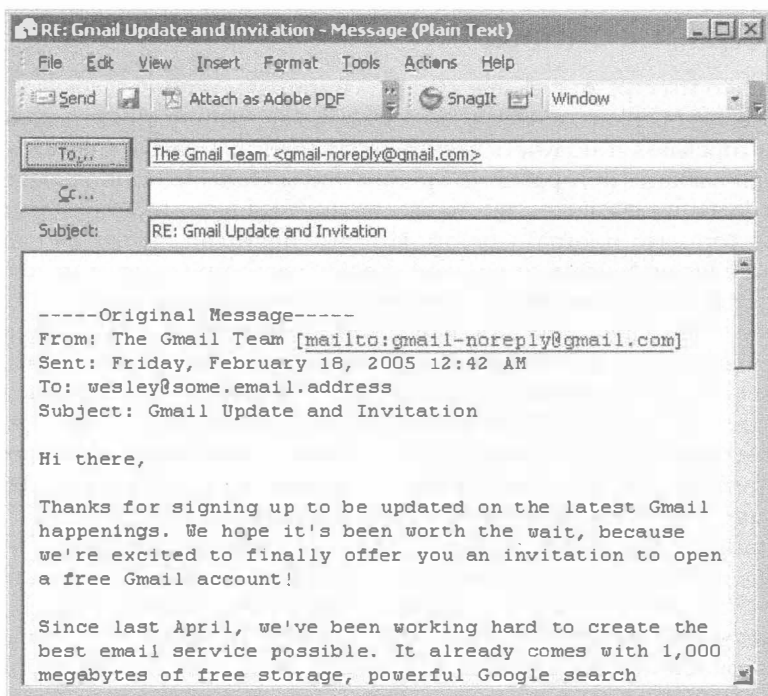


Рис. 7.9. Стандартное диалоговое окно для подготовки ответа в программе Outlook

Теперь предположим, что желательно было бы вместо использования для редактирования этого диалогового окна с ограниченными возможностями выполнять такую задачу в другом редакторе (более предпочтительном). После указания редактора, который должен использоваться со сценарием `outlook_edit.py`, можно щелкнуть на кнопке Edit (Редактировать) графического интерфейса пользователя. В данном примере в качестве такого редактора жестко задан gVim 7.3, но следует отметить, что имя программы редактора можно также задать с помощью переменной среды или предоставить возможность указывать это имя в командной строке самому пользователю (см. связанное с этим упражнение в конце данной главы).

Снимки экранов на рисунках, приведенных в настоящем разделе, были сделаны с использованием версии Outlook 2003. Эта версия Outlook такова, что при обнаружении попытки доступа к ней из внешнего сценария отображается диалоговое окно с предупреждением, показанное на рис. 7.5. После того как пользователь подтверждает допустимость затребованной операции, открывается новое диалоговое окно gVim, в котором находится содержимое диалогового окна Outlook, предназначенного для подготовки ответа. Для рассматриваемого случая пример приведен на рис. 7.10.

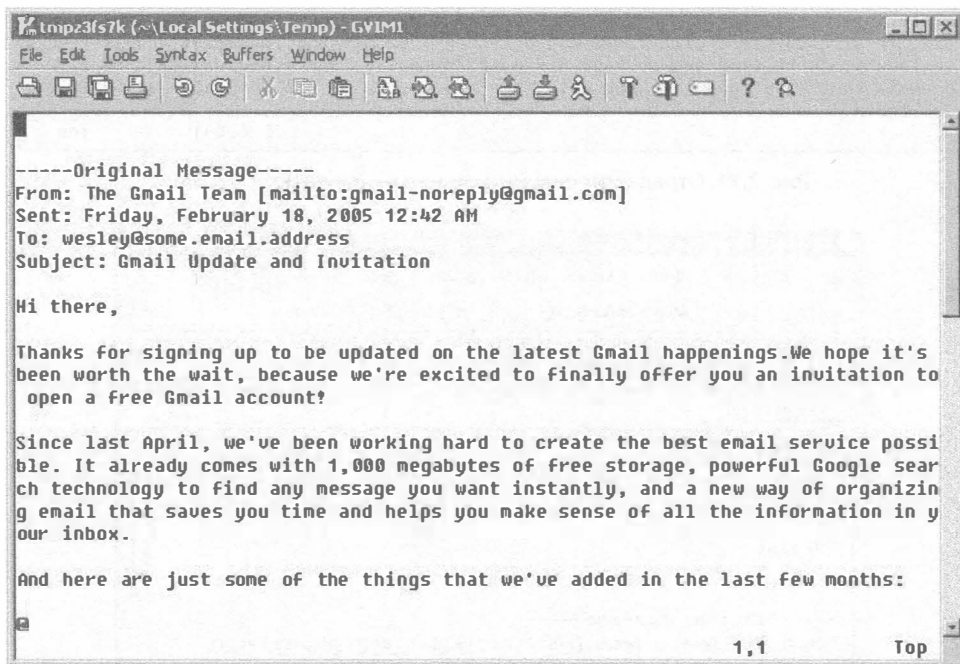


Рис. 7.10. Содержимое диалогового окна Outlook в открывшемся окне редактора gVim

После развертывания окна редактора появляется возможность ввести ответ и при желании откорректировать любую другую часть сообщения. Таким образом, было создано быстродействующее и удобное приложение для подготовки ответа (рис. 7.11). После сохранения файла и выхода из редактора это окно редактирования закрывается и содержание ответа перемещается в диалоговое окно подготовки ответа программы Outlook (рис. 7.12). Итак, нам удалось вместо программы Outlook воспользоваться для работы над ответом более удобным приложением. Для отправки готового письма осталось лишь щелкнуть на кнопке Send, т.е. наша задача выполнена!

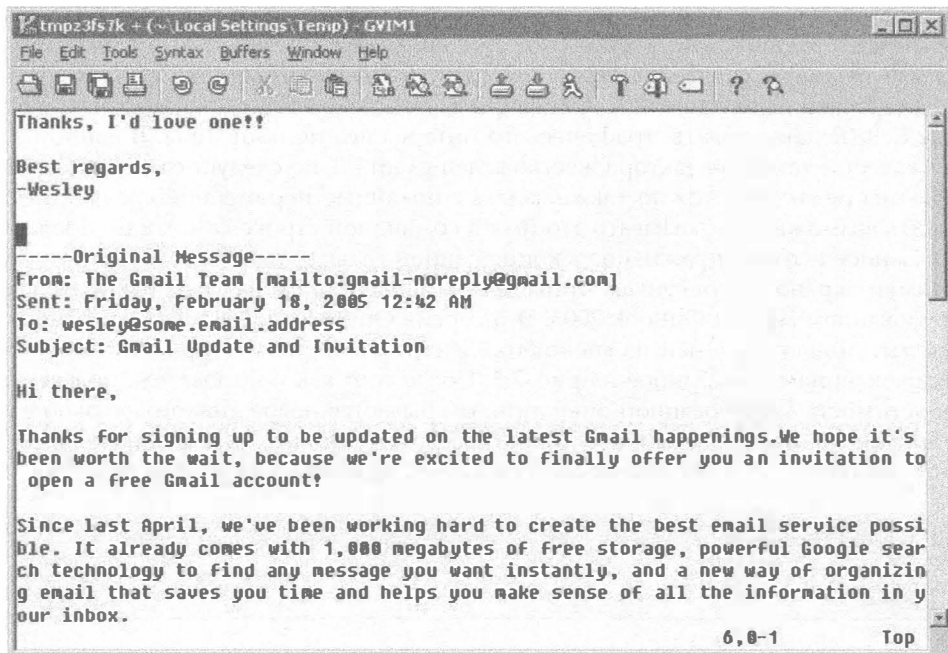


Рис. 7.11. Отредактированный ответ в окне редактора gVim

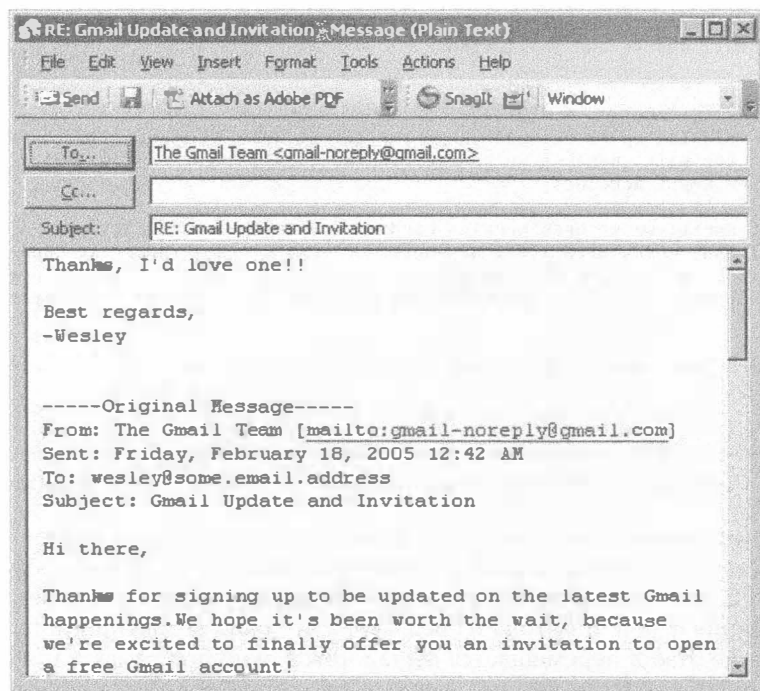


Рис. 7.12. Возврат к диалоговому окну Outlook после внесения изменений в содержимое письма

Теперь приступим к рассмотрению самого сценария, который показан в примере 7.6. Из построения описания кода сценария следует, что он состоит из четырех основных частей: подключения к программе Outlook и захват текущего объекта, над которым осуществляется работа в этой программе; удаление текста из диалогового окна Outlook и передача текста во временный файл; запуск программы редактора с указанием для него в качестве используемого данного временного текстового файла; чтение содержимого отредактированного текстового файла и передача этого содержимого назад в диалоговое окно.

Пример 7.6. Пример применения редактора Outlook (outlook_edit.pyw)

Рассмотрим применяемый способ формирования новых сообщений и редактирования ответных сообщений в диалоговом окне Outlook.

```
1  #!/usr/bin/env python
2
3  from Tkinter import Tk, Frame, Label, Button, BOTH
4  import os
5  import tempfile
6  import win32com.client as win32
7
8  def edit():
9      olook = win32.Dispatch('Outlook.Application')
10     insp = olook.ActiveInspector()
11     if insp is None:
12         return
13     item = insp.CurrentItem
14     if item is None:
15         return
16
17     body = item.Body
18     tmpfd, tmpfn = tempfile.mkstemp()
19     f = os.fdopen(tmpfd, 'a')
20     f.write(body.encode(
21         'ascii', 'ignore').replace('\r\n', '\n'))
22     f.close()
23
24     #ed = r"d:\emacs-23.2\bin\emacsclientw.exe"
25     ed = r"c:\progra-1\vim\vim73\gvim.exe"
26     os.spawnv(os.P_WAIT, ed, [ed, tmpfn])
27
28     f = open(tmpfn, 'r')
29     body = f.read().replace('\n', '\r\n')
30     f.close()
31     os.unlink(tmpfn)
32     item.Body = body
33
34  if __name__ == '__main__':
35      tk = Tk()
36      f = Frame(tk, borderwidth=2)
37      f.pack(fill=BOTH)
38      Label(f,
39          text="Outlook Edit Launcher v0.3").pack()
40      Button(f, text="Edit",
41          fg='blue', command=edit).pack(fill=BOTH)
42      Button(f, text="Quit",
43          fg='red', command=tk.quit).pack(fill=BOTH)
44      tk.mainloop()
```

Построчное объяснение

Строки 1–6

В настоящей главе нет ни одного примера, в котором решающую роль играла бы среда Tk, но эта среда, которая позволяет создать своего рода командный интерпретатор для управления работой интерфейса между пользовательским приложением и целевым приложением Office, широко применяется именно для этой цели. Соответственно, и в этом приложении, как и в других, используется целый ряд констант и графических элементов Tk. Кроме того, возникает необходимость в использовании значительного количества методов для взаимодействия с операционной системой, поэтому в рассматриваемом сценарии осуществляется импорт модуля `os` (вернее, импортируемым модулем является `nt`). Еще один используемый модуль `Python`, `tempfile`, фактически не рассматривался в данной книге, но можно отметить, что с его помощью можно воспользоваться значительным количеством утилит и классов, позволяющих разработчикам создавать временные файлы, имена файлов и каталоги. Наконец, нам следует позаботиться о том, чтобы организовать на ПК взаимодействие с приложениями Office и применяемыми для них серверами COM.

Строки 8–15

Единственные фактически применяемые на персональном компьютере клиентские строки кода COM находятся здесь. С помощью этого кода осуществляется получение дескриптора работающего экземпляра программы Outlook и поиск активного диалогового окна (это должно быть окно `oMailItem`), над которым ведется работа. Если получение дескриптора или поиск текущего объекта оканчивается неудачей, происходит выход из приложения без формирования сообщений. Пользователь узнает об этом по тому признаку, что кнопка `Edit` больше не выделяется серым цветом, а немедленно становится снова доступной (как если бы уже произошел вызов окна редактора и вся работа в нем была закончена).

Заслуживает внимания то, что в данном случае решено использовать динамическую диспетчеризацию вместо статической (вызов `win32.Dispatch()` вместо `win32.gencache.EnsureDispatch()`), поскольку динамическая диспетчеризация обычно обеспечивает более быстрый запуск, а какие-либо дополнительные значения кешируемых констант в этом сценарии не используются.

Строки 16–22

В этом разделе кода после обнаружения текущего диалогового окна (в котором подготавливается новое письмо или редактируется ответ) прежде всего происходит захват текста и запись его во временный файл. Практика показывает, что в подобных приложениях не следует обрабатывать текст, закодированный в Юникоде или содержащий диакритические символы, поэтому в этом диалоговом окне все символы, отличные от ASCII, отфильтровываются. (В одном из упражнений в конце данной главы приведены сведения о том, как обойти это ограничение и внести изменения в сценарий, чтобы он правильно работал с Юникодом.)

Редакторы для Unix с самого начала не были предназначены для работы со строками, содержащими в конце пару символов “возврат строки” и “перевод каретки” (`\r\n`), используемых в качестве символов завершения строки в файлах, созданных на персональном компьютере, поэтому еще одна задача обработки, которая должна выполняться до и после редактирования, состоит в преобразовании этих пар символов

в одинарные символы “перевод каретки” перед отправкой файла в редактор, с последующим обратным преобразованием после завершения редактирования. Современные текстовые редакторы лучше приспособлены для обработки пар символов `\r\n`, поэтому столь существенная проблема, как в прошлом, не возникает.

Строки 24–26

Рассмотрим внимательно, как происходит указанное преобразование: после задания применяемого редактора (в строке 25, в которой указано местонахождение исполняемой программы `vim` в системе; пользователи редактора Emacs могут применить примерно такую команду, как показано в закомментированной строке 24) запускается редактор с указанием в качестве параметра имени временного файла (при условии, что редактор воспринимает в качестве имени целевого файла, указанного в командной строке, первый параметр после имени программы редактора). Для этого применяется вызов `os.spawnv()`, как показано в строке 26.

Флаг `P_WAIT` используется для приостановки основного (родительского) процесса до тех пор, пока не завершится порожденный (дочерний) процесс. Иными словами, необходимо, чтобы кнопка `Edit` была выделена серым цветом, чтобы пользователь не мог предпринять попытку внести изменения одновременно в несколько ответов. На первый взгляд это выглядит как ненужное ограничение, но фактически помогает пользователю сосредоточиться на одном деле и не накапливать на рабочем столе массу частично отредактированных ответов.

К сказанному следует добавить, что применение флага `P_WAIT` в вызове `spawnv()` допускается в операционных системах обоих типов, `POSIX` и `Windows`, как и флага `P_NOWAIT` (который задает прямо противоположное требование — не ожидать завершения дочернего процесса и выполнять оба процесса параллельно). В вызове `spawnv()` допускается также применение еще двух флагов, `P_OVERLAY` и `P_DETACH`, но лишь в операционной системе `Windows`. Действие флага `P_OVERLAY` приводит к тому, что дочерний процесс замещает родительский, как и при использовании вызова `exec()` в операционной системе `POSIX`, а флаг `P_DETACH`, как и флаг `P_NOWAIT`, запускает дочерний процесс параллельно с родительским, за исключением того, что этот запуск происходит в фоновом режиме с отсоединением от клавиатуры или консоли.

В одном из упражнений в конце данной главы поставлена задача сделать эту часть кода более гибкой. Как было указано выше, необходимо обеспечить возможность указывать предпочтительный редактор непосредственно в командной строке или с помощью переменной среды.

Строки 28–32

В следующем блоке кода показано, как открыть обновленный временный файл после закрытия редактора, получить его содержимое, удалить временный файл и заменить текст в диалоговом окне. Заслуживает внимания то, что при этом происходит просто передача данных в программу Outlook; это не позволяет исключить выполнение программой Outlook непредвиденных операций обработки сообщения. Иначе говоря, может возникнуть целый ряд побочных эффектов, таких как добавление (повторное) подписи к письму, удаление символов перевода строки и т.д.

Строки 34–44

В основе этого приложения лежит функция `main()`, в которой используются средства `Tk(inter)` для вывода простого пользовательского интерфейса с отдельной рамкой,

содержащей надпись с описанием приложения и парой кнопок. После щелчка на кнопке Edit происходит запись редактора в активном диалоговом окне Outlook, а после щелчка на кнопке Quit (Выйти) данное приложение завершается.

7.4.3. Программа PowerPoint

Последний пример представляет собой приложение, которое в большей степени подходит для использования на практике. Это приложение пользователи Python попросили меня разработать много лет тому назад, и я счастлив сообщить представителям этого сообщества, что в конечном итоге готов его предоставить в общее пользование. Те, кто когда-либо были свидетелями того, как я провожу презентации на конференциях, по-видимому, обратили внимание на мою любимую уловку. Я демонстрирую аудитории изложение в виде текста того, что я говорю, хотя это может показаться неожиданным и шокирующим для некоторых из присутствующих, которым все равно приходится выслушивать мои речи.

Для этого я запускаю данный сценарий для обработки текстового файла с моим выступлением и применяю мощные средства Python для автоматического формирования презентации PowerPoint, разметки текста с помощью стилей, а затем запуска демонстрации слайдов. Все это вызывает крайнее изумление у моих слушателей. Те, кто понимает, что для достижения такого эффекта достаточно применить небольшой, несложный в написании сценарий Python, испытывают в большей степени не удивление, а удовлетворение от мысли, что они могут сделать то же самое!

Рассматриваемый сценарий действует следующим образом. Прежде всего разворачивается графический интерфейс пользователя (рис. 7.13, а) с запросом к пользователю указать местонахождение текстового файла. Если пользователь вводит допустимые сведения о местонахождении файла, в сценарии начинается обработка этого файла, но если файл не удастся найти или в качестве имени файла указано DEMO, то начинается показ демонстрационной версии. Если имя файла задано, но по какой-то причине его не удастся открыть в приложении, то в поле ввода текста вставляется строка с текстом "DEMO" и выводится ошибка, указывающая, что файл нельзя открыть (рис. 7.13, б).

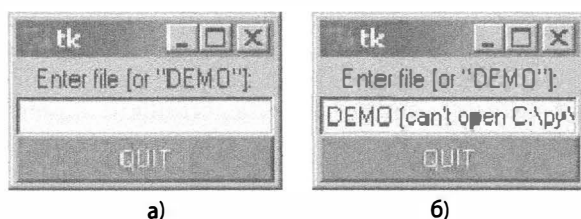


Рис. 7.13. Панель управления приложением для преобразования текста в презентацию PowerPoint с графическим интерфейсом пользователя (txt2ppt.pyw): а) очистка поля ввода имени файла после запуска; б) отображение текста "DEMO", если запрашивается демонстрационная версия или возникает какая-либо ошибка

Как показано на рис. 7.14, следующий шаг состоит в подключении к существующему приложению PowerPoint, работающему в данный момент (или запуск такового, если оно еще не эксплуатируется, а затем получение дескриптора этого приложения), создании слайда с названием презентации (с применением к названию слайда функции преобразования в прописные буквы, ALL CAPS) и дальнейшем создании всех прочих слайдов на основе содержимого текстового файла, для форматирования которого применяется стиль, напоминающий Python.

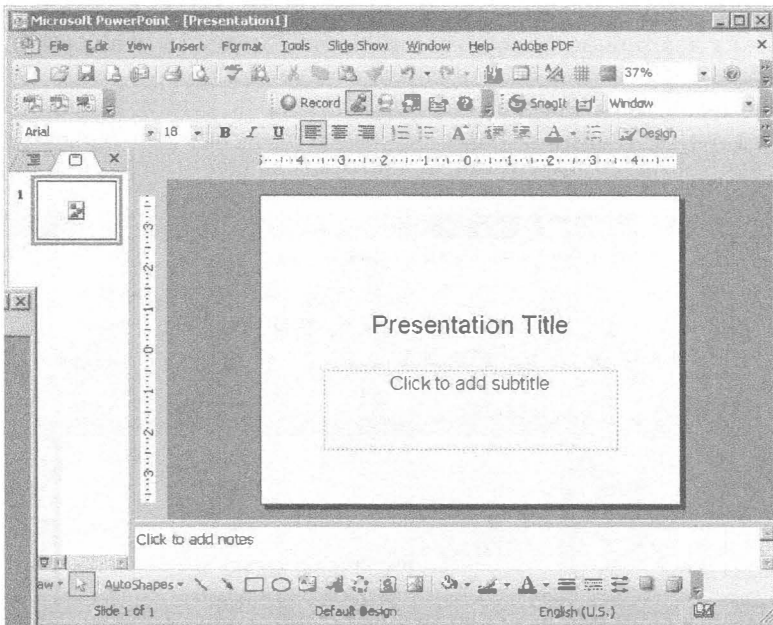


Рис. 7.14. Создание слайда с названием для демонстрационной презентации PowerPoint

На рис. 7.15 показан один из этапов выполнения сценария, в котором создается заключительный слайд демонстрационной версии. При получении снимка этого экрана последняя строка еще не была добавлена к слайду (иными словами, неполный вывод не является следствием отказа при выполнении кода).

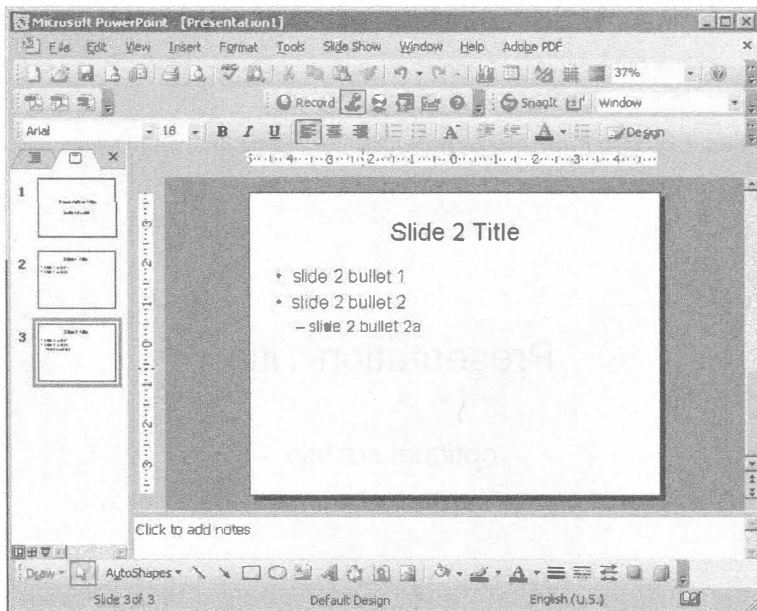


Рис. 7.15. Создание заключительного слайда демонстрационной презентации

Наконец, в коде предусмотрено добавление еще одного вспомогательного слайда, который служит для пользователя указанием, что слайды подготовлены для показа (рис. 7.16), и включает небольшой обратный отсчет от трех до нуля. (Этот снимок экрана сделан после того, как отсчет уже был начат и дошел до двух.) После этого запускается демонстрация слайдов без какой-либо дополнительной обработки. На рис. 7.17 представлен простейший вариант вывода (черный текст на белом фоне).

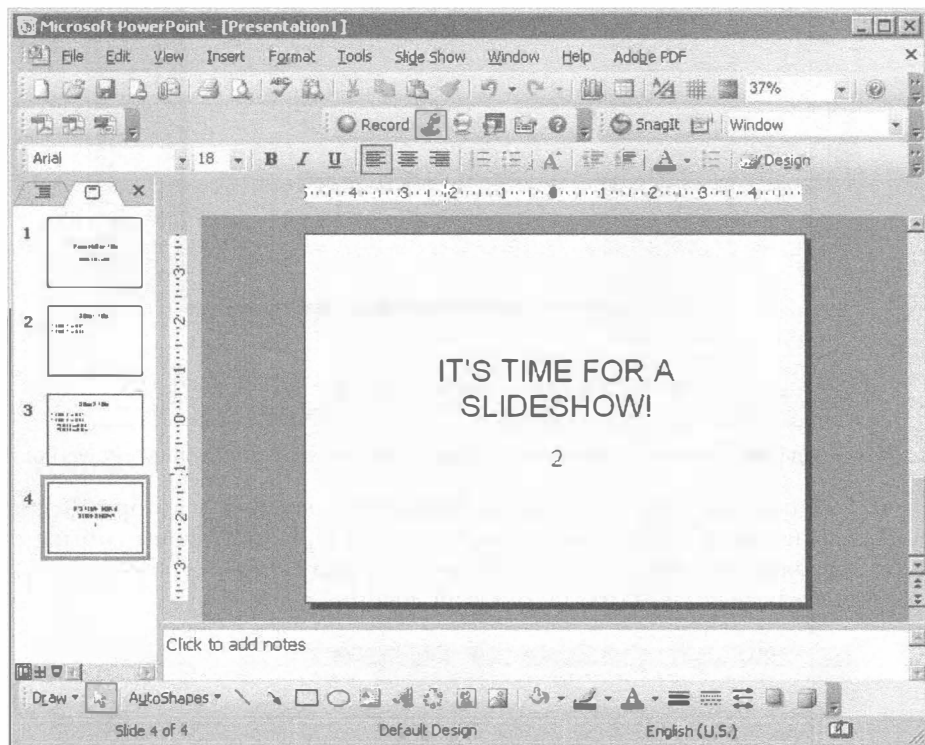


Рис. 7.16. Обратный отсчет перед запуском демонстрации слайдов

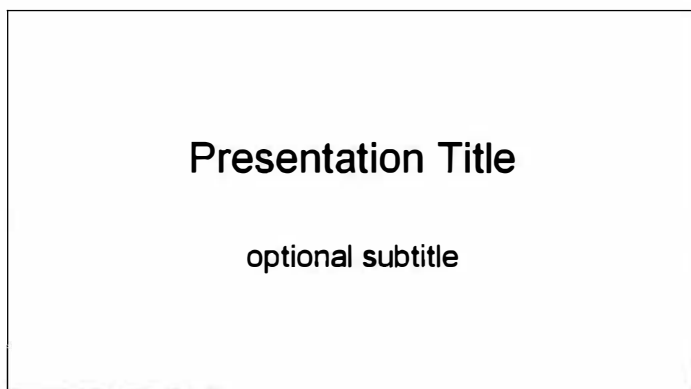


Рис. 7.17. Демонстрация слайдов началась, но шаблон (еще) не был применен

Наконец, рассмотрим, как действует шаблон презентации, предназначенный для придания желаемого внешнего вида тому, что отображается на экране (рис. 7.18), и перейдем к описанию сценария.



Рис. 7.18. Окончательный вид одного из экранов демонстрации слайдов PowerPoint после применения шаблона

В примере 7.7 представлен сценарий `txt2ppt.pyw`, за которым следует описание кода этого сценария.

Пример 7.7. Сценарий преобразования текста в презентацию PowerPoint (`txt2ppt.pyw`)

В этом сценарии формируется презентация PowerPoint из текстового файла с применением форматирования в стиле, напоминающем код Python.

```
1  #!/usr/bin/env python
2
3  from Tkinter import Tk, Label, Entry, Button
4  from time import sleep
5  import win32com.client as win32
6
7  INDENT = '    '
8  DEMO = '''
9  PRESENTATION TITLE
10     optional subtitle
11
12     slide 1 title
13         slide 1 bullet 1
14         slide 1 bullet 2
15
16     slide 2 title
17         slide 2 bullet 1
18         slide 2 bullet 2
19             slide 2 bullet 2a
20             slide 2 bullet 2b
21     '''
```

```

22
23 def txt2ppt(lines):
24     ppoint = win32.gencache.EnsureDispatch(
25         'PowerPoint.Application')
26     pres = ppoint.Presentations.Add()
27     ppoint.Visible = True
28     sleep(2)
29     nslide = 1
30     for line in lines:
31         if not line:
32             continue
33         linedata = line.split(INDENT)
34         if len(linedata) == 1:
35             title = (line == line.upper())
36             if title:
37                 stype = win32.constants.ppLayoutTitle
38             else:
39                 stype = win32.constants.ppLayoutText
40
41             s = pres.Slides.Add(nslide, stype)
42             ppoint.ActiveWindow.View.GotoSlide(nslide)
43             s.Shapes[0].TextFrame.TextRange.Text = line.title()
44             body = s.Shapes[1].TextFrame.TextRange
45             nline = 1
46             nslide += 1
47             sleep((nslide<4) and 0.5 or 0.01)
48         else:
49             line = '%s\r\n' % line.lstrip()
50             body.InsertAfter(line)
51             para = body.Paragraphs(nline)
52             para.IndentLevel = len(linedata) - 1
53             nline += 1
54             sleep((nslide<4) and 0.25 or 0.01)
55
56     s = pres.Slides.Add(nslide, win32.constants.ppLayoutTitle)
57     ppoint.ActiveWindow.View.GotoSlide(nslide)
58     s.Shapes[0].TextFrame.TextRange.Text = "It's time for a slideshow!".upper()
59     sleep(1.)
60     for i in range(3, 0, -1):
61         s.Shapes[1].TextFrame.TextRange.Text = str(i)
62         sleep(1.)
63
64     pres.SlideShowSettings.ShowType = win32.constants.ppShowTypeSpeaker
65     ss = pres.SlideShowSettings.Run()
66     pres.ApplyTemplate(r'c:\Program Files\Microsoft Office\Templates\Presentation
Designs\Stream.pot')
67     s.Shapes[0].TextFrame.TextRange.Text = 'FINIS'
68     s.Shapes[1].TextFrame.TextRange.Text = ''
69
70 def _start(ev=None):
71     fn = en.get().strip()
72     try:
73         f = open(fn, 'U')
74     except IOError, e:
75         from cStringIO import StringIO
76         f = StringIO(DEMO)
77         en.delete(0, 'end')
78     if fn.lower() == 'demo':

```

```
79         en.insert(0, fn)
80     else:
81         import os
82         en.insert(0,
83             r"DEMO (can't open %s: %s)" % (
84                 os.path.join(os.getcwd(), fn), str(e))
85         en.update_idletasks()
86     txt2ppt(line.rstrip() for line in f)
87     f.close()
88
89     if __name__ == '__main__':
90         tk = Tk()
91         lb = Label(tk, text='Enter file [or "DEMO"]:')
92         lb.pack()
93         en = Entry(tk)
94         en.bind('<Return>', _start)
95         en.pack()
96         en.focus_set()
97         quit = Button(tk, text='QUIT',
98             command=tk.quit, fg='white', bg='red')
99         quit.pack(fill='x', expand=True)
100        tk.mainloop()
```

Построчное объяснение

Строки 1–5

Любопытной особенностью этого сценария является то, что в нем не приходится импортировать много модулей. Почти все, что требуется для решения поставленной задачи, предусмотрено в самом языке Python. Как и в описанном выше редакторе содержимого диалогового окна программы Outlook, в этом приложении с графическим интерфейсом пользователя требуется применение некоторых несложных функциональных средств Tk для получения данных, введенных пользователем. Разумеется, пользователь данного сценария мог бы также пожелать, чтобы для ввода данных применялся интерфейс командной строки, но этот вариант здесь не рассматривается, поскольку в настоящей книге уже не раз было подробно описано, как это сделать. Иногда более удобной становится такая организация работы, при которой заранее происходит запуск всех необходимых инструментов, чтобы пользователю оставалось лишь обращаться к тем, которые нужны в данный момент.

Классическим примером приложений, организованных подобным образом, являются приложения, в которых широко используется функция `time.sleep()`. В рассматриваемом примере эта функция служит для замедления некоторых действий, выполняемых приложением. По желанию читатель может исключить все вызовы этой функции. Причина, по которой она здесь используется, как и в приведенном выше примере программы обработки котировок акций с помощью программы Excel, состоит в том, что для наблюдения за функционированием программы требуется немного замедлить ее работу, поскольку в противном случае код выполняется так быстро, что на первый взгляд кажется, будто программа ничего не делает или просто выводит заранее подготовленные данные.

Последние несколько строк, безусловно, представляют интерес, поскольку определяют на персонального компьютера нечто подобное программным ресурсам.

Строки 7–21

В этой части сценария определена пара общих глобальных переменных, которые представляют два значения. Первая переменная определяет применяемую по умолчанию величину отступа, которая соответствует четырем пробелам, полностью аналогично рекомендуемому в документе PEP 8 (в руководстве по стилю) отступу для кода Python, за исключением того, что на этот раз определяется величина отступа для списков разного уровня. Вторая переменная задает структуру демонстрации слайдов для тех случаев, если пользователь пожелает ознакомиться с работой сценария или завершится неудачей поиск исходного текстового файла, который требуется для сценария. Строковая константа, которая определена этой переменной, может также служить примером того, как следует задавать структуру исходного текстового файла. По такому же образцу должны создаваться все прочие презентации.

Строки 23–29

Это первые несколько строк основной функции, `txt2ppt()`. В данной части кода происходит запуск приложения PowerPoint, создается новая презентация, осуществляется вывод результатов выполнения приложения PowerPoint на экран, приложение приостанавливается на несколько секунд, а затем количество слайдов сбрасывается и становится равным одному.

Строки 30–54

Функция `txt2ppt()` принимает один параметр: все строки исходного текстового файла, представляющие собой текст презентации. Эта функция определена так, что позволяет также задать в качестве входного параметра любой текстовый объект, допускающий обработку в цикле, который включает одну или несколько строк, после чего происходит автоматическое формирование презентации слайдов. Для демонстрации маркированных списков используется объект `cStringIO.StringIO`, с помощью которого осуществляется обработка текста в цикле, а для замены реального файла применяется выражение-генератор для каждой строки. Разумеется, если используется Python 2.3 или предшествующая версия, то вместо выражения-генератора необходимо применить расширение списка. Следует отметить такой недостаток сценария, как непроизводительное расходование оперативной памяти, особенно для обработки больших исходных файлов. Впрочем, этот недостаток можно устранить.

Снова обратимся к основному циклу обработки; в нем исключаются пустые строки, после чего применяется дополнительная обработка в виде разбиения строк вслед за применением отступов. Рассмотрим внимательно следующий фрагмент кода, который позволяет полностью разобраться в том, какие действия происходят в программе:

```
>>> 'slide title'.split(' ')
['slide title']
>>> '    1st level bullet'.split(' ')

['', '1st level bullet']
>>> '        2nd level bullet'.split(' ')
['', '', '2nd level bullet']
```

Если отступ отсутствует, то разбиение строки по отступу не приводит к появлению дополнительных строк. Это означает, что обнаружено начало нового слайда, а

текст представляет собой название слайда. Если список состоит больше чем из одной строки, то при его обработке должен быть применен по меньшей мере один отступ, и мы имеем дело с продолжением материала предыдущего слайда (а не с началом нового). В первом случае происходит выполнение той части конструкции `if`, которая соответствует истинному условию, т.е. строки 35–47. Вначале рассмотрим этот блок, затем — последнюю часть условной конструкции.

Следующие пять строк (35–39) позволяют определить, обрабатывается ли слайд с названием или обычный текстовый слайд. Именно здесь все буквы названия слайда преобразуются в прописные (ALL CAPS). В операторе сравнения учитывается, что в одном из операндов все буквы должны быть прописными. Если обнаруживается соответствие, то текст представлен в виде прописных букв, а это означает, что для данного слайда должен применяться стиль названия, заданный константой `ppLayoutTitle` персонального компьютера. В противном случае это обычный слайд с названием и текстовым содержимым (`ppLayoutText`).

После определения стиля слайда создается новый слайд (строка 41) и передается в приложение PowerPoint (строка 42), слайд обозначается как активный, а его название или основной образующий текст определяется как содержимое, для которого используется стиль названия (строка 43). Следует отметить, что в индексруемых объектах Python отсчет начинается с нуля (`Shape[0]`), а в приложениях Microsoft начальным индексом является единица (`Shape(1)`), и это учтено в сценарии.

Оставшееся содержимое, подлежащее рассмотрению, относится к категории `Shape[1]` (или `Shape(2)`). В данном случае он обрабатывается по такому же принципу, как текст (строка 44); если бы это был слайд с названием, то содержимое рассматривалось бы как подзаголовок, а на стандартном слайде соответствовало бы строкам текста, обозначенным маркером.

Остальные строки этой конструкции (45–47) применяются для формирования отметки, которая показывает, что произведена запись первой строки на текущем слайде, увеличение значение счетчика, который отслеживает общее количество слайдов в презентации, а затем выполнена приостановка, чтобы пользователь мог наблюдать за действиями по управлению выполнением PowerPoint, осуществляемыми в сценарии Python.

На следующей итерации цикла происходит переход к конструкции `else` и выполняется код, относящийся к остальной части списка на том же слайде, в результате чего заполняется вторая текстовая или графическая составляющая слайда. На предыдущем этапе выполнения программы уже использовался отступ для задания местоположения текста и был определен уровень отступа, поэтому ведущие пробелы в тексте больше не требуются. Таким образом, эти пробелы удаляются из текстовой строки (с помощью `str.lstrip()`), а затем строка вставляется в текст слайда (строки 49–50).

В остальной части блока текст обозначается отступом, соответствующим текущему уровню маркированного списка (или отступы полностью исключаются, если текущий слайд представляет собой слайд с названием, для которого задан уровень отступа, равный нулю, не оказывающий какого-либо воздействия на сдвиг текста), увеличивается значение количества строк и в конце устанавливается небольшая пауза, позволяющая замедлить работу программы настолько, чтобы за ней мог наблюдать пользователь (строки 51–54).

Строки 56–62

После создания всех основных слайдов в конце добавляется еще один слайд с названием, указывающий, что вскоре начнется демонстрация слайдов. При этом текст изменяется динамически в виде обратного отсчета секунд от трех до нуля.

Строки 64–68

Основное назначение этих строк состоит в подготовке к запуску демонстрации слайдов. Фактически запуск показа слайдов осуществляется только с помощью первых двух строк (64 и 65). Строка 66 обеспечивает применение шаблона. Эта операция выполняется после начала демонстрации слайдов, чтобы можно было видеть ее результаты; это — более наглядный способ ознакомления с программой. Последние две строки в этом блоке кода (67–68) применяются для переустановки номера слайда (поскольку наступило время начать демонстрацию слайдов) и выполнения обратного отсчета, как указано выше.

Строки 70–100

Функция `_start()` применяется только после запуска сценария из командной строки. Функция `txt2ppt()` определяется как импортируемая, что позволяет использовать ее в другом сценарии, а что касается функции `_start()`, то для нее требуется графический интерфейс пользователя. Перейдя на время к строкам 90–100, можно увидеть, что применяется графический интерфейс пользователя среды Tk с полем ввода текста (с надписью, приглашающей пользователя задать имя файла или ввести слово DEMO, чтобы посмотреть демонстрацию) и кнопкой Quit.

Итак, работа функции `_start()` начинается (в строке 71) с извлечения содержимого этого поля ввода и попытки открыть указанный файл (строка 73; см. соответствующее упражнение в конце главы). Если операция открытия файла выполняется успешно, то происходит пропуск конструкции `except`, вызывается функция `txt2ppt()` для обработки файла, а затем файл закрывается после завершения этой работы (строки 86–87).

При возникновении исключения вызывается обработчик, который позволяет определить, была ли выбрана демонстрационная версия (строки 77–79). При получении положительного результата строка с определением демонстрации считывается в объект `cStringIO.StringIO` (строка 76) и этот объект передается в функцию `txt2ppt()`; в противном случае (при возникновении исключения) все равно выполняется демонстрация, но в текстовое поле вставляется сообщение об ошибке, чтобы пользователь мог понять, почему произошел отказ (строки 81–84).

7.4.4. Резюме

Хотелось бы надеяться, что изучение этой главы дало возможность читателю наглядно ознакомиться с тем, как осуществляется программирование клиентских сценариев COM на языке Python. Безусловно, наиболее надежными и многофункциональными являются серверы COM, предусмотренные в приложениях Microsoft Office, но приведенный в этой главе материал вполне может применяться для работы с серверами COM других приложений Microsoft и даже приложений OpenOffice (версии StarOffice с открытым исходным кодом), применяемых в качестве альтернативы Microsoft Office.

После приобретения компании Sun Microsystems (спонсора разработки StarOffice и OpenOffice) корпорация Oracle объявила о выпуске преемника StarOffice под названием Oracle Open Office. В связи с этим некоторые представители сообщества разработчиков программ с открытым исходным кодом сочли, что прежний статус OpenOffice как общедоступной программы оказался под угрозой, поэтому приступили к созданию LibreOffice — клона OpenOffice. В основе OpenOffice и LibreOffice

лежит один и тот же код, поэтому в них применяется одинаковый интерфейс в стиле COM, известный под названием UNO (Universal Network Objects). Для управления приложениями OpenOffice или LibreOffice в целях выполнения операций обработки документов, таких как запись PDF-файлов, преобразование из формата Microsoft Word в формат ODT (OpenDocument Text), вывод кода HTML и т.д., может применяться модуль PyUNO.

7.5. Соответствующие модули/пакеты

Расширения Python для Windows

<http://pywin32.sf.net>

xlrd, xlwt (имеется версия для Python 3)

<http://www.lexicon.net/sjmachin/xlrd.htm>

<http://pypi.python.org/pypi/xlwt>

<http://pypi.python.org/pypi/xlrd>

pyExcelexerator

<http://sourceforge.net/projects/pyexcelexerator/>

PyUNO

<http://udk.openoffice.org/python/python-bridge.html>

7.6. Упражнения

- 7.1. *Веб-службы.* Возьмите за основу пример для работы с котировками акций Yahoo! (stock.py) и внесите в приложение такие изменения, чтобы данные котировок сохранялись в файле, а не отображались на экране. Необязательное задание. Можно изменить этот сценарий так, чтобы пользователи могли выбирать — отображать ли данные котировок или сохранять в файле.
- 7.2. *Программа Excel и веб-страницы.* Напишите приложение, которое считывает данные из электронной таблицы Excel и отображает их в эквивалентной таблице HTML. (По желанию можно использовать модуль HTMLgen сторонних разработчиков.)
- 7.3. *Приложения Office и веб-службы.* Создайте интерфейс к любой существующей веб-службе, основанной на применении REST или URL, и предусмотрите запись данных в электронную таблицу Excel или их привлекательное форматирование в документе Word. Подготовьте полученные данные должным образом для печати. **Дополнительное задание.** Обеспечьте поддержку и Excel, и Word.
- 7.4. *Программа Outlook и веб-службы.* Выполните такую же доработку, как указано в упражнении 7.3, но обеспечьте вывод данных в новое сообщение электронной почты, отправляемое с помощью программы Outlook.

Дополнительное задание. Прodelайте такую же работу, но вместо этого предусмотрите отправку электронной почты с помощью обычного сервера SMTP. (Вам может потребоваться еще раз обратиться к главе 3.)

- 7.5. *Подготовка демонстрации слайдов.* В упражнениях 7.15–7.24 предусмотрено добавление новых средств в генератор демонстрации слайдов, описанный ранее в данной главе, `txt2ppt.pyw`. Это упражнение предназначено для того, чтобы читатель обратился к основам выполняемых действий и на время забыл обо всех нестандартных форматах. Реализуйте сценарий с такими же функциональными возможностями, как и в сценарии `txt2ppt.pyw`, но для вывода вместо интерфейса к PowerPoint должен применяться открытый стандарт, такой как HTML5. В качестве образца можно воспользоваться такими проектами, как LandSlide, DZSlides и HTML5Wow. Подобные примеры можно найти по адресу http://en.wikipedia.org/wiki/Web-based_slideshow. Разработайте спецификацию формата вывода простого текста, которую могли бы применять ваши пользователи, оформите ее в виде документа, разработайте необходимую программу и предоставьте своим пользователям возможность обращаться к этому инструменту для подготовки форматированного вывода собственных демонстраций.
- 7.6. *Outlook, базы данных и адресная книга.* Напишите программу, которая позволяет извлекать содержимое адресной книги Outlook и сохранять данные необходимых полей в базе данных. В качестве базы данных можно применить текстовый файл, файл DBM или реляционную СУБД. (Вам может потребоваться обратиться к главе 6.) **Дополнительное указание.** Сделайте обратное; считывайте информацию о контактах из базы данных (или предоставьте пользователю возможность непосредственно вводить данные) и создавайте или обновляйте записи в программе Outlook.
- 7.7. *Программа Microsoft Outlook и электронная почта.* Разработайте программу, позволяющую создавать резервную копию электронной почты, выбирая содержимое папки Inbox (Входящие) и/или других важных папок и сохраняя его в обычном формате mbox (или в аналогичном формате) на диске.
- 7.8. *Календарь Outlook.* Напишите простой сценарий, который создает новые записи о намеченных встречах в Outlook. Предусмотрите ввод пользователем по меньшей мере следующих данных: дата и время начала встречи, имя собеседника или тема беседы, а также продолжительность встречи.
- 7.9. *Календарь Outlook.* Подготовьте приложение, которое выводит содержимое календаря намеченных встреч в указанное место назначения, например, на экран, в базу данных, электронную таблицу Excel и т.д. **Дополнительное задание.** Предусмотрите выполнение тех же функций в запрограммированных вами задачах Outlook.
- 7.10. *Многопоточная обработка.* Внесите в версию для Excel сценария загрузки котировок акций (`estock.pyw`) такие изменения, чтобы загрузка данных происходила параллельно с использованием нескольких потоков Python. **Необязательное задание.** Можно также попытаться выполнить это упражнение с использованием потоков Visual C++, подключив модуль `win32process.beginthreadex()`.

7.11. Форматирование ячейки Excel. На рис. 7.7 показано, что в версии сценария загрузки котировок акций на основе электронной таблицы (estock.pyw) цены на акции не отображаются по умолчанию с двумя знаками после десятичной точки, даже если передается строка с заключительными нулями. Дело в том, что при преобразовании в Excel строковых данных в числовые для числового формата используется настройка по умолчанию.

- а) Измените числовой формат, чтобы должным образом формировались два десятичных знака после точки, для чего необходимо откорректировать атрибут `NumberFormat` ячейки, чтобы он стал равным `0.00`.
- б) Очевидно также, что в столбце изменений после предыдущего закрытия не только искажается форматирование после десятичной точки, но и теряется знак “плюс” (+). Но внесение исправлений, предусмотренных в пункте (а), касающихся обоих столбцов, приводит к исправлению только нарушения формирования десятичных позиций, но знак “плюс” автоматически исключается в любом положительном числе. Решение состоит в том, чтобы такие столбцы были текстовыми, а не числовыми. Это можно сделать, изменив атрибут `NumberFormat` ячейки на `@`.
- в) Изменение числового формата ячейки на текстовый приводит к потере выравнивания по правому краю, которое автоматически применяется для числовых данных. Поэтому в дополнение к решению по пункту (б) необходимо также задать в качестве значения атрибута `HorizontalAlignment` ячейки константу `xlRight` Excel на персональном компьютере. После выполнения задания по всем трем пунктам полученный вывод должен выглядеть более приемлемым, как показано на рис. 7.19.

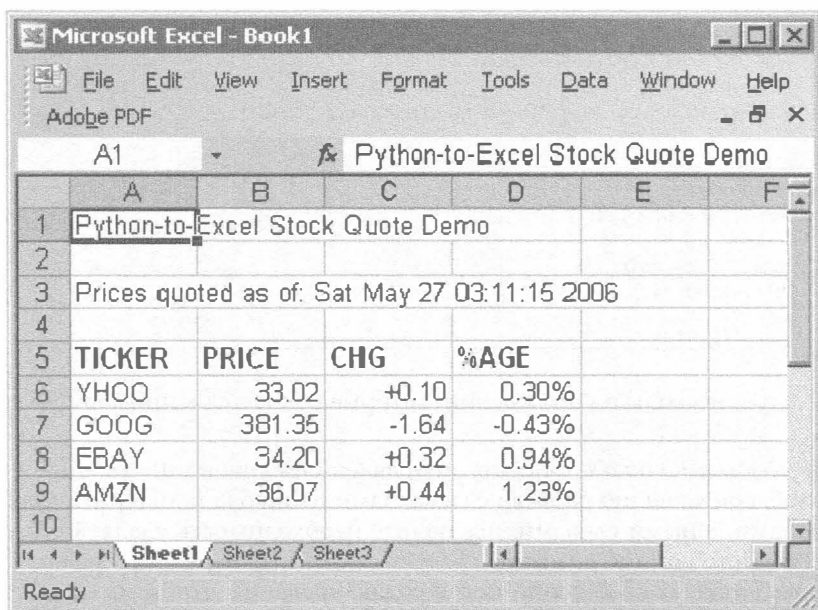


Рис. 7.19. Результаты усовершенствования сценария Python для вывода котировок акций в электронную таблицу Excel (estock.pyw)

- 7.12.** *Python 3.* В примере 7.8 показана версия для Python 3 первого примера работы с программой Excel (`excel3.pyw`) наряду с изменениями, которые выделены курсивом. Пользуясь этим решением, перенесите все прочие сценарии этой главы в Python 3.

Пример 7.8. Пример работы с программой Excel в версии Python 3 (`excel3.pyw`)

Для переноса исходного сценария `excel.pyw` достаточно вызвать на выполнение инструмент `2to3`.

```

1  #!/usr/bin/env python3
2
3  from time import sleep
4  from tkinter import Tk
5  from tkinter.messagebox import showwarning
6  import win32com.client as win32
7
8  warn = lambda app: showwarning(app, 'Exit?')
9  RANGE = list(range(3, 8))
10
11 def excel():
12     app = 'Excel'
13     xl = win32.gencache.EnsureDispatch('%s.Application' % app)
14     ss = xl.Workbooks.Add()
15     sh = ss.ActiveSheet
16     xl.Visible = True
17     sleep(1)
18
19     sh.Cells(1,1).Value = 'Python-to-%s Demo' % app
20     sleep(1)
21     for i in RANGE:
22         sh.Cells(i,1).Value = 'Line %d' % i
23         sleep(1)
24     sh.Cells(i+2,1).Value = "Th-th-th-that's all folks!"
25
26     warn(app)
27     ss.Close(False)
28     xl.Application.Quit()
29
30 if __name__ == '__main__':
31     Tk().withdraw()
32     excel()

```

Далее следует несколько упражнений, которые относятся к примеру 7.6 (`outlook_edit.pyw`).

- 7.13.** *Поддержка Юникода.* Внесите исправления в сценарий `outlook_edit.pyw`, чтобы он успешно работал с символами Юникода и диакритическими символами. Иными словами, исключите необходимость удалять эти символы. Для этого данные символы следует сохранить, передать в редактор и принять в текстах сообщений после редактирования, чтобы их можно было отправлять в сообщениях электронной почты.
- 7.14.** *Обеспечение надежности.* Добейтесь повышения гибкости сценария, предоставив пользователю возможность указывать свой предпочтительный

редактор в командной строке. Если таковой не указан, то приложение должно извлечь сведения о применяемом редакторе из переменной среды, а если эта переменная не задана, в качестве последнего средства обеспечения нормальной работы вызвать один из жестко заданных редакторов.

Следующий ряд упражнений относится к примеру 7.7 (txt2ppt.pyw).

- 7.15. *Пропуск комментариев.* Внесите такие изменения в сценарий, чтобы он поддерживал комментарии: если строка в текстовом файле начинается со знака диеза (#), называемого также знаком фунта, номера, восьмиконечника, решетки и т.д., примите предположение, что эта строка должна быть пропущена, и перейдите к следующей.
- 7.16. *Применение более удобного обозначения для слайдов с названием.* Предложите лучший способ обозначения слайдов с названием. Можно было бы преобразовывать в прописную первую букву каждого слова, но в некоторых ситуациях применение в названии начальных букв верхнего регистра нежелательно. Например, предположим, что пользователь готовит набор слайдов с названием "Intro to TCP/IP". После такого преобразования заголовок набора будет содержать ошибку из-за преобразования буквы "t" в слове "to" в прописную, а букв "cp" и "p" в "TCP/IP" — в строчные:
- ```
>>> 'Intro to TCP/IP'.title()
'Intro To Tcp/Ip'
```
- 7.17. *Побочные эффекты.* Что произойдет в функции `_start()`, если будет обнаружено наличие в текущей папке текстового файла `demo`? Следует ли рассматривать размещение файла `demo` в папке как возможность усовершенствовать сценарий или реагировать на такую ситуацию как на ошибку? Можно ли как-то воспользоваться подобной возможностью? Если да, то предусмотрите это в коде. Если нет, объясните, почему.
- 7.18. *Спецификация шаблона.* В настоящее время рассматриваемые сценарии таковы, что во всех презентациях применяется шаблон оформления `C:\Program Files\Microsoft Office\Templates\Presentation Designs\Stream.pot`. Это не интересно.
- Предоставьте пользователю возможность выбирать любые другие шаблоны из папки шаблонов или из той папки, которая задана при установке.
  - Предусмотрите для пользователя возможность указывать собственный шаблон (и его местонахождение) с помощью нового поля ввода в графическом интерфейсе пользователя, в командной строке или переменной среды (по выбору). **Дополнительное задание.** Обеспечьте возможность поддержки всех этих вариантов в указанном порядке приоритетов или предусмотрите в пользовательском интерфейсе возможность задавать применяемые по умолчанию варианты выбора шаблона, указанные в пункте а).
- 7.19. *Применение гиперссылок.* Можно предусмотреть вариант с применением в презентации ссылок на простые текстовые файлы. Обеспечьте активизацию этих ссылок в программе PowerPoint. **Подсказка.** Для этого может потребоваться задать значение `Hyperlink.Address` в качестве URL, чтобы открывался браузер со страницей, которую пользователь мог бы посетить,

щелкнув на ссылке в слайде (см. параметр `ActionSettings` для ознакомления со значением `ppMouseClick`). **Дополнительное задание.** Обеспечьте поддержку только гиперссылок, содержащихся в тексте URL, даже если эта ссылка не является единственным текстом в той же строке. Иными словами, предусмотрите обработку лишь активной части ссылки, относящейся к URL, чтобы весь прочий текст в той же строке не мог быть по ошибке обработан как гиперссылка.

- 7.20. Форматирование текста.** Предусмотрите возможность применения к содержимому презентации таких средств форматирования текста, как обозначение его полужирным, курсивным и моноширинным шрифтом (например, Courier). Для этого обеспечьте поддержку применения некоего упрощенного языка разметки для форматирования исходных текстовых файлов. Настоятельно рекомендуется использовать сложившийся стиль форматирования, такой как `reST` (`reStructuredText`), `Markdown` или аналогичный ему (например, стиль форматирования вики-сайтов), и задавать для разметки, допустим, обозначения `"monospaced"`, `*bold*`, `_italic_` и т.д. Для ознакомления с другими примерами стилей см. [http://en.wikipedia.org/wiki/Lightweight\\_markup\\_language](http://en.wikipedia.org/wiki/Lightweight_markup_language).
- 7.21. Форматирование текста.** Дополнительно предусмотрите поддержку других средств форматирования, таких как подчеркивание, затенение, применение разнообразных шрифтов, изменение цвета текста, изменение выключки (влево, вправо, по центру и т.д.), изменение размера шрифта, применение верхних и нижних колонтитулов, а также других средств, поддерживаемых программой PowerPoint.
- 7.22. Изображения.** Еще одним важным средством, которое должно быть предусмотрено в приложении, является возможность формировать слайды с изображениями. Упростим задачу, потребовав, чтобы поддерживались только слайды с названием и одним изображением (откорректированным по размерам и выровненным по центру на слайде презентации). Чтобы пользователи могли задавать в определении презентации имена файлов с изображениями, например, `:IMG:C:/py/talk/images/cover.png`, необходимо предусмотреть специальный синтаксис. **Подсказка.** До сих пор использовались только компоновки слайдов `ppLayoutTitle` или `ppLayoutText`, а для этого упражнения рекомендуется компоновка `ppLayoutTitleOnly`. Для вставки изображений используйте функцию `Shapes.AddPicture()`, а для изменения их размеров — функции `ScaleHeight()` и `ScaleWidth()`, задавая точки данных, предусмотренные в параметрах `PageSetup.SlideHeight` и `PageSetup.SlideWidth`, а также применяя атрибуты изображения `Height` и `Width`.
- 7.23. Различные компоновки.** Внесите дополнительные изменения в свое решение упражнения 7.22, чтобы сценарий мог поддерживать слайды с несколькими изображениями или слайды с изображениями и текстом в виде маркированных списков. Для этого главным образом потребуется применение других стилей компоновки.
- 7.24. Внедрение видеофайлов.** Можно также добавить еще одно дополнительное средство — возможность встраивать в презентации видеоклипы YouTube (или другие видеофайлы в формате Adobe Flash). По аналогии с

упражнением 7.23, необходимо определить собственный синтаксис для поддержки этой возможности, например :VID:<http://youtube.com/v/Tj5UmH5TdfI>. **Подсказка.** И в этом случае рекомендуется компоновка `ppLayoutTitleOnly`. Кроме того, вам потребуется прибегнуть к применению функции `Shapes.AddOLDObject()` с указанием "ShockwaveFlash.ShockwaveFlash.10" или той версии, к которой относится применяемая программа воспроизведения Flash.



ГЛАВА

8

# Создание расширений для языка Python

*В этой главе...*

- Введение/обоснование
- Модули расширения
- Другие темы

*Язык С очень эффективен, но, к сожалению, эта эффективность достигается за счет выполнения большого объема работы по управлению ресурсами низкого уровня. Современные компьютеры — очень мощные, поэтому все усилия в этом направлении обычно неоправданны; гораздо разумнее применять язык, позволяющий более эффективно расходовать рабочее время программиста, пусть даже за счет менее эффективного использования процессорного времени компьютера.*

*В этом и состоит назначение языка Python.*

Эрик Рэймонд (Eric Raymond),  
октябрь 1996 г.

В главе описано, как отдельно подготовить код расширения и встроить его функциональные возможности в среду программирования Python. Вначале будет представлено обоснование причин, по которым это может потребоваться, а затем описан пошаговый процесс достижения указанной цели. При этом следует отметить, что модули расширения, как правило, подготавливаются на языке С, поэтому в качестве основы многих примеров, приведенных в этой главе, применяется исключительно код на языке С. По желанию можно также использовать язык С++, поскольку он представляет собой надмножество С, а при создании расширений на персональных компьютерах с помощью Microsoft Visual Studio должен применяться (Visual) С++.

## 8.1. Введение/обоснование

Во вступительном разделе показано, что такое расширения Python. В последующих разделах объясняется, по каким причинам следует заняться их созданием (или отказаться от этого).

### 8.1.1. Что такое расширение

Вообще говоря, как расширение можно рассматривать любой отдельно написанный код, который встраивается в другой сценарий Python или импортируется в него. Этот дополнительный код может быть разработан исключительно на языке Python или на компилируемом языке, таком как С и С++ (а также на Java для среды Jython и на C# или VisualBasic.NET для среды IronPython).

Одной из превосходных возможностей Python является то, что его расширения взаимодействуют с интерпретатором точно так же, как и обычные модули Python. Python спроектирован таким образом, что позволяет скрывать за абстракцией импорта модуля все детали основополагающей реализации от того кода, в котором используется соответствующее расширение. Без проведения специального поиска в файловой системе разработчик клиентской программы просто не имеет возможности определить, написан ли тот или иной модуль на языке Python или на компилируемом языке.



#### Создание расширений на различных платформах

Прежде всего следует отметить, что расширения, как правило, создаются в той среде разработки, которая применяется для компиляции самого интерпретатора Python. Расширение можно откомпилировать вручную самостоятельно или получить готовый

двоичный код, но в первом случае приходится учитывать некоторые нюансы. Несмотря на то, что самостоятельная компиляция может оказаться немного сложнее по сравнению с загрузкой и установкой готовых программ, она позволяет реализовать все возможности настройки используемой версии Python. Попытку разработать расширение следует осуществлять в среде, аналогичной той, в которой откомпилирован используемый интерпретатор Python.

Примеры, приведенные в этой главе, разработаны в системе на основе Unix (компилятор обычно входит в состав таких систем), но, имея доступ к компилятору C/C++ (или Java), вполне можно воспользоваться средой разработки Python на C/C++ (или Java), и при этом изменится лишь применяемый метод компиляции. Сам код, с помощью которого можно подготовить расширение, применимое в мире Python, остается одинаковым на любой платформе.

Разработчик, подготавливающий расширение для персонального компьютера на основе Windows, должен будет применять язык Visual C++ в среде Developer Studio. В дистрибутив Python входят файлы проекта для версии VC++ 7.1, но можно использовать более старую версию.

Дополнительные сведения по общим вопросам создания расширений приведены в следующих источниках.

- C++ на персональном компьютере — <http://docs.python.org/extending/windows>
- Java/Jython — <http://wiki.python.org/jython>
- IronPython — <http://ironpython.codeplex.com>

**Предостережение.** Обычно при переносе двоичного кода с одного базового компьютера на другой, имеющий такую же архитектуру, не возникают какие-либо сложности, но иногда небольшие различия в компиляторах или процессорах приводят к тому, что код и в том и в другом случае работает по-разному.

---

## 8.1.2. Причины, по которым может потребоваться создание расширения Python

Такая научная дисциплина, как программная инженерия (software engineering), еще достаточно молода, и до сих пор в ней рассматривались лишь языки программирования, изначально обладающие определенной совокупностью возможностей. Что видишь, то и получаешь; рядовому пользователю не предоставлялась возможность добавления новых функциональных средств непосредственно в существующий язык. Однако от современной среды программирования требуется способность обеспечивать ее гибкую настройку; это также служит стимулом для более широкого повторного использования кода. Возможность расширять базовый язык была впервые предоставлена в среде программирования, основанной на языке, подобном Tcl или Python. Такой язык, как Python, уже обладает широким набором средств, поэтому мы должны понять, для чего может потребоваться его дополнительное расширение. На это есть несколько весомых причин, которые описаны ниже.

- Дополнительные или вспомогательные функциональные возможности (не предусмотренные в Python). Одной из причин, по которой может потребоваться разработка расширений Python, является возникновение потребности в получении нового функционального средства, которое не предоставлено в основной части

языка. Для достижения этой цели можно разработать код исключительно на языке Python или создать компилируемое расширение, но встречаются и такие задачи, как создание новых типов данных или внедрение Python в существующее приложение, которые не могут быть решены без компиляции интерпретатора Python.

- Повышение производительности за счет устранения узкого места. Общеизвестно, что код на интерпретируемом языке не обладает таким же быстродействием, как на компилируемом языке, поскольку его компиляция должна происходить динамически, причем во время выполнения. Одним из методов повышения производительности программы на интерпретируемом языке является перемещение значительной части кода в расширение. При этом иногда приходится сталкиваться с таким недостатком, как увеличение объема расходов.
- Оценивая затраты и полученные результаты, иногда приходится обнаруживать, что более целесообразным является простое профилирование кода для выявления узких мест, и лишь после этого перемещение достаточно небольшой части кода в расширение. При этом выигрыш достигается быстрее и не приходится расходовать слишком много дополнительных ресурсов.
- Предотвращение несанкционированного доступа к собственному исходному коду. Еще одной важной причиной для создания расширения является устранение одного из побочных недостатков применения языка сценариев. Безусловно, очень важным преимуществом таких языков является простота использования, но при этом отсутствует возможность обеспечить конфиденциальность исходного кода, поскольку именно этот код в непосредственном виде передается на выполнение.
- Перемещение определенной части кода из сценария Python в модуль на компилируемом языке способствует предотвращению несанкционированного доступа к собственному коду, поскольку открытым для постороннего взгляда становится лишь двоичный объект. Такие объекты являются откомпилированными, поэтому не поддаются достаточно легко обратному проектированию; таким образом, исходный код становится более защищенным. Это очень важно, когда речь идет о специальных алгоритмах, шифровании, безопасности программного обеспечения и т.д.
- Еще одна возможность исключения несанкционированного доступа к исходному коду состоит в том, чтобы поставлять пользователю только предварительно скомпилированные файлы `.рус`. Это вполне приемлемое промежуточное решение, если приходится отказываться от передачи определенной части самого исходного кода (файлов `.ру`) и переносить ее в расширение.

### 8.1.3. Причины, по которым целесообразно отказаться от создания расширения Python

Прежде чем приступать к рассмотрению темы, касающейся создания расширений, имеет смысл указать, почему иногда не следует вообще этим заниматься. Читатель может рассматривать этот раздел как предостережение, чтобы неправильно истолкованное описание возможностей расширений не послужило для него недобросовестной рекламой. Безусловно, расширения предоставляют многие преимущества,

и выше были вкратце описаны лишь основные из них, но с их использованием связаны также определенные перечисленные ниже недостатки.

- Для написания кода расширения должен применяться язык C/C++.
- Приходится разбираться в том, как передаются данные между Python и C/C++.
- Управление ссылками приходится брать на себя.
- Предусмотрены такие инструменты, которые позволяют достичь той же цели. Иными словами, сформировать и применить высокопроизводительный код C/C++, но при этом вообще не требуется написания какого-либо кода C/C++. Краткие сведения о некоторых из этих инструментов приведены в конце настоящей главы.

Итак, еще раз предупреждаем о том, что при написании собственных расширений приходится сталкиваться с определенными сложностями! Теперь перейдем непосредственно к рассмотрению этой темы.

## 8.2. Модули расширения

Создание расширения для языка Python связано с выполнением трех основных этапов, описанных ниже.

1. Создание прикладного кода.
2. Оформление кода с применением стандартных шаблонов.
3. Компиляция и проверка.

В данном разделе будут кратко описаны и проиллюстрированы все три этапа.

### 8.2.1. Создание прикладного кода

В первую очередь, еще до того как какой-либо код перейдет в расширение, необходимо создать отдельную библиотеку, т.е. еще во время разработки кода следует учитывать, что он войдет в состав модуля Python. Функции и объекты должны проектироваться с учетом того, что код Python будет взаимодействовать с кодом C и обмениваться с ним данными, и наоборот.

После этого следует подготовить проверочный код, который позволил бы подтвердить, что создаваемое программное обеспечение способно справиться с любыми исключительными ситуациями. При создании проверочного приложения можно даже использовать принятый при разработке на языке Python способ с вызовом основной функции `main()` из командной строки в качестве проверочного приложения. При этом осуществляются компиляция и связывание кода, после чего формируется исполняемый модуль (а не создается только разделяемый объект). Вызов такого исполняемого модуля приводит к регрессионной проверке программной библиотеки. Именно этот подход применяется в следующем примере расширения.

В тестовом варианте будут реализованы две функции C, предназначенные для дальнейшего применения в сценариях Python. Первой из них является рекурсивная функция вычисления факториала, `fac()`. Вторая функция, `reverse()`, представляет собой реализацию простого алгоритма обращения строк, основное назначение которого состоит в обратной перестановке символов в строке на месте. Эта функция возвращает строку, все символы которой расположены в противоположном порядке по

отношению к исходной строке. При этом не предусматривается выделение отдельной промежуточной строки, в которую копировались бы символы исходной строки в обратном порядке. Для этого необходимо использовать указатели, поэтому код функции должен быть тщательно спроектирован и отлажен перед подключением этой функции к сценарию Python.

Первая версия функции, `Extest1.c`, представлена в примере 8.1.

**Пример 8.1. Версия библиотечной функции (`Extest1.c`), написанная исключительно на языке C**

В этом коде представлена функция из библиотеки функций C, которая предназначена для подключения к среде Python таким образом, чтобы ее можно было вызывать из интерпретатора Python. В данном случае основной проверочной функцией является `main()`.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int fac(int n)
6 {
7 if (n < 2) return(1); /* 0! = 1! = 1 */
8 return (n)*fac(n-1); /* n! = n*(n-1)! */
9 }
10
11 char *reverse(char *s)
12 {
13 register char t, /* временное хранение */
14 *p = s, : /* прямое преобразование */
15 *q = (s + (strlen(s)-1)); /* обратное преобразование */
16
17 while (p < q) /* если p < q */
18 { /* перестановка и перемещение указателей */
19 t = *p;
20 *p++ = *q;
21 *q-- = t;
22 }
23 return s;
24 }
25
26 int main()
27 {
28 char s[BUFSIZ];
29 printf("4! = %d\n", fac(4));
30 printf("8! = %d\n", fac(8));
31 printf("12! = %d\n", fac(12));
32 strcpy(s, "abcdef");
33 printf("reversing 'abcdef', we get '%s'\n", \
34 reverse(s));
35 strcpy(s, "madam");
36 printf("reversing 'madam', we get '%s'\n", \
37 reverse(s));
38 return 0;
39 }
```

Приведенный код содержит определения двух функций, `fac()` и `reverse()`, представляющих собой реализации функциональных возможностей, которые были только что описаны. Функция `fac()` принимает единственный целочисленный параметр, с помощью рекурсивного алгоритма вычисляет результат и в конечном итоге передает полученный результат в вызывающий код после выхода из самого внешнего вызова.

В последней части кода определена необходимая функция `main()`. Эта функция служит средством проверки, с помощью которого можно передавать различные параметры в функции `fac()` и `reverse()`. С помощью этой функции можно определить, правильно ли работает приведенный выше код.

Теперь перейдем к рассмотрению этапа компиляции кода. Для многих версий Unix с компилятором `gcc` можно использовать следующую команду:

```
$ gcc Extest1.c -o Extest
$
```

Для запуска готовой программы на выполнение достаточно вызвать следующую команду, после чего должен быть сформирован требуемый вывод:

```
$ Extest
4! = 24
8! = 40320
12! = 479001600
reversing 'abcdef', we get 'fedcba'
reversing 'madam', we get 'madam'
$
```

Следует еще раз подчеркнуть, что создавая программные модули на других языках, необходимо прилагать максимальные усилия по их отладке, чтобы не оказалось так, что к сценарию на языке Python подключена библиотека, содержащая потенциальные источники отказов. Иными словами, сначала полностью отлаживайте подключаемый код и только после этого приступайте к отладке приложения в целом. При этом, чем меньше промежуточных звеньев, применяемых для подключения стороннего кода к интерфейсам Python, тем скорее можно будет решить задачу интеграции кода и обеспечения его правильной работы.

Что же касается описанных выше функций, то каждая из них принимает один параметр и возвращает одно значение. Код этих функций предельно сокращен и тщательно проверен, поэтому не должны возникать проблемы при его подключении к интерпретатору Python. Важно отметить, что до сих пор ничто не говорит о том, что разрабатываемые функции предназначены для дальнейшего использования в сочетании с интерпретатором Python. Мы просто создаем стандартное приложение C (или C++).

## 8.2.2. Оформление кода с применением стандартных шаблонов

Вся реализация расширения основана главным образом на применении понятия оболочек, о котором уже шла речь в предыдущих главах. К ним относятся составные классы, функции декоратора, средства делегирования классов и т.д. Код должен быть спроектирован таким образом, чтобы взаимодействие между сценарием в среде Python и кодом на языке реализации происходило бесперебойно. Код, обеспечивающий создание интерфейса взаимодействия, обычно оформляется с применением

стандартных шаблонов, необходимых для обеспечения подключения внешнего кода к среде, создаваемой интерпретатором Python.

Стандартные программные шаблоны состоят из четырех основных частей.

1. Включение файла заголовка Python.
2. Добавление оболочки Python `PyObject* Module_func()` для каждой функции модуля.
3. Добавление массива/таблицы `PyMethodDef ModuleMethods[]` для каждой функции модуля.
4. Добавление функции инициализатора модуля `void initModule()`.

## Включение файла заголовка Python

Перед созданием расширения необходимо прежде всего найти включаемые файлы Python и обеспечить доступ компилятора к каталогу с этими файлами. В большинстве систем на основе Unix включаемые файлы размещаются в каталоге `/usr/local/include/python2.x` или `/usr/include/python2.x`, где `2.x` указывает применяемую версию Python. Программисты, которые сами откомпилировали и установили интерпретатор Python, не должны сталкиваться с какими-либо затруднениями, поскольку данные о том, где находятся необходимые файлы, обычно сохраняются в системе.

*Добавьте заголовок Python.h в исходный файл.* Строка, в которой выполняется эта операция, должна выглядеть примерно так:

```
#include "Python.h"
```

Это самая простая часть подготовки. После этого необходимо добавить остальную часть стандартных программных шаблонов.

## Добавление оболочек Python `PyObject* Module_func()` к каждой функции

Эта часть подготовки является наиболее сложной. Для каждой функции, к которой необходимо предусмотреть доступ из среды Python, должна быть создана статическая функция `PyObject*`, в которой требуется указать имя модуля с предшествующим знаком подчеркивания (`_`).

Например, предположим, что должна быть предусмотрена возможность импортировать из сценария Python одну из разработанных ранее функций, `fac()`, а в качестве имени заключительного модуля будет использоваться `Exttest`. Это означает, что должна быть создана оболочка с именем `Exttest_fac()`. В клиентском сценарии Python в определенном месте должен находиться вызов операции `import Exttest` и вызов функции `Exttest.fac()` (который может иметь более простую форму, `fac()`, если для импорта применялся оператор `from Exttest import fac`).

Оболочка осуществляет такие действия: принимает значения в формате Python, преобразовывает их в формат C, а затем вызывает соответствующую функцию с заданными параметрами. После завершения выполнения функции возвращаемое ею значение должно быть снова преобразовано в формат Python. Для осуществления необходимых при этом действий также применяется оболочка: с ее помощью происходит получение всех возвращаемых значений, преобразование их в формат Python, а затем возврат в вызывающий код с передачей всех необходимых значений.



Что касается рассматриваемой функции `fac()`, то после вызова клиентская программа вызывает `Exttest.fac()`, что равносильно подключению оболочки. Оболочка принимает целочисленное значение в формате Python, преобразует его в целочисленное значение в формате C, вызывает функцию `fac()` на языке C, а затем возвращает полученный целочисленный результат. После этого необходимо получить это возвращаемое значение, преобразовать его в целочисленное значение в формате Python, а затем вернуть результат вызова. (Следует учитывать, что в сценарии Python должен быть предусмотрен код, имитирующий объявление `def fac(n)`. Назначение этого кода состоит в том, чтобы обеспечить вызов в сценарии Python вместо реальной функции воображаемой функции `fac()`.)

Теперь рассмотрим, как происходит преобразование форматов. Для этого применяются функции `PyArg_Parse*`, которые обеспечивают преобразование из форматов Python в форматы C, и функция `Py_BuildValue()`, преобразующая из C в Python.

Функции `PyArg_Parse*` весьма напоминают функцию `sscanf()` языка C. Они принимают поток байтов, после чего в соответствии с определенной строкой формата интерпретируют данные и распределяют полученные значения по соответствующим контейнерным переменным, которые, как и следовало ожидать, адресуются с помощью указателей. В данном случае применяются две такие функции, которые возвращают 1 после успешной интерпретации, а при неудачном завершении возвращают 0.

Функция `Py_BuildValue()` действует по аналогии со `sprintf()`, принимая строку формата и преобразуя все параметры в один возвращаемый объект, содержащий заданные значения в затребованных форматах.

Краткие сведения об указанных функциях приведены в табл. 8.1.

**Таблица 8.1.** Преобразование данных в ходе взаимодействия между Python и C/C++

| Функция                                                                   | Описание                                                                                                                                   |
|---------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Передача значений из Python в C<br><code>int PyArg_ParseTuple()</code>    | Преобразовывает параметры (заданные в виде кортежа) при передаче из Python в C                                                             |
| <code>int PyArg_ParseTupleAndKeywords()</code>                            | То же, что и <code>PyArg_ParseTuple()</code> , но интерпретирует также параметры, заданные с помощью ключевых слов                         |
| Передача значений из C в Python<br><code>PyObject* Py_BuildValue()</code> | Преобразовывает значения данных C в возвращаемый объект Python, представляющий собой либо отдельный объект, либо отдельный кортеж объектов |

Для преобразования объектов данных при обеспечении взаимодействия C и Python применяется определенный набор кодов преобразования (табл. 8.2).

**Таблица 8.2.** Преобразования, связанные со взаимодействием Python<sup>a</sup> и C/C++. Единицы задания формата

| Единица задания формата | Тип Python                           | Тип C/C++                       |
|-------------------------|--------------------------------------|---------------------------------|
| <code>s, s#</code>      | <code>str/unicode, len()</code>      | <code>char* (, int)</code>      |
| <code>z, z#</code>      | <code>str/unicode/None, len()</code> | <code>char*/NULL (, int)</code> |
| <code>u, u#</code>      | <code>unicode, len()</code>          | <code>(Py_UNICODE*, int)</code> |
| <code>i</code>          | <code>int</code>                     | <code>int</code>                |
| <code>b</code>          | <code>int</code>                     | <code>char</code>               |

| Единица задания формата | Тип Python  | Тип C/C++          |
|-------------------------|-------------|--------------------|
| h                       | int         | short              |
| l                       | int         | long               |
| k                       | int or long | unsigned long      |
| I                       | int or long | unsigned int       |
| B                       | int         | unsigned char      |
| H                       | int         | unsigned short     |
| L                       | long        | long long          |
| K                       | long        | unsigned long long |
| c                       | str         | char               |
| d                       | float       | double             |
| f                       | float       | float              |
| D                       | complex     | Py_Complex*        |
| O                       | (any)       | PyObject*          |
| S                       | str         | PyStringObject     |
| N <sup>b</sup>          | (any)       | PyObject*          |
| O&                      | (any)       | (any)              |

<sup>a</sup> Эти коды формата предназначены для Python 2, но имеют близкие эквиваленты в Python 3.

<sup>b</sup> Аналогично "O", за исключением того, что количество ссылок на объект не увеличивается.

Именно эти коды преобразования задаются в соответствующих строках формата, которые определяют способ преобразования значений при передаче данных между программными объектами на разных языках. Следует учитывать, что для языка Java применяются другие типы преобразования, поскольку в этом языке все типы данных заданы как классы. Для ознакомления с тем, какие типы Java соответствуют объектам Python, обратитесь к документации Jython. То же касается языков C# и VB.NET.

Ниже приведена в окончательном виде функция оболочки `Exttest_fac()`.

```
static PyObject *
Exttest_fac(PyObject *self, PyObject *args) {

 int res; // интерпретация результата
 int num; // параметр для fac()
 PyObject* retval; // возвращаемое значение

 res = PyArg_ParseTuple(args, "i", &num);
 if (!res) { // TypeError
 return NULL;
 }
 res = fac(num);
 retval = (PyObject*)Py_BuildValue("i", res);
 return retval;
}
```

Первый шаг состоит в интерпретации данных, полученных из Python. Это должно быть обычное целочисленное значение, поэтому для указания на необходимое преобразование используется код преобразования `i`. Если в качестве данного значения

действительно передается целое число, то полученный результат сохраняется в переменной `num`. В противном случае функция `PyArg_ParseTuple()` возвращает значение `NULL`, и это значение поступает в программный код. В рассматриваемом примере при возникновении такой ситуации формируется исключение `TypeError`, которое служит для пользователя клиентской программы указанием на то, что должно быть задано целое число.

После этого вызывается функция `fac()` с параметром, заданным в переменной `num`, а полученное значение присваивается переменной `res`, прежнее значение которой теряется. Затем должен быть сформирован возвращаемый объект, целочисленное значение в формате Python, для чего снова используется код преобразования `i`. Возвращаемый целочисленный объект Python создается с помощью функции `Py_BuildValue()`. На этом все необходимые действия оканчиваются!

В действительности программист, создавая одну оболочку за другой, постепенно может найти способы некоторого сокращения кода за счет более рационального использования переменных. Тем не менее следует стремиться к тому, чтобы код оставался удобным для восприятия. В следующем примере показано, как взять за основу функцию `Extest_fac()` и сократить ее до более краткой версии, перейдя к использованию только одной переменной, `num`:

```
static PyObject *
Extest_fac(PyObject *self, PyObject *args) {
 int num;
 if (!PyArg_ParseTuple(args, "i", &num))
 return NULL;
 return (PyObject*)Py_BuildValue("i", fac(num));
}
```

Рассмотрим разработку функции `reverse()`. Выше было показано, как вернуть из функции единственное значение, а в этом разделе описано, как внести изменения в функцию `reverse()`, чтобы обеспечить возврат двух значений вместо одного. Возвращаемым значением будет служить кортеж, состоящий из двух строк; первым элементом кортежа должна быть строка, переданная в функцию, а вторым — вновь полученная строка с символами, расположенными в обратном порядке.

Для того чтобы было проще различить оба варианта функции обращения строк, назовем последний вариант `Extest.doppel()`. Это имя указывает, в чем второй вариант (с дублированием) отличается от первого варианта, функции `reverse()`. Включая код в оболочку в виде функции `Extest_doppel()`, можно получить следующее:

```
static PyObject *
Extest_doppel(PyObject *self, PyObject *args) {
 char *orig_str;
 if (!PyArg_ParseTuple(args, "s", &orig_str)) return NULL;
 return (PyObject*)Py_BuildValue("ss", orig_str, \
 reverse(strdup(orig_str)));
}
```

Как и в функции `Extest_fac()`, берется единственное входное значение, на этот раз — строка, и сохраняется в переменной `orig_str`. Заслуживает внимания то, что в данном случае используется код преобразования `s`. Затем для создания копии строки вызывается функция `strdup()`. (В нашу задачу входит возврат не только преобразованной, но и исходной строки, поэтому необходимо подготовить отдельную строку, в которой должно быть выполнено обращение символов, а для этой цели лучше всего

подходит операция создания копии исходной строки.) Функция `strdup()` создает и возвращает копию, которая после этого сразу же передается в функцию `reverse()`. Возвращаемым значением этой функции становится строка, в которой символы расположены в обратном порядке по отношению к исходной строке.

Как показывает рассматриваемый код, в функции `Py_BuildValue()` обе выходные строки соединяются с помощью строки преобразования `ss`. В результате создается кортеж из двух строк: исходной строки и строки с обратным порядком символов. На первый взгляд может показаться, что на этом наша работа заканчивается. Но, к сожалению, дело обстоит иначе.

В этом фрагменте кода приходится сталкиваться с одним из побочных последствий неправильной организации программы C, с утечкой ресурсов памяти (так называется ситуация, при которой память распределяется, но не освобождается). Утечка ресурсов памяти может привести к исчерпанию свободной памяти, по аналогии с тем, что полки библиотеки могут опустеть, если читатели будут только брать книги и не возвращать. В программе всегда следует освобождать захваченные ресурсы после того, как необходимость в их использовании отпадает. Рассмотрим, как исправить этот недостаток кода (притом что код на первый взгляд кажется вполне правильным).

При формировании с помощью функции `Py_BuildValue()` возвращаемого объекта Python создается копия данных, переданных в эту функцию. В рассматриваемом случае в качестве такой копии выступают две строки. Проблема состоит в том, что в функции распределяется память для второй строки, но после завершения работы эта память не освобождается, что приводит к утечке ресурсов. Поэтому для устранения указанного недостатка необходимо сформировать возвращаемый объект, а затем освободить память, распределенную в функции, выполняющей роль оболочки. Мы не можем поступить иначе, кроме как ввести в код несколько дополнительных строк:

```
static PyObject *
Extest_doppel(PyObject *self, PyObject *args) {
 char *orig_str; : : // исходная строка
 char *dupe_str; : : // обращенная строка
 PyObject* retval;
 if (!PyArg_ParseTuple(args, "s", &orig_str)) return NULL;
 retval = (PyObject*)Py_BuildValue("ss", orig_str, \
 dupe_str=reverse(strdup(orig_str)));
 free(dupe_str);
 return retval;
}
```

В данном случае была введена дополнительная переменная `dupe_str`, с помощью которой формируется указатель на строку, расположенную в распределяемой памяти, и создается возвращаемый объект. После этого выделенная память освобождается с помощью функции `free()` и происходит возврат в вызывающий код. И эта задача нами решена.

## Добавление массива/таблицы `PyMethodDef` `ModuleMethods[]` для каждой функции модуля

Итак, обе необходимые оболочки сформированы, осталось лишь дать указание интерпретатору Python в отношении того, как импортировать эти оболочки и получить к ним доступ. Для этого применяется массив `ModuleMethods[]`.

Этот массив сам состоит из массивов, среди которых каждый отдельный массив содержит информацию об отдельной функции, а за ними следует массив `NULL`, который отмечает конец списка. Для модуля `Extest` создается следующий массив `ExtestMethods[]`:

```
static PyMethodDef
ExtestMethods[] = {
 { "fac", Extest_fac, METH_VARARGS },
 { "doppel", Extest_doppel, METH_VARARGS },
 { NULL, NULL },
};
```

Здесь заданы имена, доступные из интерпретатора Python, а за ними следуют соответствующие функции оболочек. Кроме того, задана константа `METH_VARARGS`, указывающая на набор параметров в форме кортежа. Если бы использовалась функция `PyArg_ParseTuple AndKeywords()` с параметрами в виде ключевых слов, то пришлось бы соединить этот флаг с константой `METH_KEYWORDS` с помощью логической операции `OR`. Наконец, список из двух функций завершается должным образом с помощью пары значений `NULL`.

## Добавление функции инициализатора модуля `void initModule()`

Заключительным этапом выполнения нашего сложного задания становится создание функции инициализатора модуля. Код этой функции вызывается после импорта модуля для использования интерпретатором. В данном коде выполняется один вызов функции `Py_InitModule()` с указанием имени модуля и массива `ModuleMethods[]`. С помощью этих данных интерпретатор может получить доступ к функциям модуля. Что касается рассматриваемого модуля `Extest`, то процедура `initExtest()` выглядит следующим образом:

```
void initExtest() {
 Py_InitModule("Extest", ExtestMethods);
}
```

На этом рассмотрение тематики применения оболочек исчерпывается. Добавим весь этот код к исходному коду из файла `Extest1.c` и представим полученные результаты в виде нового файла `Extest2.c`. На этом этап разработки текущего примера завершается.

Еще один подход к созданию расширения может состоять в том, чтобы вначале подготавливался код оболочек с использованием заглушек, тестовых или фиктивных функций, которые в ходе разработки заменялись бы полнофункциональными фрагментами реализованного кода. Это позволяет гарантировать то, что интерфейс между Python и C с самого начала будет определен правильно, а впоследствии использовать Python для проверки кода C.

## 8.2.3. Компиляция

2.0

Перейдем к этапу компиляции. Для того чтобы обеспечить построение нового расширения Python на основе оболочки, необходимо создать предпосылки успешной компиляции кода с применением библиотеки Python. Эта задача была стандартизирована (начиная с версии 2.0) для всех платформ, что позволяет намного упростить работу по созданию расширений. Для

построения, установки и распространения модулей, расширений и пакетов применяется пакет `distutils`. Этот пакет впервые появился в версии Python 2.0 и заменил применявшийся в версиях 1.x устаревший способ построения расширений, в котором использовались `make`-файлы. Работа с пакетом `distutils` осуществляется на основе следующего простого принципа.

1. Создание сценария `setup.py`.
2. Компиляция и связывание кода путем выполнения сценария `setup.py`.
3. Импорт модуля из среды Python.
4. Проверка функции.

## Создание `setup.py`

На следующем этапе должен быть создан файл `setup.py`. Основной объем работы выполняется с помощью функции `setup()`. Все строки кода, предшествующие этому вызову, служат для выполнения подготовительных шагов. Для построения модулей расширения должны быть созданы отдельные экземпляры `Extension` для каждого расширения. В данном случае требуется создание одного расширения, поэтому нужен лишь один экземпляр `Extension`:

```
Extension('Exttest', sources=['Exttest2.c'])
```

Первым параметром является (полное) имя расширения, включающее в случае необходимости все высокоуровневые пакеты. Имя должно быть задано в точечной системе обозначения с указанием всех атрибутов. Рассматриваемое нами расширение является автономным, поэтому для него задается имя `Exttest`. Параметр `sources` представляет собой список всех исходных файлов. Опять-таки речь идет о единственном файле `Exttest2.c`.

Теперь можно приступить к вызову `setup()`. Эта функция принимает параметры в виде имени создаваемого расширения и списка элементов, применяемых для построения. В данном случае создается расширение, поэтому должен быть задан список модулей расширения, которые должны быть построены, с помощью параметра `ext_modules`. Применяемый синтаксис выглядит так:

```
setup('Exttest', ext_modules=[...])
```

В данном случае имеется лишь один модуль, поэтому порождение экземпляра модуля расширения сочетается с вызовом функции `setup()`, для чего имя модуля задается в качестве константы `MOD` в предыдущей строке:

```
MOD = 'Exttest'
setup(name=MOD, ext_modules=[
 Extension(MOD, sources=['Exttest2.c'])])
```

Функция `setup()` имеет также много других параметров. В действительности число этих параметров слишком велико, и у нас нет возможности рассмотреть их в данной книге. Дополнительные сведения о создании сценария `setup.py` и вызове функции `setup()` можно найти в официальной документации Python, которая указана в конце этой главы. В примере 8.2 приведен полный сценарий, который используется в качестве рассматриваемого примера.

### Пример 8.2. Сценарий формирования (setup.py)

В этом сценарии осуществляется компиляция расширения и запись в подкаталог build/lib.\*.

```

1 #!/usr/bin/env python
2
3 from distutils.core import setup, Extension
4
5 MOD = 'Exttest'
6 setup(name=MOD, ext_modules=[
7 Extension(MOD, sources=['Exttest2.c'])])

```

### Компиляция и связывание кода путем выполнения сценария setup.py

Теперь, после подготовки файла setup.py, можно приступить к построению расширения путем вызова сценария на выполнение с директивой build. Ниже приведены результаты, полученные на компьютере Mac (сформированный вывод может измениться при использовании другой разновидности операционной системы или другой версии Python).

```

$ python setup.py build
running build
running build_ext
building 'Exttest' extension
creating build
creating build/temp.macosx-10.x-fat-2.x
gcc -fno-strict-aliasing -Wno-long-double -no-cpp-
precomp -mno-fused-madd -fno-common -dynamic -DNDEBUG -g
-I/usr/include -I/usr/local/include -I/sw/include -I/
usr/local/include/python2.x -c Exttest2.c -o build/temp.macosx-10.x-fat-2.x/Exttest2.o
creating build/lib.macosx-10.x-fat-2.x
gcc -g -bundle -undefined dynamic_lookup -L/usr/lib -L/
usr/local/lib -L/sw/lib -I/usr/include -I/usr/local/
include -I/sw/include build/temp.macosx-10.x-fat-2.x/Exttest2.o -o build/lib.macosx-
10.x-fat-2.x/Exttest.so

```

## 8.2.4. Импорт и проверка

На заключительном этапе необходимо снова перейти к работе в интерпретаторе Python и попытаться воспользоваться новым расширением так, как если бы оно было написано полностью на языке Python.

### Импорт нового модуля из сценария Python

Новый модуль расширения должен быть создан в каталоге build/lib.\*, т.е. в том каталоге, из которого запускается сценарий setup.py. Для проверки модуля необходимо либо перейти в этот каталог, либо включить модуль в состав дистрибутива Python следующим образом (эта операция называется установкой):

```
$ python setup.py install
```

Инсталляция модуля приводит к получению таких результатов:

```
running install
running build
running build_ext
running install_lib
copying build/lib.macosx-10.x-fat-2.x/Exttest.so ->
/usr/local/lib/python2.x/site-packages
```

После этого можно приступить к проверке модуля с помощью интерпретатора:

```
>>> import Exttest
>>> Exttest.fac(5)
120
>>> Exttest.fac(9)
362880
>>> Exttest.doppel('abcdefgh')
('abcdefgh', 'hgfedcba')
>>> Exttest.doppel("Madam, I'm Adam.")
("Madam, I'm Adam.", ".madA m'I ,madaM")
```

## Добавление функции проверки

Последний необходимый подготовительный этап состоит в добавлении функции проверки. В действительности для размещения такой функции уже есть подходящее место — в форме функции `main()`. Следует учитывать, что включение функции `main()` в код может служить источником потенциальной опасности нарушения последовательности вызовов, поскольку в системе должна присутствовать только одна функция `main()`. Чтобы исключить такую опасность, достаточно изменить имя применяемой функции `main()`, допустим, на `test()`, предусмотреть для нее оболочку, добавив определение `Exttest_test()`, и обновить массив `ExttestMethods`, чтобы заданные в нем методы выглядели следующим образом:

```
static PyObject *
Exttest_test(PyObject *self, PyObject *args) {
 test();
 return (PyObject*)Py_BuildValue("");
}

static PyMethodDef
ExttestMethods[] = {
 { "fac", Exttest_fac, METH_VARARGS },
 { "doppel", Exttest_doppel, METH_VARARGS },
 { "test", Exttest_test, METH_VARARGS },
 { NULL, NULL },
};
```

Назначение функции модуля `Exttest_test()` сводится к тому, что выполняется функция `test()` и возвращается пустая строка, в результате чего вызывающий код возвращается в значение `None`, предусмотренное в языке Python.

Теперь та же проверка может быть выполнена из сценария Python:

```
>>> Exttest.test()
4! = 24
8! = 40320
12! = 479001600
```



```

reversing 'abcdef', we get 'fedcba'
reversing 'madam', we get 'madam'
>>>

```

В примере 8.3 представлена заключительная версия программы Exttest2.c, которая использовалась для формирования приведенного выше вывода.

**Пример 8.3. Программа, предназначенная для включения в библиотеку C (Exttest2.c), в которой предусмотрена оболочка для сценария Python**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int fac(int n)
6 {
7 if (n < 2) return(1);
8 return (n)*fac(n-1);
9 }
10
11 char *reverse(char *s)
12 {
13 register char t,
14 *p = s,
15 *q = (s + (strlen(s) - 1));
16
17 while (s && (p < q))
18 {
19 t = *p;
20 *p++ = *q;
21 *q-- = t;
22 }
23 return s;
24 }
25
26 int test()
27 {
28 char s[BUFSIZ];
29 printf("4! = %d\n", fac(4));
30 printf("8! = %d\n", fac(8));
31 printf("12! = %d\n", fac(12));
32 strcpy(s, "abcdef");
33 printf("reversing 'abcdef', we get '%s'\n", \
34 reverse(s));
35 strcpy(s, "madam");
36 printf("reversing 'madam', we get '%s'\n", \
37 reverse(s));
38 return 0;
39 }
40
41 #include "Python.h"
42
43 static PyObject *
44 Exttest_fac(PyObject *self, PyObject *args)
45 {
46 int num;

```

```

47 if (!PyArg_ParseTuple(args, "i", &num))
48 return NULL;
49 return (PyObject*)Py_BuildValue("i", fac(num));
50 }
51
52 static PyObject *
53 Extest_doppel(PyObject *self, PyObject *args)
54 {
55 char *orig_str;
56 char *dupe_str;
57 PyObject* retval;
58
59 if (!PyArg_ParseTuple(args, "s", &orig_str))
60 return NULL;
61 retval = (PyObject*)Py_BuildValue("ss", orig_str, \
62 dupe_str=reverse(strdup(orig_str)));
63 free(dupe_str);
64 return retval;
65 }
66
67 static PyObject *
68 Extest_test(PyObject *self, PyObject *args)
69 {
70 test();
71 return (PyObject*)Py_BuildValue("");
72 }
73
74 static PyMethodDef
75 ExtestMethods[] =
76 {
77 { "fac", Extest_fac, METH_VARARGS },
78 { "doppel", Extest_doppel, METH_VARARGS },
79 { "test", Extest_test, METH_VARARGS },
80 { NULL, NULL },
81 };
82
83 void initExtest()
84 {
85 Py_InitModule("Extest", ExtestMethods);
86 }

```

При подготовке этого примера было решено разделить код исключительно на языке С от кода, предназначенного для создания интерфейса со сценарием Python. При такой организации программа становится более легкой для восприятия, к тому же не возникают такие проблемы, как при использовании предыдущего, более краткого примера. На практике такие файлы исходного кода, как правило, имеют больший объем. Кроме того, некоторые разработчики предпочитают использовать вариант, при котором для реализации оболочек применяются отдельные файлы исходного кода, с такими именами, как `ExtestWrappers.c`, или нечто в этом роде.

## 8.2.5. Подсчет ссылок

Напоминаем, что в среде Python используется механизм сбора мусора, в котором постоянно осуществляется подсчет ссылок для слежения за объектами и удаления объектов, которые больше не упоминаются ни в одной ссылке. При создании

расширений необходимо уделять еще больше внимания тому, как происходят манипуляции с объектами Python. В частности, следует учитывать, как и где происходит подсчет ссылок на такие объекты, и должен ли этот подсчет выполняться непосредственно в программе.

Ссылки на объекты подразделяются на два типа: собственные и заимствованные. Количество *собственных ссылок* (owned references) на объект отслеживает владелец объекта, т.е. программный модуль. Собственные ссылки, в частности, формируются при создании объекта Python в программном модуле с нуля.

После завершения работы с объектом Python в программном модуле необходимо прекратить действие прав владения объектом. Для этого можно уменьшить количество ссылок, передать право владения вместе с самим объектом или сохранить объект. Невыполнение требования по уничтожению собственной ссылки приводит к утечке ресурсов памяти.

В программном модуле могут также применяться *заимствованные ссылки* (borrowed references) на объекты. Операции с заимствованными ссылками не связаны с такой же степенью ответственности, как с собственными ссылками, поскольку они обычно сводятся к передаче ссылки на объект, но не требуют выполнения каких-либо манипуляций с данными каким-либо образом. Кроме того, в программном модуле нет необходимости обеспечивать подсчет заимствованных ссылок, при условии, что в нем не происходит захват самой ссылки после уменьшения количества ссылок до нуля. Для преобразования заимствованной ссылки в собственную ссылку достаточно увеличить количество ссылок на объект, на который указывает заимствованная ссылка.

В языке Python предусмотрено несколько макросов C, которые используются для изменения количества ссылок на объекты Python. Эти макросы приведены в табл. 8.3.

**Таблица 8.3.** Макросы для реализации подсчета ссылок на объекты Python

| Функция                     | Описание                                          |
|-----------------------------|---------------------------------------------------|
| <code>Py_INCREF(obj)</code> | Увеличение количества ссылок на объект <i>obj</i> |
| <code>Py_DECREF(obj)</code> | Уменьшение количества ссылок на объект <i>obj</i> |

В приведенном выше примере функции `Exttest_test()` происходит возврат значения `None` в результате создания объекта `PyObject` с пустой строкой. Но такой же цели можно достичь иначе: стать владельцем объекта `None`, `PyNone`, увеличить количество ссылок на него и явно вернуть полученное значение, как показано в следующем альтернативном фрагменте кода:

```
static PyObject *
Exttest_test(PyObject *self, PyObject *args) {
 test();
 Py_INCREF(Py_None);
 return PyNone;
}
```

`Py_INCREF()` и `Py_DECREF()` также имеют версии, в которых происходит проверка на наличие объектов `NULL`, — `Py_XINCREF()` и `Py_XDECREF()` соответственно.

Настоятельно рекомендуем обращаться к документации Python за дополнительными сведениями о расширении и внедрении Python, если возникают какие-либо вопросы, касающиеся подсчета ссылок (см. ссылки на документацию в приложении В).

## 8.2.6. Многопоточная организация и GIL

При разработке расширения необходимо учитывать, что разработанный код может, в частности, выполняться в многопоточной среде Python. В разделе 4.3.1 главы 4 приведены сведения о виртуальной машине Python (Python Virtual Machine — PVM) и глобальной блокировке интерпретатора (Global Interpreter Lock — GIL). В этом разделе было показано, что в машине PVM в любое время может функционировать только один поток выполнения, а применение блокировки GIL способствует предотвращению одновременного выполнения других потоков. Кроме того, было показано, что в коде, вызывающем внешние функции, например, из модуля расширения, блокировка GIL должна устанавливаться до момента возврата после вызова.

При этом выполнение программы становится однопоточным. Если же должна быть применена многопоточная организация, то, как было указано, разработчику расширения приходится использовать определенные способы освобождения GIL, например, перед выполнением системного вызова. Такая задача решается путем обозначения в коде участков, в которых несколько потоков могут (или не могут) выполняться безопасно, с помощью еще одной пары макросов `C`, `Py_BEGIN_ALLOW_THREADS` и `Py_END_ALLOW_THREADS`. В блоке кода, обозначенном с помощью этих макросов, разрешается выполнение не только текущего потока, но и других потоков.

Рекомендуем при использовании такой организации программы, как и при работе с макросами для подсчета ссылок, обращаться за дополнительными сведениями к документации по расширению и внедрению Python, а также к справочному руководству по API Python/C.

## 8.3. Другие темы

В этом заключительном разделе главы рассмотрим некоторые инструменты, которыми можно воспользоваться вместо написания расширений (на любом поддерживаемом языке). Здесь представлены краткие сведения о пакетах SWIG, Pyrex, Cython, rpyc и PyPy. В конце данной главы будет кратко описана связанная с ней тема, касающаяся внедрения Python.

### 8.3.1. Упрощенный генератор оболочек и интерфейса

Одно из имеющихся внешних инструментальных средств известно под названием SWIG (Simplified Wrapper and Interface Generator — упрощенный генератор оболочек и интерфейса). Этот пакет разработан Дэвидом Бизли (David Beazley), который является также автором книги *Python Essential Reference* (Addison-Wesley, 2009). Это программный инструмент, который позволяет с помощью специально аннотированных файлов заголовков C/C++ сформировать код оболочки, готовый для компиляции и дальнейшего применения программами Python, Tcl и Perl. Разработчик, применяющий SWIG, освобождается от необходимости писать стандартный код шаблонов, подобный тому, который рассматривался в данной главе. Ему остается лишь подготовить ту часть кода, которая касается реализации решения проекта на языке C/C++. После создания файлов в формате SWIG осуществляется обработка этих файлов, и вся необходимая подготовка проводится без участия разработчика. Более подробные сведения об инструменте SWIG можно найти на посвященном ему веб-сайте по адресу

<http://swig.org>

<http://en.wikipedia.org/wiki/SWIG>

## 8.3.2. Язык Pyrex

Одна из очевидных сложностей, связанных с созданием расширений C/C++ (с использованием стандартных шаблонов или с помощью SWIG), состоит в том, что разработчику приходится писать код на языке C/C++ (это — очевидное требование). Для этого он должен освоить все возможности указанного языка, а также, что еще более важно, научиться избегать его ловушек. Язык Pyrex позволяет получить все преимущества, связанные с созданием расширений, и при этом избежать указанных затруднений. Pyrex — это новый язык, созданный специально для написания расширений Python. Он представляет собой своеобразное сочетание C и Python, в котором наиболее значительная часть взята из языка Python; в действительности создатели Pyrex заходят в своих утверждениях настолько далеко, что на веб-сайте, посвященном этому языку, приводят высказывание, что Pyrex — это Python с типами данных C. При подготовке расширения достаточно лишь написать исходный код с применением синтаксиса Pyrex и обработать этот код с помощью компилятора Pyrex. Компилятор Pyrex создает файлы C, которые затем можно откомпилировать и использовать как при создании обычного расширения. Мы знаем таких программистов, которые дали зарок никогда больше не писать программы непосредственно на языке C, после того как обнаружили возможности Pyrex. Для того чтобы получить дополнительные сведения о пакете Pyrex, перейдите на его начальную страницу:

<http://cosc.canterbury.ac.nz/~greg/python/Pyrex>  
[http://en.wikipedia.org/wiki/Pyrex\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Pyrex_(programming_language))

## 8.3.3. Язык Cython

Cython — это клон Pyrex, создание которого началось в 2007 году. Первый выпуск Cython, получивший обозначение 0.9.6, появился примерно в то же время, что и Pyrex 0.9.6. Разработчики Cython руководствуются более динамичным и агрессивным подходом к созданию Cython, чем представители команды Pyrex, поскольку выпуск новых версий Pyrex начинается после гораздо более тщательной проверки. В результате исправления, усовершенствования и дополнения к Cython появляются быстрее и чаще по сравнению с Pyrex, но оба эти проекта продолжают активно развиваться. Дополнительные сведения о Cython и о том, в чем этот пакет отличается от Pyrex, можно найти на указанных ниже сайтах.

<http://cython.org>  
<http://wiki.cython.org/DifferencesFromPyrex>  
<http://wiki.cython.org/FAQ>

## 8.3.4. Язык Pysco

Преимуществом Pyrex и Cython является то, что эти языки позволяют избавиться от необходимости писать код исключительно на языке C. Для освоения работы с этими языками приходится изучать новый синтаксис (иными словами, овладевать еще одним языком программирования). К тому же в конечном итоге созданный код Pyrex/Cython преобразуется в код C. При этом в действительности разработчики создают расширения или используют такие инструменты, как SWIG или Pyrex/Cython, в целях повышения производительности. Однако иногда высокой производительности программы можно добиться, не выходя за пределы языка Python.

Исходя из этих соображений, и был создан язык Psycο. Разработчики Psycο поставили перед собой задачу добиться ускорения работы всего существующего кода Python вместо замены части этого кода кодом C. В языке Psycο применяется своего рода динамический (just-in-time — JIT) компилятор, поэтому нет необходимости вносить в исходный код Python какие-либо существенные изменения, кроме импорта модуля Psycο и передачи ему указания приступить к оптимизации кода (во время выполнения).

### 2.2-2.6

Язык Psycο предоставляет также возможность профилировать код для определения того, где может быть достигнуто наиболее существенное повышение производительности. Предусмотрена даже возможность разрешить ведение журналов для контроля над тем, какие действия выполняет Psycο при оптимизации кода. Единственным ограничением этого пакета является то, что он поддерживает исключительно 32-разрядные архитектуры Intel 386 (в операционных системах Linux, Mac OS X, Windows и BSD), на которых работают версии Python 2.2.2-2.6.x, но не версии 3.x. Ко времени написания данной книги поддержка версии 2.7 была неполной. Чтобы получить дополнительные сведения, перейдите на следующие сайты:

<http://psyco.sf.net>

<http://en.wikipedia.org/wiki/Psyco>

## 8.3.5. PyPy

PyPy — это проект, ставший преемником Psycο. Разработчики этого проекта поставили перед собой намного более амбициозную цель — создать среду общего назначения для разработки интерпретируемых языков, независимо от платформы или целевой среды выполнения. Во время запуска проекта PyPy была поставлена намного более скромная задача — создать интерпретатор Python, написанный на языке Python. В действительности многие до сих пор считают, что именно в этом состоит назначение проекта PyPy, тогда как указанная задача уже давно решена, и данный конкретный интерпретатор стал лишь частью всей экосистемы PyPy.

Иными словами, инструментарий PyPy открывает широкие перспективы, поскольку позволяет проектировщикам языков сосредоточиться исключительно на вопросах синтаксического и семантического анализа интерпретируемого языка, над которым они в данное время работают. Этот инструментарий берет на себя решение всех сложных проблем трансляции в код, предназначенный для выполнения в собственной архитектуре, таких как управление памятью, преобразование в байт-код, сбор мусора, внутреннее представление числовых типов, формирование примитивных структур данных, поддержка собственной архитектуры и т.д.

Работа с инструментарием PyPy складывается таким образом: проектировщик разрабатывает язык и реализует его с помощью ограниченной версии Python со статической типизацией, получившей название RPython. Как было указано выше, первой из поставленных задач было создание интерпретатора Python на языке Python, и определение Python было впервые написано на языке RPython. По-видимому, это и послужило причиной для выбора такого имени для пакета PyPy. Однако с помощью RPython можно реализовать любой язык, а не только Python.

Работа с инструментарием PyPy складывается как последовательность применения цепочки инструментов, в результате чего код RPython транслируется в программный

объект низкого уровня, такой как модуль C, байт-код Java или код CIL (Common Intermediate Language). Последний представляет собой байт-код для языков, при написании которых применяется стандарт CLI (Common Language Infrastructure). Иными словами, разработчикам интерпретируемых языков остается заниматься лишь проектированием самого языка и гораздо меньше задумываться о реализации и целевой архитектуре. Дополнительные сведения можно получить на следующих сайтах:

<http://pypy.org>  
<http://codespeak.net/pypy>  
<http://en.wikipedia.org/wiki/PyPy>

### 8.3.6. Внедрение

Язык Python поддерживает еще одну возможность применения в сочетании с другими программами — внедрение. Подход, предусматривающий внедрение интерпретатора Python, является обратным по отношению к его расширению. Расширение предусматривает создание оболочки для кода C на языке Python, а внедрение сводится к тому, что в приложении C создается оболочка для интерпретатора Python. Это позволяет создавать крупные и цельные приложения, которые вместе с тем являются надежными, специализированными, способными решать ответственные задачи и обладающие тем преимуществом, что в их состав входит внедренный интерпретатор Python. Дело в том, что наличие интерпретатора Python в приложении неизмеримо расширяет его возможности.

Для разработчиков расширений предусмотрен целый ряд официальных документов, к которым они могут обращаться за дополнительными сведениями.

Ниже приведены ссылки на некоторую документацию Python, относящуюся к тематике данной главы (<http://docs.python.org/extending/embedding>).

#### *Расширение и внедрение*

<http://docs.python.org/ext>

#### *API Python/C*

<http://docs.python.org/c-api>

#### *Распространение модулей Python*

<http://docs.python.org/distutils>

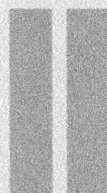
## 8.4. Упражнения

- 8.1. *Расширение Python.* В чем состоят некоторые из преимуществ расширений Python?
- 8.2. *Расширение Python.* Укажите, какие недостатки или потенциальные опасности связаны, на ваш взгляд, с использованием расширений.

- 8.3. *Написание расширений.* Установите на своем компьютере компилятор C/C++ и ознакомьтесь с программированием C/C++ (или освежите свои знания). Создайте простую вспомогательную функцию, к которой вы можете получить доступ и настроить ее в качестве расширения. Продемонстрируйте работу вашей программы и в C/C++, и в Python.
- 8.4. *Перенос из Python в C.* Возьмите за основу несколько упражнений, выполненных вами в предыдущих главах, и перенесите полученные результаты в C/C++ в качестве модулей расширения.
- 8.5. *Создание оболочки для кода C.* Найдите фрагмент кода C/C++, который вы, возможно, разработали уже давно, но хотели бы перенести в Python. Но вместо того, чтобы переносить этот код, преобразуйте его в модуль расширения.
- 8.6. *Написание расширений.* В одном из упражнений в главе по объектно-ориентированному программированию в книге *Core Python Programming* или *Core Python Language Fundamentals* было предложено создать функцию `dollarize()` в составе класса, которая должна была применяться для форматирования значений с плавающей точкой с преобразованием в числовую строку, предназначенную для использования в финансовых приложениях. Создайте расширение, которое включает оболочку для функции `dollarize()`, и введите в этот модуль функцию регрессионного тестирования, например, с именем `test()`. **Дополнительное задание.** В дополнение к созданию расширения C преобразуйте код функции `dollarize()` для обработки с помощью Pyrex или Cython.
- 8.7. *Расширение и внедрение.* В чем состоят различия между расширением и внедрением?
- 8.8. *Отказ от написания расширений.* Возьмите за основу код C/C++, который использовался в упражнении 8.3, 8.4 или 8.5, и преобразуйте его в код, имитирующий Python, с помощью Pyrex или Cython. Опишите полученный вами опыт использования Pyrex/Cython в сравнении с теми навыками, которые требуются для включения того же кода в состав расширения C.



ЧАСТЬ



# Разработка веб-приложений

# Веб-клиенты и веб-серверы

*В этой главе...*

- Введение
- Инструменты веб-клиентов Python
- Веб-клиенты
- Веб-серверы (HTTP)
- Связанные модули

*Если в вашем распоряжении имеется браузер из проекта CERN для WWW (World Wide Web — распределенная гипертекстовая система), вы можете просматривать гипертекстовую версию данного руководства в WWW.*

Гвидо ван Россум (Guido van Rossum), ноябрь 1992 г.  
(первое упоминание о веб в списке рассылки Python).

## 9.1. Введение

Разнообразие веб-приложений чрезвычайно велико, поэтому структура данной книги была организована так, чтобы читатель мог сосредоточиться отдельно на каждой из многочисленных тем, связанных с разработкой веб-приложений.

Глава представляет собой введение в программирование для веб с точки зрения архитектуры “клиент–сервер”, но на этот раз в рамках веб. В ней представлены важные сведения, необходимые для усвоения материала оставшихся глав настоящей книги.

### 9.1.1. Навигация по веб-страницам с помощью средств архитектуры клиент–сервер

В навигации по веб-страницам применяется уже известная читателям архитектура “клиент–сервер”. Однако в этом случае *веб-клиентами* являются браузеры, иными словами, приложения, позволяющие пользователям просматривать документы в World Wide Web. Браузеры взаимодействуют с *веб-серверами*, т.е. с процессами, выполняющимися на хост-компьютерах поставщиков информации. Серверы ожидают поступления запросов от клиентов на получение информации, обрабатывают их, а затем возвращают затребованные данные. Как и большинство серверов в системе “клиент–сервер”, веб-серверы предназначены для работы в течение неопределенно долгого времени. Навигация по веб-страницам наглядно продемонстрирована на рис. 9.1. Пользователь вызывает программу веб-клиента, такую как браузер, и устанавливает с ее помощью соединение с веб-сервером, находящимся в другом месте в Интернете, для получения необходимой информации.

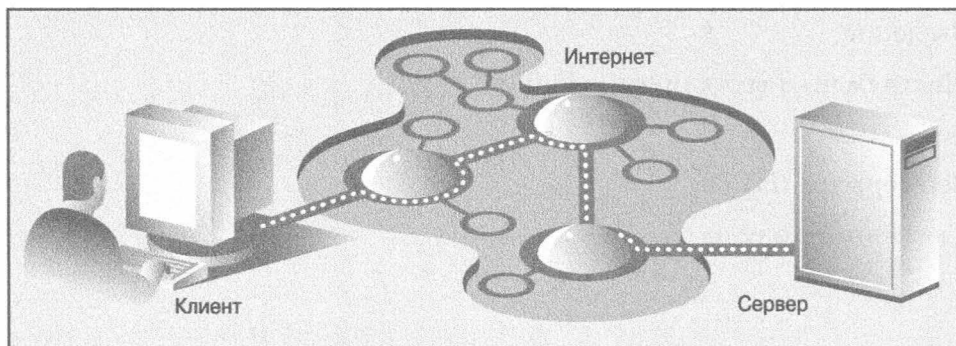


Рис. 9.1. Веб-клиент и веб-сервер в Интернете. Клиент отправляет по Интернету запрос на сервер, который отвечает на этот запрос и отправляет необходимые данные назад клиенту

Запросы, отправляемые веб-клиентами веб-серверам, могут быть разнообразными. В частности, запросы могут предусматривать получение веб-страницы для просмотра или передачу формы с данными для обработки. Поступившие запросы обрабатываются веб-сервером (возможно, с привлечением других систем), а ответ возвращается клиенту в специальном формате, удобном для отображения данных.

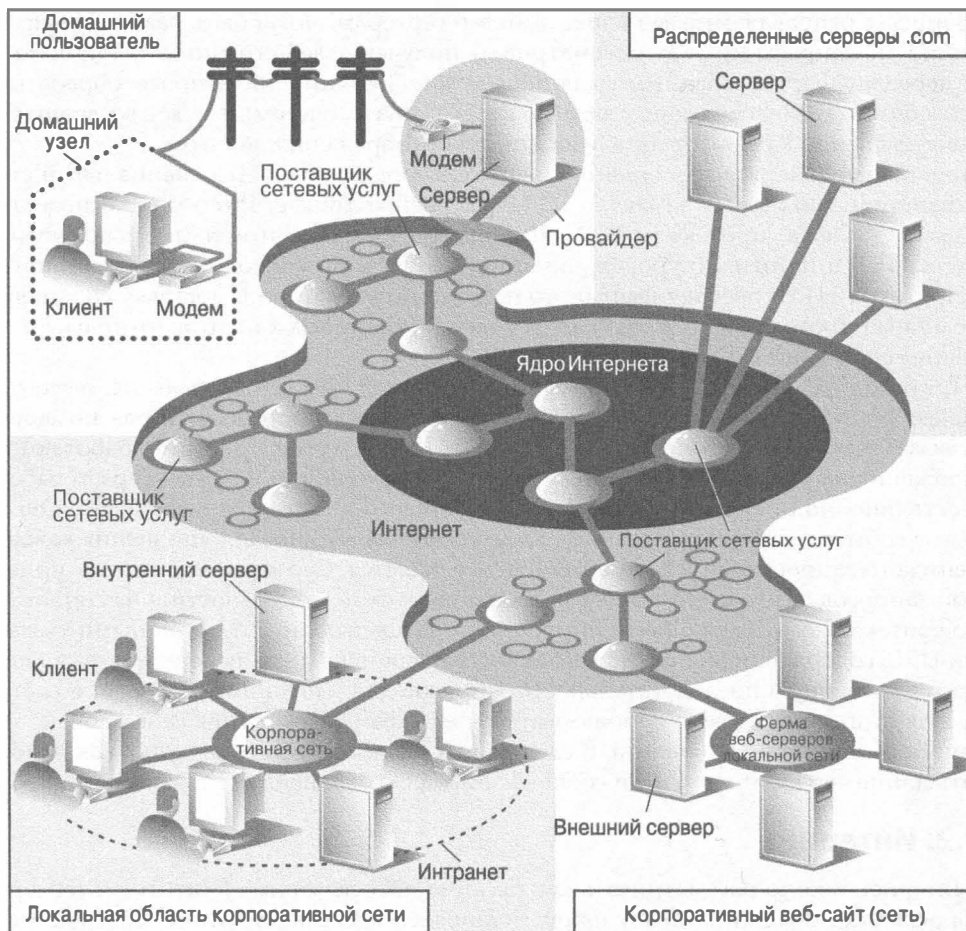
Для взаимодействия веб-клиентов и веб-серверов применяется специальный стандартизированный язык — *протокол HTTP* (HyperText Transfer Protocol). В основе протокола HTTP лежат протоколы TCP и IP. Иными словами, в этом языке используются средства TCP и IP низкого уровня для выполнения всех необходимых операций связи. Протокол HTTP обеспечивает не только маршрутизацию и доставку сообщений (с помощью протоколов TCP и IP), но и ответы на запросы клиентов (отправку и получение сообщений HTTP).

Протокол HTTP, относится к категории *протоколов без сохранения состояния* (stateless protocols), поскольку он не отслеживает информацию, которая позволяла бы связать один клиентский запрос с другим. По такому же принципу работают все приложения в архитектуре “клиент–сервер”, рассмотренные в книге. Сервер работает постоянно, но операции взаимодействия клиента с сервером распадаются на отдельные события, структурированные таким образом, что после выполнения каждого клиентского запроса вся информация о нем удаляется. Сервер всегда готов к приему новых запросов, но каждый из них рассматривается по отдельности. Отсутствие общего контекста, объединяющего запросы, иногда компенсируется длинными указателями URL, содержащими большое количество переменных и значений, образующих структурированную последовательность запросов для имитации сохранения состояния. Для этого могут также использоваться *cookie-файлы* — статические данные, хранящиеся на компьютере клиента. В следующих разделах будет показано, как использовать длинные указатели URL и *cookie-файлы* для сохранения состояния.

## 9.1.2. Интернет

Интернет можно рассматривать как океан взаимосвязанных клиентов и серверов, разбросанных по всему земному шару. Установление связи клиента с сервером можно сравнить с прокладыванием мостов между островами, пока не будет достигнут остров, на котором расположен серверный компьютер. Для пользователя клиентского компьютера все детали подключения к серверу остаются скрытыми. Пользователь работает на более высоком уровне абстракции и видит только прямое соединение между его компьютером (клиентом) и сервером, веб-страницы которого он посещает, а всю черную работу выполняют протоколы HTTP, TCP и IP. Для обычного пользователя не представляет никакого интереса информация о промежуточных узлах, через которые пролег маршрут, поэтому детали взаимодействия клиента и сервера целесообразно скрыть. Общая схема организации Интернета показана на рис. 9.2.

Следует подчеркнуть, что по Интернету передаются как открытые, так и конфиденциальные данные. По умолчанию служба шифрования не предусмотрена, т.е. стандартные протоколы просто передают данные из одного приложения в другое в том виде, в каком они были отправлены. Для повышения безопасности обычные сокететы были дополнены специальным *протоколом уровня защищенных сокетов* (secure socket layer — SSL), обеспечивающим шифрование всех данных, передаваемых через сокет. Теперь разработчики сами вправе решать, должны ли в их приложениях применяться дополнительные средства повышения безопасности.



**Рис. 9.2.** Общая схема организации Интернета. Слева показаны веб-клиенты, а справа — веб-серверы

## Общие сведения о клиентах и серверах

Как показано на рис. 9.2, Интернет состоит из многочисленных взаимосвязанных сетей, функционирующих по определенным правилам. В левой части рисунка показаны веб-клиенты, с которыми работают пользователи. Некоторые пользователи находятся у себя дома и подключаются через поставщика услуг Интернета, а другие выполняют свои должностные обязанности и выходят в Интернет через локальную сеть компании. На этом рисунке не показаны брандмауэры и прокси-серверы, широко применяемые для управления обменом данными в Интернете.

**Брандмауэры (firewalls)** предотвращают несанкционированный доступ к корпоративной (или домашней) сети, блокируя известные точки входа, через которые может проникнуть нарушитель. Конфигурация брандмауэра может настраиваться с учетом особенностей конкретной сети. Если на компьютере, подключенном к глобальной сети, не установлен брандмауэр, то нарушители могут найти незащищенный порт и получить доступ к системе. Сетевые администраторы стремятся снизить вероятность

взлома, блокируя все неиспользуемые порты и открывая возможность доступа только к тем портам, через которые осуществляется обмен данными с конкретными службами, например, на веб-серверах. Еще одним средством защиты является применение доступа исключительно на основе сетевого протокола SSH (Secure Shell), основанного на протоколе SSL.

Еще одним эффективным средством защиты являются *прокси-серверы* (проху-servers), которые могут применяться как вместе с брандмауэрами, так и отдельно. Иногда сетевые администраторы предпочитают предоставлять доступ к Интернету лишь ограниченному количеству компьютеров. Это позволяет лучше контролировать трафик на выходе и входе в локальную сеть. Кроме того, прокси-серверы позволяют кэшировать данные. Это означает, что после посещения сотрудником компании определенной веб-страницы происходит запись в кэш содержимого этой страницы на прокси-сервере. Если произойдет повторное обращение к той же веб-странице того или иного сотрудника, выборка содержимого страницы будет выполнена из кэша, что способствует сокращению времени загрузки. Для получения необходимых данных веб-браузеру больше не потребуется проходить полностью всю процедуру взаимодействия с веб-сервером, поскольку требуемое содержимое уже находится на прокси-сервере. Помимо этого, отдел информационных технологий имеет возможность выяснить, что сотрудники компании посещали те или иные веб-сайты, а также узнать, когда это произошло (и даже какой клиентский компьютер для этого использовался). Такие прокси-серверы называются *прямыми*.

По аналогии действуют *обратные прокси-серверы*, которые могут рассматриваться как (своего рода) противоположность прямым прокси-серверов. (В действительности и прямой, и обратный прокси-серверы могут быть установлены на одном и том же компьютере.) Обратный прокси-сервер принимает запросы от клиентов, поступающие из внешней сети. Чаще всего такие запросы предназначены для получения доступа к *внутреннему серверу* (back-end server) и выборки необходимой информации. Обратные прокси-серверы также могут кэшировать данные и возвращать их непосредственно клиенту.

Таким образом, прямые прокси-серверы располагаются в непосредственной близости от клиентов и кэшируют данные, поступающие в ответ на их запросы из внешней сети, а обратные прокси-серверы располагаются в непосредственной близости от внутренних серверов и кэшируют полученные от них данные. Обратные прокси-серверы часто применяются в качестве посредников при обмене данными между клиентами и другими серверами, выполняя при этом кэширование данных, распределяя нагрузку и т.д. Кроме того, обратные прокси-серверы могут выполнять функции брандмауэров или шифровать данных (при передаче по протоколам SSL, HTTP, Secure FTP и т.д.). Иначе говоря, обратные прокси-серверы осуществляют очень важные функции, поэтому получили широкое распространение. Перейдем к описанию того, что такое внутренние веб-серверы и для чего они предназначены.

В правой части на рис. 9.2 показаны некоторые варианты организации веб-серверов и способы их размещения. Корпорации с большими веб-сайтами, как правило, имеют целые *фермы веб-серверов* (Web-server farm), расположенные на площадках поставщиков услуг Интернета. Физическое размещение серверных компьютеров на общих площадках, называют также *совместным размещением* (co-location). Под этим подразумевается, что серверы компании находятся у поставщика услуг Интернета наряду с компьютерами других корпоративных заказчиков. Эти серверы либо специализируются на предоставлении клиентам различных данных, либо входят в состав резервной системы с дублирующей информацией, предназначенной для работы

в условиях высокой нагрузки (большого количества клиентов). Для меньших корпоративных веб-сайтов может не потребоваться такого большого количества аппаратных средств и сетевого оборудования, как для более крупных, поэтому количество применяемых ими совместно размещенных серверов на площадке поставщика услуг Интернета может ограничиваться одним или несколькими.

В любом случае, тот факт, что большинство совместно размещенных серверов находятся у поставщика услуг Интернета с подключением к *опорной сети* (backbone), означает, что корпоративные веб-сайты имеют более скоростные соединения с высокой пропускной способностью. Это позволяет клиентам быстро получать доступ к серверам, поскольку серверы подключены к опорной сети и маршруты прохождения трафика клиентов на пути к серверам становятся короче. Благодаря этому появляется возможность обслуживать больше клиентов за единицу времени.

## Протоколы Интернета

Несмотря на то что навигация по веб-страницам является самым распространенным способом использования Интернета, следует помнить, что этот способ не единственный и не самый старый. Интернет появился почти на тридцать лет раньше сети веб. До изобретения веб Интернет главным образом использовался образовательными и исследовательскими учреждениями, а передача данных осуществлялась с помощью таких оригинальных протоколов Интернета, как FTP, SMTP и NNTP, которые продолжают широко применяться и в наши дни.

В языке Python в первую очередь были реализованы средства программирования для Интернета, поэтому в нем можно найти поддержку не только для всех указанных выше, но и для многих других протоколов. Мы проводим различие между интернет-программированием и веб-программированием. Под веб-программированием подразумевается создание приложений исключительно для веб, таких как веб-клиенты и веб-серверы. Именно этим приложениям посвящена данная глава.

Интернет-программирование охватывает гораздо более широкий диапазон приложений, включая приложения, в которых используются некоторые из упомянутых выше протоколов Интернета, а также программирование в рамках поддержки сети и сокетов в целом, о чем шла речь в нескольких предыдущих главах.

## 9.2. Инструменты веб-клиентов Python

Необходимо помнить, что браузеры являются лишь одним из типов веб-клиентов. В качестве клиента рассматривается любое приложение, запрашивающее данные от веб-сервера. Клиенты некоторых типов предназначены специально для получения документов или данных из Интернета. Необходимость в использовании клиентов других типов обусловлена ограниченностью функций браузеров, которые предназначены в основном для просмотра содержимого веб-сайтов и взаимодействия с ними. С другой стороны, на клиентскую программу могут быть возложены гораздо более широкие функции — не только загрузка, но и сохранение данных, манипулирование ими и даже передача в другое место или другому приложению.

В качестве простого веб-клиента могут рассматриваться приложения, в которых для загрузки или доступа к информации в Интернете используется модуль `urllib` (с помощью методов `urllib.urlopen()` или `urllib.urlretrieve()`). Для этого достаточно лишь предоставить допустимый веб-адрес.

## 9.2.1. Унифицированный указатель информационного ресурса

Для навигации в сети веб используются веб-адреса — *унифицированные указатели информационных ресурсов* (Uniform Resource Locator — URL). Они служат не только для указания местонахождения документа в веб, но и для вызова программ с интерфейсом CGI, например, для формирования документа, передаваемого клиенту. Указатели URL относятся к более широкому классу идентификаторов, которые называются *универсальными идентификаторами ресурсов* (Uniform Resource Identifier — URI). Идентификаторы URI разработаны в ожидании будущих соглашений об именовании, которые еще только предстоит разработать. Указатели URL можно рассматривать как разновидность идентификаторов URI, в которых в составе адресации используется существующий протокол или схема (например, http, ftp и т.д.). Для полноты картин следует отметить, что вместо термина *универсальный идентификатор ресурса* иногда используется термин *универсальное имя ресурса* (Uniform Resource Name — UNN). В отличие от указателей URL, идентификаторы URI и имена URN не столь широко используются для адресации, а в основном входят в состав идентификаторов XML.

Веб-адреса, как и обычные почтовые адреса, имеют определенную структуру. В США основная часть почтового адреса обычно имеет вид “номер дома и название улицы”, например 123 Main Street. В других странах могут быть приняты другие правила. Формат URL имеет следующий вид:

```
prot_sch://net_loc/path;params?query#frag
```

Компоненты URL описаны в табл. 9.1.

**Таблица 9.1.** Компоненты веб-адреса

| Компонент URL   | Описание                                                               |
|-----------------|------------------------------------------------------------------------|
| <i>prot_sch</i> | Сетевой протокол или схема загрузки                                    |
| <i>net_loc</i>  | Местонахождение сервера (и, возможно, сведения о пользователе)         |
| <i>path</i>     | Путь к файлу или приложению CGI с разделителями в виде косой черты (/) |
| <i>params</i>   | Необязательные параметры                                               |
| <i>query</i>    | Пары “ключ–значение”, разделенные знаками амперсанда (&)               |
| <i>frag</i>     | Фрагмент документа, обозначенный точкой привязки                       |

Компонент адреса *net\_loc* может быть разбит еще на несколько компонентов, обязательных или применяемых в случае необходимости. Строка *net\_loc* выглядит следующим образом:

```
user:passwd@host:port
```

Отдельные компоненты этой строки представлены в табл. 9.2.

**Таблица 9.2.** Компоненты строки, которые указывают на местонахождение в сети

| Компонент     | Описание                                                                                |
|---------------|-----------------------------------------------------------------------------------------|
| <i>user</i>   | Имя пользователя или имя входа                                                          |
| <i>passwd</i> | Пароль пользователя                                                                     |
| <i>host</i>   | Имя или адрес компьютера, на котором работает веб-сервер (обязательный компонент)       |
| <i>port</i>   | Номер порта (если он имеет значение, отличное от применяемого по умолчанию значения 80) |



Среди этих четырех компонентов `host` является самым важным. Параметр с номером порта `port` необходим, только если для веб-сервера применяется номер порта, отличный от предусмотренного по умолчанию. (Дополнительные сведения о номерах портов приведены в главе 2.)

Имена пользователей, а также, возможно, пароли используются только при установлении FTP-соединений, и даже при этом они обычно не требуются, поскольку пользователи таких соединений чаще всего относятся к категории анонимных (anonymous).

В языке Python для работы с URL предусмотрены два разных модуля, в значительной степени отличающихся в части набора функций и прочих возможностей. Это модули `urlparse` и `urllib`. Вкратце рассмотрим функции этих модулей.

## 9.2.2. Модуль `urlparse`

Модуль `urlparse` предоставляет основные функции для манипулирования строками URL. К ним относятся `urlparse()`, `urlunparse()` и `urljoin()`.

### Функция `urlparse.urlparse()`

Функция `urlparse()` разбивает строку URL на некоторые из основных компонентов, описанных ранее. Она имеет следующий синтаксис:

```
urlparse(urlstr, defProtSch=None, allowFrag=None)
```

Функция `urlparse()` проводит синтаксический анализ параметра `urlstr` с разбивкой на шестиэлементный кортеж (`prot_sch`, `net_loc`, `path`, `params`, `query`, `frag`). Каждый из этих компонентов был описан ранее. Параметр `defProtSch` задает применяемый по умолчанию сетевой протокол (или схему загрузки), если он не представлен в параметре `urlstr`. Флаг `allowFrag` указывает, что разрешено использовать часть строки URL с обозначением фрагмента. Ниже приведен пример вывода функции `urlparse()`, полученного при обработке заданного указателя URL:

```
>>> urlparse.urlparse('http://www.python.org/doc/FAQ.html')
('http', 'www.python.org', '/doc/FAQ.html', '', '', '')
```

### Функция `urlparse.urlunparse()`

Функция `urlunparse()` выполняет задачу, прямо противоположную функции `urlparse()`. Эта функция объединяет шестиэлементный кортеж (`prot_sch`, `net_loc`, `path`, `params`, `query`, `frag`) `urltup`, аналогичный тому, который формируется функцией `urlparse()`, в отдельную строку URL и возвращает ее. Соответственно, может быть определено следующее тождество:

```
urlunparse(urlparse(urlstr)) urlstr
```

Как и следовало ожидать, функция `urlunparse()` имеет следующий синтаксис:

```
urlunparse(urltup)
```

## Функция `urlparse.urljoin()`

Функция `urljoin()` применяется в тех случаях, когда требуется сформировать большое количество взаимосвязанных URL, например, для ряда веб-страниц, которые должны войти в состав веб-сайта. Функция `urljoin()` имеет следующий синтаксис:

```
urljoin(baseurl, newurl, allowFrag=None)
```

Функция `urljoin()` принимает в качестве параметра базовый URL, *baseurl*, и соединяет его базовый путь (*net\_loc* плюс полный путь, который ведет к файлу, заданному в конце, но не включает его) с *newurl*. Например:

```
>>> urlparse.urljoin('http://www.python.org/doc/FAQ.html',
... 'current/lib/lib.htm')
'http://www.python.org/doc/current/lib/lib.html'
```

Общие сведения о функциях модуля `urlparse` приведены в табл. 9.3.

**Таблица 9.3.** Основные функции модуля `urlparse`

| Функция <code>urlparse</code>                                  | Описание                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>urlparse(urlstr, defProtSch=None, allowFrag=None)</code> | Проводит синтаксический анализ параметра <i>urlstr</i> с разбиением на отдельные компоненты. Если протокол (или схема) не задан в <i>urlstr</i> , используется <i>defProtSch</i> ; параметр <i>allowFrag</i> определяет, разрешено ли включение фрагмента URL |
| <code>urlunparse(urltup)</code>                                | Проводит операцию, обратную по отношению к синтаксическому анализу, формируя на основе кортежа данных URL ( <i>urltup</i> ) единую строку URL                                                                                                                 |
| <code>urljoin(baseurl, newurl, allowFrag=None)</code>          | Проводит слияние базовой части URL ( <i>baseurl</i> ) с дополнительной частью URL ( <i>newurl</i> ), формируя полный URL; параметр <i>allowFrag</i> применяется так же, как в функции <code>urlparse()</code>                                                 |

## 9.2.3. Модуль/пакет `urllib`



### Основной модуль: `urllib` в Python 2 и Python 3

Если не стоит задача создания сетевого клиента более низкого уровня, то модуль `urllib` предоставляет все необходимые функции. Модуль `urllib` опирается на библиотеку веб-функций высокого уровня, поддерживает основные интернет-протоколы, включая HTTP, FTP и Gopher, а также предоставляет доступ к локальным файлам. В частности, в модуле `urllib` предусмотрены функции загрузки данных (из Интернета, локальной сети или локального узла) с использованием вышеуказанных протоколов. Вообще говоря, применение этого модуля позволяет обойтись без использования модулей `httplib`, `ftplib` и `gopherlib`, если не требуются предусмотренные в них функциональные средства низкого уровня. В подобных случаях указанные модули могут рассматриваться просто как альтернативные. (Примечание. Вообще говоря, модули с именами в формате `*lib` предназначены для разработки клиентов соответствующих протоколов. Однако это правило не всегда соблюдается. Исключением, например, является модуль `urllib`, который следовало бы назвать `internetlib` или присвоить какое-то аналогичное имя!)

## 3.x

Как уже было сказано, в версии Python 2 предусмотрены модули `urllib`, `urllib2`, `urlparse`, `urllib2` и т.п. В версии Python 3 была предпринята попытка упростить работу со всеми этими взаимосвязанными модулями, объединив их в отдельный пакет `urllib`. Поэтому, например, части применявшихся ранее модулей `urllib` и `urllib2` объединены в модуль `urllib.request`, а модуль `urlparse` был преобразован в модуль `urllib.parse`. Пакет `urllib` в Python 3 включает также вспомогательные модули `response`, `error` и `robotparser`. Об этих изменениях следует помнить, изучая материал настоящей главы и рассматривая примеры или упражнения.

Модуль `urllib` предоставляет функции для загрузки данных с веб-серверов, заданных указателями URL, а также кодирования и декодирования произвольных строк для последующего включения в состав действительных строк URL. В следующем разделе рассматриваются функции `urlopen()`, `urlretrieve()`, `quote()`, `unquote()`, `quote_plus()`, `unquote_plus()` и `urlencode()`. Кроме того, приведено описание некоторых методов, которые предусмотрены для работы с файловым объектом, возвращаемым функцией `urlopen()`.

## Метод `urllib.urlopen()`

Метод `urlopen()` открывает интернет-соединение с ресурсом, заданным строкой URL, и возвращает файловый объект. Метод имеет следующий синтаксис:

```
urlopen(urlstr, postData=None)
```

Метод `urlopen()` открывает указатель URL, на который указывает строка `urlstr`. Если не указан протокол (или схема загрузки) или если передана схема файла, то `urlopen()` открывает локальный файл.

Для всех запросов HTTP обычным типом запроса является GET. В этих случаях строка запроса, передаваемая на веб-сервер (пары “ключ–значение”, закодированные или заключенные в кавычки, как в строковом выводе функции `urlencode()`), должна быть задана в составе `urlstr`.

Если требуется метод запроса POST, то строка запроса (опять-таки закодированная) должна быть помещена в переменную `postData`. (Дополнительные сведения о методах запроса GET и POST приведены ниже в этой главе, но следует отметить, что эти методы или команды HTTP относятся к программированию для веб и к самому протоколу HTTP, а не связаны исключительно с языком Python.)

После успешного установления соединения метод `urlopen()` открывает файловый объект, как если бы целевым объектом был файл, открытый в режиме чтения. Предположим, что этот файловый объект имеет имя `f`. В таком случае дескриптор этого файлового объекта поддерживает такие методы чтения, как `f.read()`, `f.readline()`, `f.readlines()`, `f.close()` и `f.fileno()`.

Кроме того, предусмотрен метод `f.info()`, возвращающий заголовки MIME (Multipurpose Internet Mail Extension — многоцелевое расширение почты Интернет). С помощью этих заголовков браузер определяет, какие приложения должны просматривать возвращенные файлы разных типов. Как правило, браузеры позволяют просматривать код HTML и простые текстовые файлы, а также визуализировать графические файлы PNG (Portable Network Graphics), JPEG (Joint Photographic Experts Group) и устаревшие файлы GIF (Graphics Interchange Format). Для просмотра файлов других типов, таких как файлы мультимедиа или файлы документов определенных типов, требуются внешние приложения.

Наконец, метод `geturl()` позволяет получить истинное значение URL для открытого в конечном итоге целевого объекта. В этом указателе URL учитываются все перенаправления, которые могли произойти. Общие сведения об этих методах чтения объектов, подобных файлу, приведены в табл. 9.4.

**Таблица 9.4.** Метод `urllib.urlopen()` чтения файловых объектов

| Методы объекта, возвращаемого методом <code>urlopen()</code> | Описание                                                                                   |
|--------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| <code>f.read([bytes])</code>                                 | Считывает весь объем данных или часть данных в количестве <i>bytes</i> из объекта <i>f</i> |
| <code>f.readline()</code>                                    | Считывает одну строку из объекта <i>f</i>                                                  |
| <code>f.readlines()</code>                                   | Считывает все строки из объекта <i>f</i> в список                                          |
| <code>f.close()</code>                                       | Закрывает соединение URL, относящееся к объекту <i>f</i>                                   |
| <code>f.fileno()</code>                                      | Возвращает номер файла, относящегося к объекту <i>f</i>                                    |
| <code>f.info()</code>                                        | Возвращает заголовки MIME, содержащиеся в объекте <i>f</i>                                 |
| <code>f.geturl()</code>                                      | Возвращает в окончательном виде URL, с помощью которого был открыт объект <i>f</i>         |

Для получения доступа с помощью более сложного указателя URL или для выполнения таких операций, как аутентификация с использованием основного метода или дайджест-проверки, перенаправление, обработка cookie-файлов и т.д., рекомендуется использовать модуль `urllib2`. В этом модуле также предусмотрена функция `urlopen()`, но имеются и другие функции и классы, позволяющие устанавливать соединения с помощью URL многих других типов.

### 2.6, 3.0

Так или иначе, разработчикам, которые намереваются по-прежнему работать в версии 2.x, настоятельно рекомендуется использовать метод `urllib2.urlopen()`, поскольку последний пакет обозначен как устаревший, начиная с версии 2.6, и будет удален в версии 3.0. Как указано в приведенном выше примечании, касающемся модулей, функциональные возможности обоих модулей в версии Python 3 объединены и перенесены в модуль `urllib.request`. Иными словами, функция `urllib.request.urlopen()` версии 3.x перенесена непосредственно из версии 2.x `urllib2.urlopen()` (а не `urllib.urlopen()`).

## Метод `urllib.urlretrieve()`

Метод `urlretrieve()` позволяет загрузить сразу все содержимое HTML-страницы и сохранить его в виде файла. Тем самым можно избавиться от необходимости предварительно открывать URL и получать доступ к обозначенному с его помощью объекту как к файлу. Ниже приведен синтаксис вызова метода `urlretrieve()`.

```
urlretrieve(url, filename=None, reporthook=None, data=None)
```

Вместо чтения из URL, как при использовании метода `urlopen()`, метод `urlretrieve()` загружает весь файл HTML, указанный с помощью параметра *urlstr*, на локальный диск. Если имя файла задано с помощью параметра *localfile*, то загрузка происходит в этот файл. В противном случае используется временный файл.

Если файл был уже скопирован из Интернета или существует локальный файл с тем же именем, то последующая загрузка не происходит.

С помощью параметра `downloadStatusHook` может быть задана функция, которая вызывается после загрузки и доставки каждого блока данных. Указанная функция вызывается со следующими тремя параметрами: количество блоков, считанных к этому времени, размер блока в байтах и общий размер файла в байтах. Эта функция становится очень удобной, если пользователю должна быть предоставлена информация о статусе загрузки в текстовом или графическом виде.

Метод `urlretrieve()` возвращает двухэлементный кортеж (`filename`, `mime_hdrs`), где `filename` — имя локального файла, содержащего загруженные данные; `mime_hdrs` — множество заголовков MIME, возвращенных используемым веб-сервером. Дополнительные сведения см. в описании класса `Message` модуля `mimetools`. Если файл является локальным, то параметр `mime_hdrs` имеет значение `None`.

## Функции `urllib.quote()` и `urllib.quote_plus()`

Функции `quote*()` принимают данные указателя URL и кодируют их с применением формата, позволяющего включать эти данные в состав строки URL. Необходимость преобразования обусловлена тем, что для включения в указатель URL могут передаваться строки со специальными символами, которые не могут быть выведены на экран или являются недопустимыми для непосредственной передачи на веб-сервер. Функции `quote*()` выполняют преобразование таких специальных символов с помощью определенной стандартной кодировки. Обе функции `quote*()` имеют следующий синтаксис:

```
quote(urldata, safe='/')
```

Некоторые символы никогда не преобразовываются. К ним относятся запятые, знаки подчеркивания, точки и дефисы, а также алфавитно-цифровые символы в коде ASCII. Все прочие символы подлежат преобразованию. В частности, отметим, что символы, которые не могут быть заданы непосредственно, заменяются своими шестнадцатеричными эквивалентами в коде, перед которыми ставится префикс в виде знака процента (%), т.е. обозначениями `%xx`, где `xx` — шестнадцатеричное представление значения символа ASCII. После обработки с помощью функции `quote*()` строка `urldata` преобразуется в эквивалентную строку, которая может применяться в составе строки URL. В строке `safe` необходимо указать набор символов, которые также не должны преобразовываться. По умолчанию этот набор символов состоит из косой черты (/).

Функция `quote_plus()` аналогична `quote()`, за исключением того, что в ней кодируются также пробелы с помощью знаков “плюс” (+). Ниже приведен пример использования функции `quote()` вместо `quote_plus()`.

```
>>> name = 'joe mama'
>>> number = 6
>>> base = 'http://www/~foo/cgi-bin/s.py'
>>> final = '%s?name=%s&num=%d' % (base, name, number)
>>> final
'http://www/~foo/cgi-bin/s.py?name=joe mama&num=6'
>>>
>>> urllib.quote(final)
'http://%3a//www/%7efoo/cgi-bin/s.py%3fname%3djoe%20mama%26num%3d6'
>>>
>>> urllib.quote_plus(final)
'http://%3a//www/%7efoo/cgi-bin/s.py%3fname%3djoe+mama%26num%3d6'
```

## Функции `urllib.unquote()` и `urllib.unquote_plus()`

Как и можно было предположить, функции `unquote*()` выполняют действия, прямо противоположные функциям `quote*()`; они преобразовывают все символы, закодированные в виде `%xx`, в их эквиваленты в коде ASCII. Функции `unquote*()` имеют следующий синтаксис:

```
unquote*(urldata)
```

Вызов `unquote()` приводит к декодированию всех символов, представленных в допустимом в URL формате в параметре `urldata`, после чего происходит возврат результирующей строки. Кроме того, функция `unquote_plus()` преобразует знаки “плюс” снова в пробельные символы.

## Метод `urllib.urlencode()`

Метод `urlencode()` принимает в качестве параметра словарь, состоящий из пар “ключ–значение”, и кодирует его для включения в состав запроса в строке URL запроса CGI. Пары представлены в формате `ключ=значение` и разделены амперсандами (`&`). Кроме того, ключи и их значения передаются в функцию `quote_plus()` для кодировки в обусловленном формате. Ниже приведен пример вывода функции `urlencode()`.

```
>>> aDict = { 'name': 'Georgina Garcia', 'hmdir': '~ggarcia' }
>>> urllib.urlencode(aDict)
'name=Georgina+Garcia&hmdir=%7eggarcia'
```

В модулях `urllib` и `urlparse` предусмотрены также другие функции, которые не рассматриваются в данном разделе. Для ознакомления с дополнительными сведениями обратитесь к документации.

Кратко функции `urllib`, описанные в этом разделе, представлены в табл. 9.5.

**Таблица 9.5.** Основные функции модуля `urllib`

| Функции <code>urllib</code>                                               | Описание                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>urlopen(urlstr, postData=None)</code>                               | Открывает URL <code>urlstr</code> и передает данные запроса в параметре <code>postData</code> , если запрос относится к типу POST                                                                                                                                                                                                  |
| <code>urlretrieve(urlstr, localfile=None, downloadStatusHook=None)</code> | Загружает файл, который указан в URL <code>urlstr</code> , в локальный файл <code>localfile</code> или временный файл, если параметр <code>localfile</code> не задан. Параметр <code>downloadStatusHook</code> , если он задан, представляет собой функцию, с помощью которой можно получить статистические данные о ходе загрузки |
| <code>quote(urldata, safe='/')</code>                                     | Кодирует содержащиеся в <code>urldata</code> символы, недопустимые в URL; символы, представленные в строке <code>safe</code> , не кодируются                                                                                                                                                                                       |
| <code>quote_plus(urldata, safe='/')</code>                                | То же, что и <code>quote()</code> , но кодирует также пробелы в виде знака “плюс” (+) (а не в виде <code>%20</code> )                                                                                                                                                                                                              |
| <code>unquote(urldata)</code>                                             | Декодирует символы, закодированные в <code>urldata</code>                                                                                                                                                                                                                                                                          |
| <code>unquote_plus(urldata)</code>                                        | То же, что и <code>unquote()</code> , но преобразовывает знаки “плюс” в пробелы                                                                                                                                                                                                                                                    |

| Функции <code>urllib</code>  | Описание                                                                                                                                                                                                                    |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>urlencode(dict)</code> | Кодирует пары "ключ-значение", заданные с помощью параметра <code>dict</code> , для формирования строки, допустимой для запросов CGI; для кодирования строк ключей и значений применяется функция <code>quote_plus()</code> |

## Поддержка SSL

Прежде чем завершить обсуждение модуля `urllib` и перейти к примерам, отметим, что этот модуль позволяет открывать соединения HTTP с помощью протокола SSL. (Основные изменения, которые позволили ввести поддержку SSL, внесены в модуль `socket`.) Модуль `httplib` поддерживает использование URL для работы по протоколу SSL с помощью схемы соединения `https`. Кроме этих двух модулей, поддержка SSL предусмотрена в следующих клиентских модулях, предназначенных для работы с другими протоколами: `imaplib`, `poplib` и `smtplib`.

### 9.2.4. Пример аутентификации с помощью модуля `urllib2` при установлении соединения по протоколу HTTP

Как было указано в предыдущем разделе, модуль `urllib2` позволяет выполнять более сложные операции при открытии указателей URL. В частности, этот модуль позволяет выполнять обычную аутентификации (по имени и паролю) при доступе к веб-сайтам. Наиболее простое решение по обеспечению безопасности состоит в использовании дополнительного компонента URL, `net_loc`, о чем шла речь ранее в этой главе, например `http://username:passwd@www.python.org`. Это решение имеет один недостаток — для его реализации не применяется программа, но модуль `urllib2` позволяет решить указанную проблему еще двумя способами.

Во-первых, можно создать механизм базовой аутентификации (`urllib2.HTTPBasicAuthHandler`), а также зарегистрировать имя входа и пароль для базового URL и области аутентификации (`realm`), представляющей собой защищенную область веб-сайта, заданную строкой. После создания механизма аутентификации с его помощью можно сформировать и установить инструмент для открытия всех поступающих указателей URL.

Строка с обозначением области аутентификации находится в файле `.htaccess`, который определяет защищенную часть веб-сайта. Пример такого файла приведен ниже.

```
AuthType basic
AuthName "Secure Archive"
AuthUserFile /www/htdocs/.htpasswd
require valid-user
```

В этой части веб-сайта область аутентификации задана параметром `AuthName`. Имя пользователя и (зашифрованный) пароль создаются с использованием команды `htpasswd` (и записываются в файл `.htpasswd`). Дополнительные сведения об областях и процессе аутентификации в веб см. в документе RFC 2617 (HTTP Authentication: Basic and Digest Access Authentication), а также на странице Wikipedia по адресу [http://en.wikipedia.org/wiki/Basic\\_access\\_authentication](http://en.wikipedia.org/wiki/Basic_access_authentication).

В качестве альтернативы можно создать инструмент для открытия URL с механизмом аутентификации, имитирующий поведение пользователя, который вводит имя и пароль после получения приглашения от браузера; иными словами, отправляет клиентский запрос HTTP с соответствующими заголовками аутентификации. Эти два метода демонстрируются в примере 9.1.

### Пример 9.1. Метод базовой аутентификации HTTP (urlopen\_auth.py)

В этом сценарии используются оба описанных ранее метода базовой аутентификации HTTP. Необходимо применять модуль urllib2, поскольку в нем предусмотрены функциональные возможности, которые отсутствуют в модуле urllib.

```
1 #!/usr/bin/env python
2
3 import urllib2
4
5 LOGIN = 'wesley'
6 PASSWD = "you'llNeverGuess"
7 URL = 'http://localhost'
8 REALM = 'Secure Archive'
9
10 def handler_version(url):
11 from urlparse import urlparse
12 hdlr = urllib2.HTTPBasicAuthHandler()
13 hdlr.add_password(REALM,
14 urlparse(url)[1], LOGIN, PASSWD)
15 opener = urllib2.build_opener(hdlr)
16 urllib2.install_opener(opener)
17 return url
18
19 def request_version(url):
20 from base64 import encodestring
21 req = urllib2.Request(url)
22 b64str = encodestring('%s:%s' % (LOGIN, PASSWD))[:-1]
23 req.add_header("Authorization", "Basic %s" % b64str)
24 return req
25
26 for funcType in ('handler', 'request'):
27 print '*** Using %s:' % funcType.upper()
28 url = eval('%s_version' % funcType)(URL)
29 f = urllib2.urlopen(url)
30 print f.readline()
31 f.close()
```

---

## Построчное объяснение

### Строки 1–8

В этой части выполняется обычная настройка и вводятся некоторые константы для использования в остальной части сценария. Еще раз отметим, что конфиденциальная информация должна храниться в защищенной базе данных или по меньшей мере определяться переменными окружения либо в предварительно откомпилированных файлах .рус. Эти данные не следует задавать в виде открытого текста в файле исходного кода.



## Строки 10–17

В версии этого сценария, предусматривающего применение механизма аутентификации, определяется основной класс обработчика, как описано ранее, а затем добавляется информация для аутентификации. После этого механизм аутентификации используется для создания инструмента открытия указателей URL, который затем устанавливается. После этого при открытии всех URL будет выполняться аутентификация. В сценарии использовались фрагменты кода из официальной документации Python к модулю `urllib2`.

## Строки 19–24

В версии сценария, основанной на использовании запросов, создается объект `Request` и в запросы HTTP добавляется простой заголовок аутентификации в кодировке `base64`. Этот объект заменяет строку URL при вызове метода `urlopen()`, когда происходит возврат в функцию `main`. Заслуживает внимания тот факт, что исходный URL “зашифр” в объект `urllib2.Request`. Благодаря этому его можно без проблем заменить в последующих вызовах метода `urllib2.urlopen()`. Этот сценарий был разработан на основании рекомендаций Майкла Фурда (Michael Foord) и Ли Харра (Lee Harr) из книги *Python Cookbook*, которую можно найти по адресу

<http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/305288>  
<http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/267197>

Сценарий стал бы намного лучше, если бы в нем можно было использовать класс `HTTPRealm Finder`, разработанный Харром, поскольку его не пришлось бы повторно определять в коде нашего примера.

## Строки 26–31

В остальной части этого сценария заданный указатель URL просто открывается с использованием обоих методов и отображается первая строка (остальные строки игнорируются) HTML-страницы, возвращаемой сервером после успешной аутентификации. Необходимо отметить, что при неправильно заданной информации об аутентификации происходит возврат ошибки протокола HTTP (без формирования HTML-страницы).

Сформированный вывод должен выглядеть примерно так:

```
$ python urlopen_auth.py
*** Using HANDLER:
<html>
*** Using REQUEST:
<html>
```

Кроме официальной документации Python по `urllib2`, можно порекомендовать ознакомиться со следующей страницей:

<http://www.voidspace.org.uk/python/articles/urllib2.shtml>

## 9.2.5. Перенос примера аутентификации HTTP в Python 3

3.x

Во времени написания книги перенос данного приложения требовал не-много больше работы, чем простое использование инструмента 2to3. Разумеется, этот инструмент позволяет выполнить основную работу, но в дальнейшем приходится вносить некоторые исправления. Еще раз рассмотрим сценарий `urlauth_open.py` и применим к нему указанный инструмент:

```
$ 2to3 -w urlopen_auth.py
...
```

Аналогичную команду можно выполнить и на персональных компьютерах, но, как уже было показано в предыдущих главах, в результатах обнаруживаются различия, связанные с преобразованием сценария из версии Python 2 в версию Python 3, исходный файл перекрывается версией для Python 3, а для версии Python 2 автоматически создается резервная копия.

Переименуем новый файл, полученный в результате обработки файла `urlopen_auth.py`, в `urlopen_auth3.py`, а резервную копию `urlopen_auth.py.bak` снова переименуем в `urlopen_auth.py`. В системах POSIX для этого можно применить одну из многих команд переименования файлов (а на персональных компьютерах для этого можно использовать систему Windows или команду `ren` системы DOS):

```
$ mv urlopen_auth.py urlopen_auth3.py
$ mv urlopen_auth.py.bak urlopen_auth.py
```

Таким образом, мы по-прежнему применяем принятую ранее в данной книге стратегию именования сценариев, позволяющую отличать программы, подготовленные для версии Python 2, от перенесенных в версию Python 3. Так или иначе, в настоящем примере вызов указанного инструмента для переноса в другую версию — только первый шаг. Если еще была надежда, что достичь успеха удастся с первого раза, то она быстро рассеялась:

```
$ python3 urlopen_auth3.py
*** Using HANDLER:
b'<HTML>\n'
*** Using REQUEST:
Traceback (most recent call last):
 File "urlopen_auth3.py", line 28, in <module>
 url = eval('%s_version' % funcType)(URL)
 File "urlopen_auth3.py", line 22, in request_version
 b64str = encodestring('%s:%s' % (LOGIN, PASSWD))[:-1]
 File "/Library/Frameworks/Python.framework/Versions/3.2/lib/python3.2/base64.py",
line 353, in encodestring
 return encodebytes(s)
 File "/Library/Frameworks/Python.framework/Versions/3.2/lib/python3.2/base64.py",
line 341, in encodebytes
 raise TypeError("expected bytes, not %s" % s.__class__.__name__)
TypeError: expected bytes, not str
```

Попытаемся воспользоваться отработанным подходом и внесем изменения в строковую переменную в строке 22, добавив ведущий символ “b” перед открывающей кавычкой, как в примере `b'%s:%s' % (LOGIN, PASSWD)`. Теперь после очередной

попытки выполнить сценарий возникает другая ошибка (приветствуем новых членов клуба специалистов по переносу в версию Python 3!):

```
$ python3 urlopen_auth3.py
*** Using HANDLER:
b'<HTML>\n'
*** Using REQUEST:
Traceback (most recent call last):
 File "urlopen_auth3.py", line 28, in <module>
 url = eval('%s version' % funcType)(URL)
 File "urlopen_auth3.py", line 22, in request_version
 b64str = encodestring(b'%s:%s' % (LOGIN, PASSWD))[:-1]
TypeError: unsupported operand type(s) for %: 'bytes' and 'tuple'
```

Очевидно, что объекты `bytes` не поддерживают оператор форматирования строк, поскольку (даже с формальной точки зрения) такие объекты не должны использоваться как строки. Вместо этого строку необходимо отформатировать как текст (в Юникоде), а затем преобразовать результат в объект `bytes`: `bytes('%s:%s' % (LOGIN, PASSWD), 'utf-8')`. После этого изменения сформированный вывод начинает в большей степени напоминать ожидаемый:

```
$ python3 urlopen_auth3.py
*** Using HANDLER:
b'<HTML>\n'
*** Using REQUEST:
b'<HTML>\n'
```

Все же мы не смогли достичь полного соответствия, поскольку в сценарии применяются объекты `bytes` (с ведущими символами “b”, кавычками и т.д.) помимо того текста, который нас интересует. Заменяем вызов функции `print()` следующим: `print(str(f.readline(), 'utf-8'))`. Теперь вывод, полученный с помощью версии Python 3, становится идентичным выводу сценария Python 2:

```
$ python3 urlopen_auth3.py
*** Using HANDLER:
<html>

*** Using REQUEST:
<html>
```

Вполне очевидно, что для переноса в новую версию требуется определенный объем ручной работы, но само по себе решение этой задачи вполне осуществимо. Еще раз следует отметить, что в версии Python 3 модули `urllib`, `urllib2` и `urlparse` полностью объединены под маркой `urllib`. Учитывая то, как организована работа инструмента `2to3`, импорт модуля `urllib.parse` уже обеспечивается. Поэтому соответствующий оператор в определении метода `handler_version()` стал лишним и был удален. Описанные изменения наряду с прочими исправлениями показаны в примере 9.2.

#### Пример 9.2. Сценарий аутентификации HTTP для версии Python 3 (`urlopen_auth3.py`)

Эта версия сценария `urlopen_auth.py` для Python 3.

```
1 #!/usr/bin/env python3
2
```

```
3 import urllib.request, urllib.error, urllib.parse
4
5 LOGIN = 'wesley'
6 PASSWD = "you'llNeverGuess"
7 URL = 'http://localhost'
8 REALM = 'Secure Archive'
9
10 def handler_version(url):
11 hdlr = urllib.request.HTTPBasicAuthHandler()
12 hdlr.add_password(REALM,
13 urllib.parse.urlparse(url)[1], LOGIN, PASSWD)
14 opener = urllib.request.build_opener(hdlr)
15 urllib.request.install_opener(opener)
16 return url
17
18 def request_version(url):
19 from base64 import encodestring
20 req = urllib.request.Request(url)
21 b64str = encodestring(
22 bytes('%s:%s' % (LOGIN, PASSWD), 'utf-8'))[:-1]
23 req.add_header("Authorization", "Basic %s" % b64str)
24 return req
25
26 for funcType in ('handler', 'request'):
27 print('*** Using %s:' % funcType.upper())
28 url = eval('%s_version' % funcType)(URL)
29 f = urllib.request.urlopen(url)
30 print(str(f.readline(), 'utf-8'))
31 f.close()
```

Теперь перейдем к рассмотрению немного более сложных веб-клиентов.

## 9.3. Веб-клиенты

К основным типам веб-клиентов относятся веб-браузеры. Их главное назначение состоит в поиске и загрузке документов из Интернета. Однако некоторые веб-клиенты предназначены для выполнения более широких функций. Некоторые из них будут рассматриваться в этом разделе.

### 9.3.1. Простой поисковый робот, спайдер, бот

Примером более сложного веб-клиента является *поисковый робот* (crawler), или *спайдер* (spider), или *бот* ([ro]bots). Это программа, позволяющая исследовать ресурсы Интернета и загружать содержимое страниц для решения многих задач, включая следующие:

- индексация с помощью большой поисковой машины, например Google или Yahoo!;
- автономный просмотр, т.е. загрузка содержимого веб-сайта на локальный жесткий диск, переупорядочение гиперссылок для создания почти зеркального отображения сайта и просмотр его на локальном компьютере;
- загрузка и сохранение данных для ведения журнала или архива;
- кэширование веб-страниц для сокращения времени загрузки на случай повторного посещения веб-сайта.

Поисковый робот, рассматриваемый в примере 9.3, `crawl.py`, принимает исходный веб-адрес (URL), загружает соответствующую веб-страницу, а затем другие страницы, ссылки на которые выбирает пользователь, но только находящиеся в том же домене, что и исходная страница. Соблюдение таких ограничений необходимо для экономии места на диске.

### Пример 9.3. Поисковый робот (`crawl.py`)

Поисковый робот состоит из двух классов; один из них управляет всем процессом навигации, т.е. перехода по ссылкам (Crawler), а второй обеспечивает получение и интерпретацию каждой загруженной веб-страницы (Retriever). (Эта программа стала результатом доработки тех версий, которые были приведены в предыдущих изданиях данной книги.)

```

1 #!/usr/bin/env python
2
3 import cStringIO
4 import formatter
5 from htmllib import HTMLParser
6 import httplib
7 import os
8 import sys
9 import urllib
10 import urlparse
11
12 class Retriever(object):
13 __slots__ = ('url', 'file')
14 def __init__(self, url):
15 self.url, self.file = self.get_file(url)
16
17 def get_file(self, url, default='index.html'):
18 'Create usable local filename from URL'
19 parsed = urlparse.urlparse(url)
20 host = parsed.netloc.split('@')[-1].split(':')[0]
21 filepath = '%s%s' % (host, parsed.path)
22 if not os.path.splitext(parsed.path)[1]:
23 filepath = os.path.join(filepath, default)
24 linkdir = os.path.dirname(filepath)
25 if not os.path.isdir(linkdir):
26 if os.path.exists(linkdir):
27 os.unlink(linkdir)
28 os.makedirs(linkdir)
29 return url, filepath
30
31 def download(self):
32 'Download URL to specific named file'
33 try:
34 retval = urllib.urlretrieve(self.url, self.file)
35 except (IOError, httplib.InvalidURL) as e:
36 retval = (('*** ERROR: bad URL "%s": %s' % (
37 self.url, e)),)
38 return retval
39
40 def parse_links(self):
41 'Parse out the links found in downloaded HTML file'
42 f = open(self.file, 'r')

```

```

43 data = f.read()
44 f.close()
45 parser = HTMLParser(formatter.AbstractFormatter(
46 formatter.DumbWriter(cStringIO.StringIO())))
47 parser.feed(data)
48 parser.close()
49 return parser.anchorlist
50
51 class Crawler(object):
52 count = 0
53
54 def __init__(self, url):
55 self.q = [url]
56 self.seen = set()
57 parsed = urlparse.urlparse(url)
58 host = parsed.netloc.split('@')[-1].split(':')[0]
59 self.dom = '.'.join(host.split('.')[-2:])
60
61 def get_page(self, url, media=False):
62 'Download page & parse links, add to queue if nec'
63 r = Retriever(url)
64 fname = r.download()[0]
65 if fname[0] == '*':
66 print fname, '... skipping parse'
67 return
68 Crawler.count += 1
69 print '\n(', Crawler.count, ')'
70 print 'URL:', url
71 print 'FILE:', fname
72 self.seen.add(url)
73 ftype = os.path.splitext(fname)[1]
74 if ftype not in ('.htm', '.html'):
75 return
76
77 for link in r.parse_links():
78 if link.startswith('mailto:'):
79 print '... discarded, mailto link'
80 continue
81 if not media:
82 ftype = os.path.splitext(link)[1]
83 if ftype in ('.mp3', '.mp4', '.m4v', '.wav'):
84 print '... discarded, media file'
85 continue
86 if not link.startswith('http://'):
87 link = urlparse.urljoin(url, link)
88 print '*', link,
89 if link not in self.seen:
90 if self.dom not in link:
91 print '... discarded, not in domain'
92 else:
93 if link not in self.q:
94 self.q.append(link)
95 print '... new, added to Q'
96 else:
97 print '... discarded, already in Q'
98 else:
99 print '... discarded, already processed'
100

```

```
101 def go(self, media=False):
102 'Process next page in queue (if any)'
103 while self.q:
104 url = self.q.pop()
105 self.get_page(url, media)
106
107 def main():
108 if len(sys.argv) > 1:
109 url = sys.argv[1]
110 else:
111 try:
112 url = raw_input('Enter starting URL: ')
113 except (KeyboardInterrupt, EOFError):
114 url = ''
115 if not url:
116 return
117 if not url.startswith('http://') and \
118 not url.startswith('ftp://'):
119 url = 'http://%s/' % url
120 robot = Crawler(url)
121 robot.go()
122
123 if __name__ == '__main__':
124 main()
```

---

## Построчное описание (с рассмотрением отдельных классов)

### Строки 1–10

В начале сценария находятся стандартные строки вызова интерпретатора Python в системе Unix и операторы импорта модулей/пакетов, используемых в сценарии. Ниже приведены краткие описания.

- `cStringIO`, `formatter`, `htmllib`. В этих модулях для интерпретации кода HTML используются разные классы.
- `httplib`. Из этого модуля берется только определение исключения.
- `os`. Этот модуль предоставляет необходимые функции для работы с файловой системой.
- `sys`. Этот модуль применяется только для вызова массива `argv` и обработки параметров командной строки.
- `urllib`. Этот модуль требуется для получения доступа к функции `urlretrieve()` и загрузки веб-страниц.
- `urlparse`. Функции `urlparse()` и `urljoin()` этого модуля применяются для выполнения операций обработки URL.

### Строки 12–29

Класс `Retriever` обеспечивает загрузку страниц из Интернета и интерпретацию ссылок, находящихся в каждом загруженном документе; затем полученные ссылки по мере необходимости добавляются в очередь для последующей обработки. Экземпляр объекта `Retriever` создается для каждой страницы, загружаемой из Интернета.

Для расширения функциональных возможностей класса `Retriever` применяются несколько методов: сам конструктор (`__init__()`), а также методы `get_file()`, `download()` и `parse_links()`.

Забегая вперед, отметим, что метод `get_file()` принимает заданный URL, затем формирует уникальное и соответствующее заданному шаблону имя файла, предназначенное для сохранения на локальном диске содержимого, загруженного из Интернета. Действия указанного метода по формированию имени файла вкратце можно описать как удаление префикса `http://` из URL и исключение всех лишних компонентов, таких как имя пользователя, пароль и номер порта, для получения имени хоста (строка 20).

Если в URL отсутствует суффикс в виде расширения файла, то ему присваивается заданное по умолчанию имя файла `index.html`, которое может быть переопределено в вызывающем коде. Сами эти операции, а также операцию создания окончательного значения `filepath` можно видеть в строках 21–23.

После этого происходит определение имени конечного каталога назначения (строка 24) и проверка того, существует ли этот каталог; в случае положительного ответа каталог остается нетронутым и происходит возврат пары “URL–путь к файлу”. Если произошел переход в конструкцию `if`, значит, каталог не существует или под этим именем в файловой системе записан файл. В последнем случае файл должен быть уничтожен. Наконец, создается целевой каталог, включая все промежуточные каталоги, с помощью функции `os.makedirs()` (строка 28).

Теперь вернемся к инициализатору `__init__()`. Создается экземпляр объекта `Retriever`, который сохраняет в качестве атрибутов (экземпляра) и сам URL (`str`), и соответствующее имя файла, возвращенное методом `get_file()`. В текущем проекте экземпляры создаются для каждого загруженного файла. Если веб-сайт состоит из чрезвычайно большого количества файлов, то даже при наличии подобных малых экземпляров потребность в памяти может стать весьма значительной. Для сокращения до минимума используемых ресурсов создается переменная `__slots__`, которая указывает, что экземпляры могут иметь только атрибуты `self.url` и `self.file`.

## Строки 31–49

Вскоре мы перейдем к подробному описанию работы поискового робота, а пока отметим, что в нем экземпляры объектов `Retriever` создаются для каждого загруженного файла. Как и следовало ожидать, метод `download()` применяется для выхода в Интернет и загрузки страницы по указанной ссылке (строка 34). Этот метод вызывает функцию `urllib.urlretrieve()` с параметром в виде URL, а затем сохраняет полученное содержимое в файле (имя которого получено от функции `get_file()`).

Если загрузка оказалась успешной, происходит возврат имени файла (строка 34), но если возникает ошибка, вместо этого возвращается строка с описанием ошибки, обозначенная префиксом `***` (строки 35–36). Поисковый робот проверяет это возвращаемое значение и вызывает функцию `parse_links()` для синтаксического анализа и выделения ссылок из вновь загруженной страницы, но лишь при условии, что все предыдущие действия были выполнены успешно.

Более важным методом в этой части рассматриваемого приложения является метод `parse_links()`. Очевидно, что задача поискового робота состоит в загрузке веб-страниц, но рекурсивный поисковый робот (такой как наш) отыскивает новые ссылки на каждой загруженной странице и также их обрабатывает. Поисковый робот прежде всего открывает загруженную веб-страницу и извлекает все содержимое HTML в виде одной строки (строки 42–44).



Загадочные манипуляции, выполняемые в строках 45–49, взяты из известного рецепта, в котором используется класс `htmllib.HTMLParser`. Хотелось бы заинтриговать читателя словами, что этот рецепт передается программистами на языке Python из поколения в поколение, но, к сожалению, это неправда. Так или иначе, приступим к описанию кода в указанных строках.

Основной смысл осуществляемых действий заключается в том, что класс синтаксического анализатора не предусматривает ввода-вывода, поэтому для выполнения такой задачи применяется объект `formatter`. Что касается объектов форматирования, то в языке Python имеется только один программный компонент, заслуживающий этого названия: `formatter.AbstractFormatter`. Этот объект осуществляет синтаксический анализ данных и использует объект записи для вывода. Аналогичным образом в языке Python имеется лишь один действительно удобный объект записи: `formatter.DumbWriter`. В качестве необязательного параметра этот объект принимает объект файла, в который должна быть выполнена запись. Если этот параметр не будет задан, то запись происходит в стандартное устройство вывода, что, по-видимому, нежелательно. Учитывая это, мы создаем в рассматриваемом сценарии экземпляр объекта `cStringIO`. Экземпляр объекта `StringIO` служит для перенаправления в него вывода (что можно сравнить с использованием фиктивного устройства `/dev/null` в операционной системе). Читатель поискать казанные имена классов в Интернете и найти во многих местах аналогичные фрагменты кода с дополнительными комментариями.

Объект `htmllib.HTMLParser` содержит довольно большой объем кода и считается устаревшим, начиная с версии 2.6, поэтому в следующем разделе приведен более краткий пример, демонстрирующий применение современных функциональных средств. В данном примере мы не отказываемся от применения этого общепринятого рецепта, поскольку он по-прежнему остается вполне подходящим для достижения той цели, для которой он предназначен.

Так или иначе, достаточно сложная задача создания синтаксического анализатора свелась к применению единственного вызова (строки 45–46). Остальная часть этого блока предусматривает передачу кода HTML, закрытие средства синтаксического анализа, затем возврат списка интерпретированных ссылок/точек привязки.

## Строки 51–59

Основой этого приложения является класс `Crawler`, который управляет всем процессом обхода страниц для отдельного веб-сайта. Добавление средств поддержки многопоточности к рассматриваемому приложению позволило бы создать отдельные экземпляры для каждого узла, по страницам которого осуществляется обход. Класс `Crawler` состоит из трех элементов, сохраняемых конструктором на этапе создания экземпляра. Первым из них является `self.q`, очередь ссылок, подлежащих загрузке. В ходе выполнения размеры этой очереди, представленной в виде списка, все время изменяются. После обработки каждой страницы список сокращается, а вслед за обнаружением новых ссылок на очередной загруженной странице — расширяется.

В число двух других значений данных для `Crawler` входит `self.seen`, множество, содержащее все ссылки, просмотренные (загруженные) ранее. Наконец, сохраняется доменное имя для основной ссылки, `self.dom`. Это значение используется для определения того, являются ли какие-либо последующие ссылки частью того же домена. Все три значения создаются в методе инициализатора `__init__()`, который представлен в строках 54–59.

Обратите внимание, что синтаксический анализ имени домена осуществляется с использованием метода `urlparse.urlparse()` (строка 58), так же, как и при определении имени хоста из URL в объекте `Retriever`. Чтобы получить имя домена, достаточно взять две последние части имени хоста. Следует учитывать, что узел не используется для чего-либо иного, поэтому можно сократить код, объединив строки 58 и 59, как показано ниже, но в таком случае код становится более сложным для восприятия:

```
self.dom = '.'.join(urlparse.urlparse(
 url).netloc.split('@')[-1].split(':')[0].split('.')[2:])
```

Непосредственно над определением метода `__init__()` в классе `Crawler` имеется также элемент статических данных с именем `count`. Это счетчик, назначение которого состоит в отслеживании количества объектов, которые мы загрузили из Интернета. Он увеличивается после каждой успешно загруженной страницы.

## Строки 61–105

В классе `Crawler`, кроме конструктора, представлены другие важные методы: `get_page()` и `go()`. Метод `go()` используется исключительно для запуска метода `Crawler`. Он вызывается из основной части кода. В основе метода `go()` лежит цикл, который продолжит выполняться, пока еще в очереди имеются новые ссылки, которые должны быть загружены. Тем не менее основной движущей силой этого класса является метод `get_page()`.

Метод `get_page()` после получения первой ссылки создает объект `Retriever` и запускает его работу. Если страница загружена успешно, счетчик увеличивается и ссылка добавляется к множеству уже просмотренных (строка 72). В противном случае ссылки, ставшие причиной возникновения ошибок, пропускаются (строки 65–67). В данном случае используется множество, поскольку порядок следования ссылок не имеет значения, а поиск в множестве происходит намного быстрее, чем в списке.

Метод `get_page()` просматривает все ссылки, содержащиеся в каждой загруженной странице (при этом пропускаются все ссылки, не относящиеся к веб-страницам (строки 73–75)) и определяет, следует ли продолжать добавление ссылок в очередь (строки 77–99). В главном цикле метода `go()` обработка ссылок продолжается, пока очередь не опустеет, после чего формируется результат с данными об успешном завершении (строки 103–105).

Ссылки, которые относятся к другому домену (строки 90–91) или уже были загружены (строки 98–99), теперь находятся в очереди ссылок, ожидающих обработки (строки 96–97), или относятся к типу `mailto:`: ссылки пропускаются и не добавляются к очереди (строки 78–80). То же касается медиафайлов (строки 81–85).

## Строки 107–124

Для начала обработки в метод `main()` должен быть передан указатель URL. Если он вводится в командной строке (например, при непосредственном вызове этого сценария; строки 108–109), работа начинается с него. В противном случае сценарий переходит в интерактивный режим и пользователю передается запрос на ввод начального URL (строка 112). После получения исходной ссылки порождается экземпляр класса `Crawler` и начинается выполнение сценария (строки 120–121).

Пример вызова `crawl.py` может выглядеть примерно так:

```
$ crawl.py
Enter starting URL: http://www.null.com/home/index.html
```

```
(1)
URL: http://www.null.com/home/index.html
FILE: www.null.com/home/index.html
* http://www.null.com/home/overview.html ... new, added to Q
* http://www.null.com/home/synopsis.html ... new, added to Q
* http://www.null.com/home/order.html ... new, added to Q
* mailto:postmaster@null.com ... discarded, mailto link
* http://www.null.com/home/overview.html ... discarded, already in Q
* http://www.null.com/home/synopsis.html ... discarded, already in Q
* http://www.null.com/home/order.html ... discarded, already in Q
* mailto:postmaster@null.com ... discarded, mailto link
* http://bogus.com/index.html ... discarded, not in domain

(2)
URL: http://www.null.com/home/order.html
FILE: www.null.com/home/order.html
* mailto:postmaster@null.com ... discarded, mailto link
* http://www.null.com/home/index.html ... discarded, already processed
* http://www.null.com/home/synopsis.html ... discarded, already in Q
* http://www.null.com/home/overview.html ... discarded, already in Q

(3)
URL: http://www.null.com/home/synopsis.html
FILE: www.null.com/home/synopsis.html
* http://www.null.com/home/index.html ... discarded, already processed
* http://www.null.com/home/order.html ... discarded, already processed
* http://www.null.com/home/overview.html ... discarded, already in Q

(4)
URL: http://www.null.com/home/overview.html
FILE: www.null.com/home/overview.html
* http://www.null.com/home/synopsis.html ... discarded, already processed
* http://www.null.com/home/index.html ... discarded, already processed
* http://www.null.com/home/synopsis.html ... discarded, already processed
* http://www.null.com/home/order.html ... discarded, already processed
```

После выполнения этого сценария в локальной файловой системе создается каталог `www.null.com` с подкаталогом `home`. В подкаталоге `home` находятся все обработанные файлы.

Если после просмотра представленного выше кода читатель захочет понять, имеет ли смысл заниматься созданием программы обхода веб-страниц на языке Python, то ему следует знать, что первые версии поисковых роботов Google были написаны на языке Python. Дополнительные сведения см. в разделе <http://infolab.stanford.edu/~backrub/google.html>.

### 9.3.2. Синтаксический анализ содержимого веб-страниц

В предыдущем подразделе мы рассмотрели поисковый робот. Часть глобального поиска в Интернете составляет синтаксический анализ ссылок, которые носят официальное название *точек привязки*. В течение долгого времени для синтаксического анализа веб-страниц широко известный класс `htmllib.HTMLParser`; но в дальнейшем были разработаны новые, более совершенные модули и пакеты. Некоторые из них будут рассматриваться в данном подразделе.

В примере 9.4 мы изучим стандартный библиотечный инструмент, класс `HTMLParser` в модуле `HTMLParser` (впервые введенный в версии 2.2). Предполагалось, что модуль `HTMLParser.HTMLParser` должен заменить `htmllib.HTMLParser`, поскольку

он проще, обеспечивает представление содержимого более низкого уровня и обработку кода XHTML. По сравнению с ним модуль `html5lib.HTMLParser` является менее современным и более сложным, так как основан на модуле `sgmlib`, а это означает, что в нем должны учитываться нюансы языка SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки). В официальной документации приведены довольно отрывочные сведения о том, как использовать `HTMLParser`. `HTMLParser`, но мы надеемся, что приведенный здесь пример окажется более полезным.

Мы также продемонстрируем использование еще двух из трех других средств синтаксического анализа, наиболее широко применяемых в Интернете, `BeautifulSoup` и `html5lib`, которые доступны для загрузки отдельно, в дополнение к стандартной библиотеке. Доступ к этим двум синтаксическим анализаторам можно получить на сайте `Cheeseshop` или по адресу `http://pypi.python.org`. Для того чтобы испытывать меньше трудностей при установке, можно также воспользоваться для получения того или другого инструментами `easy_install` или `pip`.

Из числа трех наиболее популярных средств синтаксического анализа мы не включили `lxml`; изучение его оставляем читателю в качестве упражнения. В конце главы приведены также другие упражнения, с помощью которых можно более подробно узнать, какие нюансы связаны с применением новых модулей вместо `html5lib`.

## Применение модуля `HTMLParser` в поисковом роботе

Сценарий `parse_links.py`, приведенный в примере 9.4, предназначен исключительно для выявления точек привязки с помощью синтаксического анализа интерпретации в любых входных данных. После получения URL этот сценарий извлекает все ссылки, пытается произвести все необходимые корректировки для преобразования этих ссылок в полноценные URL, затем проводит сортировку указателей и отображает их для пользователя. Для обработки каждого URL применяются три средства синтаксического анализа. В частности, для `BeautifulSoup` предоставляются два различных решения. Первое из них проще и предусматривает синтаксический анализ всех дескрипторов, после чего проводится поиск всех дескрипторов привязки. Второе решение требует использования класса `SoupStrainer`, который специально предназначен для работы с дескрипторами привязки и их синтаксического анализа.

### Пример 9.4. Средство синтаксического анализа ссылок (`parse_links.py`)

В этом сценарии используются три различных средства синтаксического анализа для извлечения ссылок из дескрипторов привязки HTML. Это стандартный библиотечный модуль `HTMLParser`, а также пакеты `BeautifulSoup` и `html5lib` сторонних разработчиков.

```

1 #!/usr/bin/env python
2
3 from HTMLParser import HTMLParser
4 from cStringIO import StringIO
5 from urllib2 import urlopen
6 from urlparse import urljoin
7
8 from BeautifulSoup import BeautifulSoup, SoupStrainer
9 from html5lib import parse, treebuilders
10
11 URLs = (
12 'http://python.org',
13 'http://google.com',

```

```

14)
15
16 def output(x):
17 print '\n'.join(sorted(set(x)))
18
19 def simpleBS(url, f):
20 'simpleBS() - use BeautifulSoup to parse all tags to get anchors'
21 output(urljoin(url, x['href']) for x in BeautifulSoup(
22 f).findAll('a'))
23
24 def fasterBS(url, f):
25 'fasterBS() - use BeautifulSoup to parse only anchor tags'
26 output(urljoin(url, x['href']) for x in BeautifulSoup(
27 f, parseOnlyThese=SoupStrainer('a')))
28
29 def htmlparser(url, f):
30 'htmlparser() - use HTMLParser to parse anchor tags'
31 class AnchorParser(HTMLParser):
32 def handle_starttag(self, tag, attrs):
33 if tag != 'a':
34 return
35 if not hasattr(self, 'data'):
36 self.data = []
37 for attr in attrs:
38 if attr[0] == 'href':
39 self.data.append(attr[1])
40 parser = AnchorParser()
41 parser.feed(f.read())
42 output(urljoin(url, x) for x in parser.data)
43
44 def html5libparse(url, f):
45 'html5libparse() - use html5lib to parse anchor tags'
46 output(urljoin(url, x.attributes['href']) \
47 for x in parse(f) if isinstance(x,
48 treebuilders.simpletree.Element) and \
49 x.name == 'a')
50
51 def process(url, data):
52 print '\n*** simple BS'
53 simpleBS(url, data)
54 data.seek(0)
55 print '\n*** faster BS'
56 fasterBS(url, data)
57 data.seek(0)
58 print '\n*** HTMLParser'
59 htmlparser(url, data)
60 data.seek(0)
61 print '\n*** HTML5lib'
62 html5libparse(url, data)
63
64 def main():
65 for url in URLs:
66 f = urlopen(url)
67 data = StringIO(f.read())
68 f.close()
69 process(url, data)
70
71 if __name__ == '__main__':
72 main()

```

## Построчное объяснение

### Строки 1–9

Кроме всего прочего, в этом сценарии используются четыре модуля из стандартной библиотеки. Модуль `HTMLParser` является одним из средств синтаксического анализа; три другие модуля относятся к категории программных средств общего назначения. Вторая группа инструкций импорта относится к модулям и (или) пакетам сторонних разработчиков (не относящиеся к стандартной библиотеке). Применяемая в этом сценарии последовательность инструкций импорта соответствует общепринятому стандарту расположения этих инструкций. В первую очередь располагаются инструкции импорта стандартных библиотечных модулей (пакетов), за ними следуют инструкции для работы со средствами сторонних разработчиков и, наконец, размещается раздел импорта модулей (пакетов), локальных по отношению к приложению.

### Строки 11–17

В переменной `URLs` перечислены анализируемые веб-страницы; здесь можно произвольно добавлять, изменять или удалять URL. Функция `output()` принимает итератор по ссылкам, удаляет дубликаты, помещая все ссылки в множество, сортирует ссылки в лексикографическом порядке, а затем объединяет их в строку с разделителем в виде символа новой строки, которая отображается для пользователя.

### Строки 19–27

Следует подчеркнуть, что в функциях `fasterBS()` и `simpleBS()` используется интерфейс `BeautifulSoup`. В функции `simpleBS()` интерпретация происходит при создании экземпляра `BeautifulSoup` с помощью дескриптора файла. В следующем коротком фрагменте кода выполняется именно это, для чего используется уже загруженная страница с веб-сайта `PyCon` с адресом `pycon.html`.

```
>>> from BeautifulSoup import BeautifulSoup as BS
>>> f = open('pycon.html')
>>> bs = BS(f)
```

После получения экземпляра и вызова его метода `findAll()`, запрашивающего дескриптор точки привязки ("`a`"), возвращается список дескрипторов, как показано ниже.

```
>>> type(bs)
<class 'BeautifulSoup.BeautifulSoup'>
>>> tags = bs.findAll('a')
>>> type(tags)
<type 'list'>
>>> len(tags)
19
>>> tag = tags[0]
>>> tag
PyCon 2011 Atlanta
>>> type(tag)
<class 'BeautifulSoup.Tag'>
>>> tag['href']
u'/2011/'
```

Объект `Tag` — это точка привязки, которая должна иметь дескриптор `"href"`, поэтому мы запрашиваем именно это значение. Затем происходит вызов метода `urlparse.urljoin()` и передача основной части URL вместе со ссылкой, позволяющей получить полный URL. Это продолжение нашего примера, который рассматривался в предыдущих главах (принято предположение об использовании URL PyCon):

```
>>> from urlparse import urljoin
>>> url = 'http://us.pycon.org'
>>> urljoin(url, tag['href'])
u'http://us.pycon.org/2011/'
```

Выражение генератора выполняет итерацию по всем ссылкам, окончательно сформированным методом `urlparse.urljoin()` из всех дескрипторов точки привязки, и передает их в функцию `output()` для обработки в соответствии с приведенным выше описанием. Если этот код немного сложнее для понимания в связи с использованием выражения генератора, можно рассмотреть следующий эквивалентный ему развернутый код:

```
def simpleBS(url, f):
 parsed = BeautifulSoup(f)
 tags = parsed.findAll('a')
 links = [urljoin(url, tag['href']) for tag in tags]
 output(links)
```

С точки зрения удобства для чтения всегда лучше вместо первой однострочной версии применять развернутый вариант. Это особенно рекомендуется при разработке проектов с открытым исходным кодом, а также рабочих или совместно создаваемых проектов.

Функция `simpleBS()` является довольно простой для понимания, но имеет определенные недостатки, в частности, обработка данных с ее помощью не так эффективна, как могла бы быть. Мы используем интерфейс `BeautifulSoup` для интерпретации всех дескрипторов в этом документе и последующего поиска точек привязки. Ускорению работы могло бы способствовать применение фильтра, который выделяет только дескрипторы с точками привязки (и игнорирует остальные).

Именно этот замысел реализован в функции `fasterBS()`. В этой функции выполняются все описанные выше действия. Для этого применяется вспомогательный класс `SoupStrainer` (и передается запрос для фильтрации только дескрипторов точек привязки в качестве параметра `parseOnlyThese`). С помощью класса `SoupStrainer` можно передать в интерфейс `BeautifulSoup` признак того, что элементы, не представляющие интереса при построении дерева синтаксического анализа, должны быть пропущены. Благодаря этому появляется возможность сэкономить не только время, но и память. Кроме того, после завершения разбора в дереве синтаксического анализа остаются только точки привязки, поэтому отпадает необходимость в использовании метода `findAll()` перед выполнением итераций.

## Строки 29–42

В функции `htmlparser()` используется стандартный класс библиотеки `HTMLParser.HTMLParser` для проведения синтаксического анализа. Становится понятно, почему `BeautifulSoup` в качестве средства синтаксического анализа является более популярным; с его помощью можно разработать более короткий и менее сложный код, чем при использовании `HTMLParser`. Кроме того, в том варианте

применения класса `HTMLParser`, который приведен выше, не достигается такая же производительность, поскольку мы вынуждены строить список вручную, т.е. создавать пустой список и повторно вызывать метод `append()` этого класса.

К тому же напрашивается вывод, что класс `HTMLParser` имеет более низкий уровень по сравнению классом с `BeautifulSoup`. Мы создаем подкласс этого класса и обязаны создать метод с именем `handle_starttag()`, который вызывается каждый раз при обнаружении нового дескриптора в файловом потоке (строки 31–39). Мы пропускаем все дескрипторы, отличные от дескрипторов точек привязки (строки 33–34), а затем добавляем все ссылки на точки привязки к переменной `self.data` (строки 37–39). Инициализация переменной `self.data` происходит, как только в этом возникает необходимость (строки 35–36).

Для ввода в действие нового средства синтаксического анализа необходимо создать его экземпляр и передать ему данные (строки 40–41). Как уже было сказано, результаты помещаются в переменную `parser.data`, создаются полные URL, которые затем отображаются (строка 42), как в нашем предыдущем примере `BeautifulSoup`.

## Строки 44–49

В заключительном примере используется класс `html5lib` — средство синтаксического анализа документов HTML, которое соответствует спецификации HTML5. Простейший способ использования `html5lib` состоит в вызове функции `parse()` этого класса и передаче ей полезной нагрузки (строка 47). Функция формирует и выводит дерево в своем собственном формате `simpletree`.

Можно также выбрать для использования любую разновидность среди широко применяемых форматов дерева, включая `minidom`, `ElementTree`, `lxml` или `BeautifulSoup`. Для выбора альтернативного формата дерева достаточно передать функции `parse()` имя желаемого формата в качестве параметра `treebuilder`:

```
import html5lib
f = open("pycon.html")
tree = html5lib.parse(f, treebuilder="lxml")
f.close()
```

Обычно вполне приемлемым является формат `simpletree`, если только не требуется какой-то другой формат дерева. После выполнения пробного прогона и синтаксического анализа образца документа будет получен вывод, который выглядит примерно так:

```
>>> import html5lib
>>> f = open("pycon.html")
>>> tree = html5lib.parse(f)
>>> f.close()
>>> for x in data:
... print x, type(x)
...
<html> <class 'html5lib.treebuilders.simpletree.DocumentType'>
<html> <class 'html5lib.treebuilders.simpletree.Element'>
<head> <class 'html5lib.treebuilders.simpletree.Element'>
<None> <class 'html5lib.treebuilders.simpletree.TextNode'>
<meta> <class 'html5lib.treebuilders.simpletree.Element'>
<None> <class 'html5lib.treebuilders.simpletree.TextNode'>
<title> <class 'html5lib.treebuilders.simpletree.Element'>
<None> <class 'html5lib.treebuilders.simpletree.TextNode'>
```



```

<None> <class 'html5lib.treebuilders.simpletree.CommentNode'>
 . . .
 <class 'html5lib.treebuilders.simpletree.Element'>
<None> <class 'html5lib.treebuilders.simpletree.TextNode'>
<h1> <class 'html5lib.treebuilders.simpletree.Element'>

<a> <class 'html5lib.treebuilders.simpletree.Element'>
<None> <class 'html5lib.treebuilders.simpletree.TextNode'>
<h2> <class 'html5lib.treebuilders.simpletree.Element'>
<None> <class 'html5lib.treebuilders.simpletree.TextNode'>
 . . .

```

В процессе перебора в основном встречаются объекты `TextNode` или `Element`. В рассматриваемом примере нас фактически не интересуют объекты `TextNode`, поскольку задача состоит в извлечении объектов `Element` одного конкретного типа — точек привязки. Для того чтобы отфильтровать именно эти объекты, мы применяем две проверки с помощью конструкций `if` в выражении генератора: отыскиваем только объекты `Element`, а среди этих объектов — только точки привязки (строки 47–49). Обнаружив дескрипторы, соответствующие этим условиям, мы извлекаем атрибуты `"href"`, объединяем их в полный URL и выводим, как и в предыдущей версии (строка 46).

## Строки 51–72

Это приложение приводится в действие функцией `main()`, которая обрабатывает каждую ссылку, найденную в строках 11–14. Выполняется один вызов для загрузки веб-страницы, после чего данные сразу же передаются в объект `StringIO` (строки 65–68), поэтому появляется возможность проводить итерацию по данным с применением каждого из средств синтаксического анализа (строка 69), для чего осуществляется вызов функции `process()`.

Функция `process()` (строки 51–62) принимает целевой URL и объект `StringIO`, а затем вызывает каждое из средств синтаксического анализа, которые выполняют свою задачу и выводят полученные результаты. После каждого успешного разбора (вслед за первым) функция `process()` должна также переустанавливать объект `StringIO` в начало (строки 54, 57 и 60) для применения следующего средства синтаксического анализа.

Завершив подготовку кода и обеспечив его нормальную работу, можно выполнить прогон и проверить, как каждое средство синтаксического анализа выводит все ссылки (отсортированные в алфавитном порядке), заданные с помощью дескрипторов точек привязки в указателе URL веб-страницы. Следует отметить, что ко времени написания этой книги был осуществлен предварительный перенос в Python 3 класса `BeautifulSoup`, но не `html5lib`.

## 9.3.3. Программирование инструментов для просмотра веб-страниц

В заключительном разделе, посвященном веб-клиентам, мы представим немного другой пример, в котором используется инструмент сторонних разработчиков `Mechanize` (основанный на инструменте с тем же именем, написанным для языка Perl), который предназначен для моделирования браузера. Существует также версия этого инструмента для Ruby.

В предыдущем примере (`parse_links.py`) в качестве одного из средств синтаксического анализа применялся класс `BeautifulSoup`, который позволяет расшифровать содержимое веб-страницы. Мы еще раз прибегнем к использованию этого класса в данном примере.

Если читатель желает проверить описанное самостоятельно, то должен установить в своей системе и `Mechanize`, и `BeautifulSoup`. Еще раз отметим, что это программное обеспечение можно получить и установить отдельно или воспользоваться таким инструментом, как `easy_install` или `pip`.

В примере 9.5 представлен сценарий `mech.py`, который в большей степени можно рассматривать как приложение на основе сценария или пакетное приложение. Классы или функции не применяются. Весь сценарий состоит из одной большой функции `main()`, разбитой на семь частей, в каждой из которых рассматривается одна из страниц веб-сайта, который служит источником данных для текущего примера: веб-сайт конференции `PyCon`, проведенной в 2011 году. Мы выбрали этот сайт для данного примера, поскольку он вряд ли изменится со временем (для проводимых впоследствии конференций, по-видимому, будут созданы собственные специализированные приложения).

Даже если этот сайт изменится, есть много других веб-сайтов, для которых можно адаптировать этот пример. В частности, можно указать любую службу электронной почты на основе веб-интерфейса, на которую оформлена подписка, или какие-то часто посещаемые сайты технических новостей или блогов. Изучив сценарий `mech.py` и выполняемые им действия, можно полностью понять, как он работает, и в дальнейшем легко приспособить данный образец кода для работы в любом другом месте.

#### Пример 9.5. Организация просмотра Интернета программным путем (`mech.py`)

Это очень простой сценарий, работа которого во многом напоминает применение пакетов команд. В нем используется инструмент `Mechanize` сторонних разработчиков для исследования веб-сайта конференции `PyCon 2011` и интерпретации этого сайта с помощью еще одного нестандартного инструмента, `BeautifulSoup`.

```

1 #!/usr/bin/env python
2
3 from BeautifulSoup import BeautifulSoup, SoupStrainer
4 from mechanize import Browser
5
6 br = Browser()
7
8 # home page
9 rsp = br.open('http://us.pycon.org/2011/home/')
10 print '\n***', rsp.geturl()
11 print "Confirm home page has 'Log in' link; click it"
12 page = rsp.read()
13 assert 'Log in' in page, 'Log in not in page'
14 rsp = br.follow_link(text_regex='Log in')
15
16 # login page
17 print '\n***', rsp.geturl()
18 print 'Confirm at least a login form; submit invalid creds'
19 assert len(list(br.forms())) > 1, 'no forms on this page'
20 br.select_form(nr=0)
21 br.form['username'] = 'xxx' # wrong login
22 br.form['password'] = 'xxx' # wrong passwd

```

```
23 rsp = br.submit()
24
25 # login page, with error
26 print '\n***', rsp.geturl()
27 print 'Error due to invalid creds; resubmit w/valid creds'
28 assert rsp.geturl() == 'http://us.pycon.org/2011/account/login/', rsp.geturl()
29 page = rsp.read()
30 err = str(BS(page).find("div",
31 {"id": "errorMsg"}).find('ul').find('li').string)
32 assert err == 'The username and/or password you specified are not correct.', err
33 br.select_form(nr=0)
34 br.form['username'] = YOUR_LOGIN
35 br.form['password'] = YOUR_PASSWD
36 rsp = br.submit()
37
38 # login successful, home page redirect
39 print '\n***', rsp.geturl()
40 print 'Logged in properly on home page; click Account link'
41 assert rsp.geturl() == 'http://us.pycon.org/2011/home/', rsp.geturl()
42 page = rsp.read()
43 assert 'Logout' in page, 'Logout not in page'
44 rsp = br.follow_link(text_regex='Account')
45
46 # account page
47 print '\n***', rsp.geturl()
48 print 'Email address parseable on Account page; go back'
49 assert rsp.geturl() == 'http://us.pycon.org/2011/account/email/', rsp.geturl()
50 page = rsp.read()
51 assert 'Email Addresses' in page, 'Missing email addresses'
52 print ' Primary e-mail: %r' % str(
53 BS(page).find('table').find('tr').find('td').find('b').string)
54 rsp = br.back()
55
56 # back to home page
57 print '\n***', rsp.geturl()
58 print 'Back works, on home page again; click Logout link'
59 assert rsp.geturl() == 'http://us.pycon.org/2011/home/', rsp.geturl()
60 rsp = br.follow_link(url_regex='logout')
61
62 # logout page
63 print '\n***', rsp.geturl()
64 print 'Confirm on Logout page and Log in link at the top'
65 assert rsp.geturl() == 'http://us.pycon.org/2011/account/logout/', rsp.geturl()
66 page = rsp.read()
67 assert 'Log in' in page, 'Log in not in page'
68 print '\n*** DONE'
```

---

## Построчное объяснение

### Строки 1–6

Этот сценарий весьма упрощен. Действительно, в нем не применяются какие-либо стандартные библиотечные пакеты или модули, поэтому все, что имеется, — это импорт классов BeautifulSoup.BeautifulSoup и mechanize.Browser.

## **Строки 8–14**

В первую очередь на веб-сайте PyCon 2011 мы посещаем начальную страницу. В качестве подтверждения для пользователя отображается URL этой страницы (строка 10). Следует отметить, что посещение этого URL происходит в самом конце, поскольку первоначальная ссылка может перенаправить пользователя в другое место. Последняя часть этого раздела (строки 12–14) подтверждает, что пользователь не зарегистрирован. Для этого происходит поиск ссылки "Log in" и переход по ней.

## **Строки 16–23**

После подтверждения того, что мы находимся на странице регистрации (на которой имеется не меньше одной формы), происходит выбор первой (и единственной) формы, заполнение полей аутентификации ошибочными данными (мы окажемся не правы, если и имя входа, и пароль имеют значение "xxx") и передача этой формы.

## **Строки 25–36**

После подтверждения ошибки регистрации на странице входа (строки 28–32) мы заполняем поля правильными учетными данными (которые должен задать сам читатель, [YOUR\_LOGIN, YOUR\_PASSWD]) и передаем данные регистрации повторно.

## **Строки 38–44**

После утверждения результатов аутентификации снова происходит перенаправление на начальную страницу. Для подтверждения этого (в строках 41–43) проверяется наличие ссылки "Logout" (которая появляется на странице только в случае успешного входа в систему). Затем необходимо щелкнуть на ссылке "Учетная запись".

## **Строки 46–54**

Для регистрации необходимо использовать адрес электронной почты. Очевидно, что таких адресов может быть несколько, но первичный адрес должен быть единственным. Ваши адреса электронной почты должны быть представлены на первой вкладке, на которую вы перейдете после посещения своей страницы с информацией учетной записи. Мы используем BeautifulSoup для синтаксического анализа и отображения таблицы адресов электронной почты и заполняем первую ячейку первой строки таблицы (строки 52–53). На следующем шаге необходимо щелкнуть на ссылке "click on the back button" (щелкните на кнопке возврата), чтобы возвратиться на начальную страницу.

## **Строки 56–60**

Этот раздел самый короткий из всех. В действительности он предназначен лишь для проверки того, вернулись ли мы на начальную страницу (строка 59), после чего используется ссылка "Logout" для выхода из системы.

## **Строки 62–68**

В последнем разделе мы проверяем, что действительно находимся на странице выхода из системы и не зарегистрированы. Для этого проверяется наличие ссылки "Log in" на этой странице (строки 66–67).

Это приложение демонстрирует, что задача использования класса Mechanize.Browser — довольно несложная. Для работы с этим классом достаточно лишь четко

представлять себе действия пользователя в браузере и связывать их с правильными вызовами методов. В конечном итоге главным источником беспокойства является то, не будут ли внесены изменения в базовую веб-страницу или приложение их разработчиками, поскольку из-за этого наш сценарий может стать непригодным для использования. Следует отметить, что ко времени написания данной книги программное обеспечение Mechanize еще не было перенесено в версию Python 3.

## Резюме

На этом рассмотрение веб-клиентов различных типов в данной главе завершается. Теперь мы можем обратить свое внимание на веб-серверы.

## 9.4. Веб-серверы (HTTP)

Перед этим мы обсуждали использование языка Python для создания веб-клиентов и выполнения задач по упрощению обработки запросов с помощью веб-серверов. Как известно (и как было показано выше в данной главе), Python может использоваться для создания и простых, и сложных веб-клиентов.

Нам еще предстоит рассмотреть создание веб-серверов, и настоящий раздел посвящен именно этой теме. Как известно, наиболее широко применяемыми веб-клиентами являются Google Chrome, Mozilla Firefox, Microsoft Internet Explorer и Opera. Какие веб-серверы являются самыми распространенными? Это Apache, ligHTTPD, Microsoft IIS, LiteSpeed компании LiteSpeed Technologies и httpd компании ACME Laboratories. В условиях конкретного приложения эти серверы могут оказаться слишком громоздкими, поэтому иногда имеет смысл создать простой и вместе с тем удобный веб-сервер с помощью языка Python.

Следует отметить, что серверы Python чрезмерно упрощены и не предназначены для использования в промышленном масштабе, но иногда бывают весьма полезными в качестве серверов для разработчиков. Такие серверы разработки, как Django и Google App Engine, основаны на модуле BaseHTTPServer, описанном в следующем разделе.

### 9.4.1. Простые веб-серверы Python

Весь основной код серверов уже имеется в стандартной библиотеке Python. Достаточно лишь настроить его с учетом конкретных потребностей. Для создания веб-сервера требуются базовый сервер и *обработчик*.

Базовый веб-сервер — это стандартный шаблон со всеми обязательными функциями. Его назначение состоит в обеспечении связи по протоколу HTTP между клиентом и сервером. Базовый класс сервера (как и следовало ожидать) имеет имя HTTPServer и находится в модуле BaseHTTPServer.

Обработчик — это часть программного обеспечения, которая выполняет основную работу по предоставлению веб-страниц. Он обрабатывает клиентские запросы и возвращает требуемые файлы, которые могут быть либо статическими, либо формируемыми динамически. Сложность обработчика определяет сложность веб-сервера. В стандартной библиотеке Python предусмотрены три разных обработчика.

Самым основным, простым, стандартным обработчиком является BaseHTTPRequestHandler, который находится в модуле BaseHTTPServer, где также

имеется код базового веб-сервера. Этот обработчик не выполняет какой-либо иной обработки, кроме получения клиентского запроса, поэтому все прочие необходимые действия разработчик должен реализовывать самостоятельно, как в приведенном ниже примере сервера `myhttpd.py`.

Обработчик `SimpleHTTPRequestHandler`, который находится в модуле `SimpleHTTPServer`, основан на обработчике `BaseHTTPRequestHandler` и реализует довольно несложную обработку стандартных запросов `GET` и `HEAD`. Этот обработчик еще не позволяет реализовывать какие-либо сложные функции, но с его помощью можно выполнять простые задания.

Наконец, в модуле `CGIHTTPServer` предусмотрен обработчик `CGIHTTPRequestHandler`, который основан на обработчике `SimpleHTTPRequestHandler` и вводит поддержку запросов `POST`. Он обладает способностью вызывать стандартные сценарии `CGI` (`Common Gateway Interface` — интерфейс общего шлюза) для выполнения необходимой обработки, а также позволяет формировать код `HTML` и отправлять его клиенту. В этой главе рассматриваются только серверы, обеспечивающие обработку на основе стандарта `CGI`, а в следующей главе будет показано, почему сфера применения технологии `CGI` в Интернете теперь значительно сузилась, но разработчики по-прежнему должны изучать понятия, лежащие в ее основе.

## 3.x

Для упрощения взаимодействия с пользователем, достижения единообразия и улучшения сопровождения кода эти модули (фактически классы этих модулей) объединены в общий модуль с именем `server.py` и устанавливаются в составе пакета `http` в Python 3. (Аналогичным образом модуль `httplib` в версии Python 2, клиент `HTTP`, был переименован в `http.client` в Python 3.) Общие сведения об этих трех модулях, их классах и охватывающем пакете `http.server` в Python 3 приведены в табл. 9.6.

Таблица 9.6. Модули и классы веб-серверов

Модуль	Описание
<code>BaseHTTPServer<sup>a</sup></code>	Предоставляет основной веб-сервер и основные классы обработчиков, <code>HTTPServer</code> и <code>BaseHTTPRequestHandler</code> соответственно
<code>SimpleHTTPServer<sup>a</sup></code>	Содержит класс <code>SimpleHTTPRequestHandler</code> , предназначенный для выполнения запросов <code>GET</code> и <code>HEAD</code>
<code>CGIHTTPServer<sup>a</sup></code>	Содержит класс <code>CGIHTTPRequestHandler</code> , который обеспечивает обработку запросов <code>POST</code> и выполнение функций <code>CGI</code>
<code>http.server<sup>b</sup></code>	Все три указанные выше класса и модули Python 2 объединены в Python 3 в один пакет

<sup>a</sup> Удалено в Python 3.0.

<sup>b</sup> Новое в Python 3.0.

## Реализация простого базового веб-сервера

Чтобы проще было понять, как работают более сложные обработчики, находящиеся в модулях `SimpleHTTPServer` и `CGIHTTPServer`, реализуем простую обработку `GET` для класса `BaseHTTPRequestHandler`. В примере 9.6 представлен код полностью работоспособного веб-сервера, `myhttpd.py`.

**Пример 9.6. Простой веб-сервер (myhttpd.py)**

Этот простой веб-сервер может читать запросы GET, получать веб-страницу (файл .html) и возвращать ее вызывающему клиенту. В нем используется обработчик BaseHTTPRequestHandler, который находится в модуле BaseHTTPServer, и реализован метод do\_GET() для обработки запросов GET.

```

1 #!/usr/bin/env python
2
3 from BaseHTTPServer import \
4 BaseHTTPRequestHandler, HTTPServer
5
6 class MyHandler(BaseHTTPRequestHandler):
7 def do_GET(self):
8 try:
9 f = open(self.path[1:], 'r')
10 self.send_response(200)
11 self.send_header('Content-type', 'text/html')
12 self.end_headers()
13 self.wfile.write(f.read())
14 f.close()
15 except IOError:
16 self.send_error(404,
17 'File Not Found: %s' % self.path)
18
19 def main():
20 try:
21 server = HTTPServer(('', 80), MyHandler)
22 print 'Welcome to the machine...'
23 print 'Press ^C once or twice to quit.'
24 server.serve_forever()
25 except KeyboardInterrupt:
26 print '^C received, shutting down server'
27 server.socket.close()
28
29 if __name__ == '__main__':
30 main()

```

Этот сервер основан на обработчике BaseHTTPRequestHandler и состоит из одного метода do\_GET() (строки 6-7), который вызывается при получении запроса GET базовым сервером. Предпринимается попытка открыть путь (после удаления ведущего знака "/"), переданный клиентом (строка 9), и в случае успеха возвращается код статуса OK (200), а загруженная веб-страница перенаправляется пользователю (строка 13) через канал wfile. Если файл не найден, происходит возврат кода статуса 404 (строки 15-17).

В функции main() просто порождается экземпляр класса веб-сервера и происходит его вызов для запуска (как и в предыдущих примерах) бесконечного цикла сервера. Цикл прерывается после нажатия <Ctrl+C> или аналогичной комбинации клавиш. Если у вас есть соответствующий доступ и вы можете вызвать этот сервер на выполнение, то обнаружите, что он формирует вывод, предназначенный для записи в журнал, который выглядит примерно так:

```

myhttpd.py
Welcome to the machine... Press ^C once or twice to quit

```

```
localhost - - [26/Aug/2000 03:01:35] "GET /index.html HTTP/1.0" 200 -
localhost - - [26/Aug/2000 03:01:29] code 404, message File Not Found: x.html
localhost - - [26/Aug/2000 03:01:29] "GET /dummy.html HTTP/1.0" 404 -
localhost - - [26/Aug/2000 03:02:03] "GET /hotlist.htm HTTP/1.0" 200 -
```

Разумеется, этот простой и небольшой веб-сервер настолько примитивен, что не может даже обрабатывать обычные текстовые файлы. Решение этой задачи оставляем читателю в качестве упражнения (см. упражнение 9.10 в конце этой главы).

## Больше возможностей и меньше кода — простой веб-сервер CGI

Недостатком предыдущего примера является также то, что он не позволяет обрабатывать запросы CGI. Модуль `BaseHTTPServer` действительно оправдывает свое название базового. Поднявшись на одну ступень выше, мы получаем модуль `SimpleHTTPServer`. В нем от имени пользователя реализованы методы `do_HEAD()` и `do_GET()`, поэтому отпадает необходимость их создавать, в отличие от `BaseHTTPServer`.

Сервером наивысшего уровня (воспринимайте это утверждение с долей иронии) в стандартной библиотеке является `CGIHTTPServer`. В дополнение к методам `do_HEAD()` и `do_GET()` в нем определен метод `do_POST()`, который позволяет обрабатывать данные формы. Это настолько удобно, что сервер разработки, поддерживающий технологию CGI, можно создать с применением буквально двух строк кода. Это настолько малый фрагмент кода, что было решено даже не приводить его пример в данной главе (читатель может сразу же проверить его работу, введя следующий код в своем компьютере):

```
#!/usr/bin/env python
import CGIHTTPServer
CGIHTTPServer.test()
```

Необходимо отметить, что мы исключили проверку условий завершения работы сервера с помощью комбинации клавиш `<Ctrl+C>`, а также возможности привлекательного оформления вывода, воспользовавшись лишь тем, что предоставляет нам функция `CGIHTTPServer.test()`. Это уже немало. Для запуска сервера достаточно вызвать его из своего командного интерпретатора. Ниже приведен пример выполнения кода на ПК. При проведении экспериментов на компьютере POSIX будут получены весьма похожие результаты:

```
C:\py>python cgihttpd.py
Serving HTTP on 0.0.0.0 port 8000 ...
```

В этом примере выполняется запуск сервера по умолчанию в порту 8000, но можно изменить это значение во время вызова команды, задавая номер порта в качестве параметра командной строки:

```
C:\py>python cgihttpd.py 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

Для проверки достаточно убедиться, что папка `cgi-bin` существует (эта папка требуется для некоторых сценариев Python CGI) и находится на том же уровне каталогов, что и каталог со сценарием. Нет смысла устанавливать Apache, задавать префиксы обработчика CGI и выполнять все прочие настройки, если требуется лишь одно — проверить простой сценарий. Сведения о том, как писать сценарии CGI, приведены



в главе 10. В этой главе также показано, по каким причинам следует отказаться от использования технологии CGI.

Вполне очевидно, что задача запуска веб-сервера и обеспечение его функционирования с применением исключительно средств Python является довольно несложной. Еще раз отметим, что не нужно ставить перед собой цель самому разрабатывать код для всех применяемых серверов. Как правило, следует заниматься лишь созданием веб-приложений, которые работают на веб-серверах. Описанные здесь серверные модули предназначены исключительно для создания серверов, применяемых во время разработки, независимо от того, ведется ли программирование приложений или веб-платформ.

На производстве необходимо вместо этого применять только серверы производственного назначения, такие как Apache, ligHTTPD, или любые из тех, что были перечислены в начале этого раздела. Однако мы надеемся, что этот раздел позволяет представить себе, насколько могут быть упрощены даже самые сложные задачи благодаря использованию тех возможностей, которые предоставляет язык Python.

## 9.5. Связанные модули

В табл. 9.7 представлен список модулей, которые могут применяться при разработке веб-приложений. Определенная часть этих модулей рассматривалась в настоящей главе.

**Таблица 9.7.** Модули, применяемые при разработке веб-приложений

Модуль/пакет	Описание
<b>Веб-приложения</b>	
cgi	Получает данные формы CGI
cgitb <sup>c</sup>	Обработывает трассировки CGI
htmllib	Выпущенное ранее средство синтаксического анализа HTML для простых файлов HTML; класс HTML-Parser является производным от sgmlib.SGMLParser
HTMLparser <sup>c</sup>	Более новое средство синтаксического анализа для HTML и XHTML, не основанное на SGML
htmlentitydefs	Общие определения сущностей HTML
Cookie	Серверные cookie-файлы для управления состоянием HTTP
cookielib <sup>c</sup>	Классы обработки cookie-файлов для клиентов HTTP
webbrowser <sup>b</sup>	Контроллер: запускает обработку веб-документов в браузере
sgmlib	Применяется для разбора простых файлов SGML
robotparser <sup>a</sup>	Выполняет разбор файлов robots.txt для анализа доступности URL для выборки
httplib <sup>a</sup>	Используется для создания клиентов HTTP
urllib	Обеспечивает доступ к серверам через URL и с помощью других программ, связанных с URL; метод urllib.urlopen() заменен методом urllib2.urlopen() в Python 3 под именем urllib.request.urlopen()
urllib2; urllib.request <sup>g</sup> , urllib.error <sup>g</sup>	Классы и функции для открытия (практически применяемых) URL; в Python 3 предусмотрено разбиение еще на два подпакета
urlparse, urllib.parse <sup>g</sup>	Программы для интерпретации строк URL; соответствующая библиотека в Python 3 переименована в urllib.parse

Модуль/пакет	Описание
<b>Обработка XML</b>	
<code>xmllib</code>	Исходное несложное средство синтаксического анализа XML (рассматривается как устаревшее)
<code>xml<sup>b</sup></code>	Пакет XML, в котором предусмотрены различные средства синтаксического анализа (некоторые из них перечислены ниже)
<code>xml.sax<sup>b</sup></code>	Простой API-интерфейс для средства синтаксического анализа XML (SAX), совместимое с SAX2
<code>xml.dom<sup>b</sup></code>	Средство синтаксического анализа XML на основе DOM (Document Object Model — объектная модель документа)
<code>xml.etree<sup>f</sup></code>	Ориентированное на создание дерева средство синтаксического анализа XML на основе гибкого контейнерного объекта Element
<code>xml.parsers.expat<sup>b</sup></code>	Интерфейс для средства синтаксического анализа XML Expat, не предусматривающего проверку допустимости
<code>xmllrpc<sup>c</sup></code>	Поддержка клиента для удаленного вызова процедур XML (RPC) по протоколу HTTP
<code>SimpleXMLRPCServer<sup>c</sup></code>	Основная платформа для серверов XML-RPC на языке Python
<code>DocXMLRPCServer<sup>d</sup></code>	Платформа для самодокументирующих серверов XML-RPC
<b>Веб-серверы</b>	
<code>BaseHTTPServer</code>	Абстрактный класс, с помощью которого ведется разработка веб-серверов
<code>SimpleHTTPServer</code>	Средства обслуживания самых простых запросов HTTP (HEAD и GET)
<code>CGIHTTPServer</code>	Кроме обработки таких веб-файлов, как при использовании <code>SimpleHTTPServer</code> , позволяет также обрабатывать запросы CGI (HTTP POST)
<code>http.server<sup>g</sup></code>	Новое имя комбинированного пакета Python 3, в котором объединены модули <code>BaseHTTPServer</code> , <code>SimpleHTTPServer</code> и <code>CGIHTTPServer</code>
<code>wsgiref<sup>f</sup></code>	Пакет, определяющий стандартный интерфейс между веб-серверами и веб-приложениями
<b>Пакеты сторонних разработчиков (не относящиеся к стандартной библиотеке)</b>	
<code>HTMLgen</code>	Вспомогательный класс для CGI, преобразующий объекты Python в компоненты с допустимым кодом HTML <a href="http://starship.python.net/crew/friedrich/HTMLgen/html/main.html">http://starship.python.net/crew/friedrich/HTMLgen/html/main.html</a>
<code>BeautifulSoup</code>	Средство синтаксического анализа HTML и XML, а также программа чтения содержимого экрана <a href="http://crummy.com/software/BeautifulSoup">http://crummy.com/software/BeautifulSoup</a>
<code>Mechanize</code>	Пакет для веб-просмотра на основе <code>WWW:Mechanize</code> <a href="http://wwwsearch.sourceforge.net/mechanize/">http://wwwsearch.sourceforge.net/mechanize/</a>

<sup>a</sup> Новое в Python 1.6.<sup>b</sup> Новое в Python 2.0.<sup>c</sup> Новое в Python 2.2.<sup>d</sup> Новое в Python 2.3.<sup>e</sup> Новое в Python 2.4.<sup>f</sup> Новое в Python 2.5.<sup>g</sup> Новое в Python 3.0.

## 9.6. Упражнения

- 9.1. *Модуль urllib*. Напишите программу, которая принимает URL, введенный пользователем (который указывает на веб-страницу или на файл FTP, например `http://python.org` или `ftp://ftp.python.org/pub/python/README`), и загружает указанный ресурс на компьютер с тем же именем файла (или измененным именем, аналогичным указанному первоначально, если имя в таком формате не может применяться в конкретной системе). Веб-страницы (HTTP) следует сохранять как файлы с расширением `.htm` или `.html`, а файлы FTP должны оставаться с исходным расширением.
- 9.2. *Модуль urllib*. Перепишите сценарий `grabWeb.py` из примера 11.04 в книге *Core Python Programming* или *Core Python Language Fundamentals*, который загружает веб-страницу и отображает первую и последнюю непустые строки полученного файла HTML, чтобы можно было использовать функцию `urlopen()` вместо `urlretrieve()` для непосредственной обработки данных (а не загружать весь файл до начала его обработки).
- 9.3. *URL и регулярные выражения*. Предположим, что применяемый браузер позволяет сохранять URL избранных веб-сайтов в виде HTML-файла с закладками (это обеспечивают браузеры на основе Mozilla) или в виде ряда файлов с расширением `.url` в каталоге `favorites` (это обеспечивает Internet Explorer). Определите, какой метод предусмотрен в этом браузере для записи активных ссылок и регистрации данных по местонахождению (где и как они хранятся). Не внося изменения ни в один из файлов, исключите URL и имена соответствующих веб-сайтов (если они заданы) и сформируйте состоящий из двух столбцов список с именами и ссылками в качестве вывода, а затем сохраните эти данные в файле на диске. Усекайте имена сайтов или URL, чтобы длина каждой строки вывода не превышала 80 символов.
- 9.4. *URL, модуль urllib, исключения и регулярные выражения*. В качестве дополнения к упражнению 9.3 добавьте к своему сценарию код для проверки каждой из ваших избранных ссылок. Составьте в виде отчета список действующих ссылок (с именами). Под этим подразумеваются веб-сайты, которые больше не активны, или удаленные веб-страницы. Ссылки, которые все еще остаются действующими, просто выводите на экран и сохраняйте на диске.

Приведенные ниже упражнения 9.5–9.8 относятся к файлам журналов доступа веб-сервера и регулярным выражениям. Веб-серверы (и их администраторы), как правило, должны обеспечивать ведение файлов журнала доступа (обычно `logs/access_log` в основном каталоге веб-сервера), в которых отслеживаются запросы. Со временем эти файлы приобретают слишком большой объем и требуют либо сохранения на отдельном носителе, либо усечения. Возможно, имеет смысл извлекать из этих файлов только значимую информацию, после чего удалять их для экономии места на диске. Приведенные ниже упражнения позволяют приобрести определенный опыт работы с регулярными выражениями и узнать, как можно использовать их для архивирования и анализа данных веб-сервера.

- 9.5. Подсчитайте, какое количество запросов каждого типа (GET и POST) зарегистрировано в файле журнала.

- 9.6. Подсчитайте количество успешных загрузок страниц/данных. Выведите на экран все ссылки, при обработке которых был получен код возврата 200 (ОК, ошибки не найдено), с указанием количества обращений к каждой ссылке.
- 9.7. *Подсчет ошибок.* Выведите на экран все ссылки, при обработке которых возникли ошибки (коды возврата, которые начинаются с 400 или 500), с указанием количества обращений к каждой ссылке.
- 9.8. *Отслеживание IP-адресов.* Для каждого IP-адреса выведите список со всеми операциями загрузки страниц/данных с указанием количества обращений к каждой ссылке.
- 9.9. *Cookie-файлы веб-браузера и регистрация веб-сайта.* В некоторых главах (7, 9, 13) книг *Core Python Programming* и *Core Python Language Fundamentals* было показано, как работать с базой данных регистрации имен входа пользователей. Для этого создавался чисто текстовый сценарий, управляемый с помощью меню. Перенесите этот сценарий на веб-страницу, чтобы теперь информация о пользователях и паролях обрабатывалась в системе проверки аутентичности сайта.

**Дополнительное задание.** Ознакомьтесь с тем, как формируются cookie-файлы веб-браузера, и поддерживайте сеанс работы на сайте в течение четырех часов после последней успешной регистрации.

- 9.10. *Создание веб-серверов.* Разработанный нами код сценария `myhttpd.py` (пример 9.6) позволяет только читать файлы HTML и возвращать их вызывающему клиенту. Добавьте поддержку для простых текстовых файлов с расширением `.txt`. Обеспечьте возврат правильного типа MIME, а именно `text/plain`.

**Дополнительное задание.** Добавьте поддержку для файлов JPEG с расширением `.jpg` или `.jpeg`, с применением типа MIME `image/jpeg`.

Упражнения 9.11–9.14 требуют обновления кода поискового робота, `crawl.py`, который рассматривался в примере 9.3.

- 9.11. *Веб-клиенты.* Внесите изменения в сценарий `crawl.py`, чтобы в нем использовались другие системы интерпретации, `HTMLParser`, `BeautifulSoup`, `html5lib` или `lxml`.
- 9.12. *Веб-клиенты.* URL, предоставляемые в качестве входа для `crawl.py`, должны иметь ведущий признак протокола `http://`, а URL верхнего уровня должны содержать заключительную косую черту, например `http://www.prenhallprofessional.com/`. Обеспечьте более надежную работу сценария `crawl.py`, разрешив пользователю вводить только имя хоста (без части с обозначением протокола, в предположении, что это HTTP), а также снимите требование по обязательному применению заключительной косой черты. Например, вполне допустимым должен стать ввод `www.prenhallprofessional.com`.
- 9.13. *Веб-клиенты.* Обновите сценарий `crawl.py`, чтобы он также загружал ссылки, в которых используется префикс `ftp:`. Все ссылки `mailto:` игнорируются сценарием `crawl.py`. Внесите изменения, чтобы этот сценарий исключал из рассмотрения также `telnet:`, `news:`, `gopher:` и `about:` и подобные ссылки.

- 9.14. Веб-клиенты.** Сценарий `crawl.py` загружает только файлы `.html` с помощью ссылок, которые указывают на веб-страницы того же сайта, не обрабатывая и не сохраняя изображения, которые также являются действительными файлами на этих страницах. Кроме того, этот сценарий не позволяет работать с серверами, не воспринимающими URL, в которых отсутствует заключительная косая черта (`/`). Введите в сценарий `crawl.py` такие классы, которые позволили бы устранить эти недостатки.

Класс `My404UrlOpener` должен представлять собой подкласс `urllib.FancyURLOpener` и состоять из отдельного метода, `http_error_404()`, который определяет, возникала ли ошибка 404 в связи с применением URL без заключительной косой черты. В таком случае добавляется косая черта и запрос повторяется (но только один раз). Если вторая попытка также оканчивается неудачей, происходит безусловный возврат ошибки 404. Необходимо задать `urllib._urlopener` с экземпляром этого класса, чтобы обеспечить его применение в `urllib`.

Создайте еще один класс с именем `LinkImageParser`, который происходит от `htmllib.HTMLParser`. Этот класс должен содержать конструктор для вызова конструктора базового класса, а также инициализировать список с файлами изображений, полученными в результате синтаксического анализа из веб-страниц. Необходимо перекрыть метод `handle_image()`, чтобы обеспечить добавление имен файлов изображений к списку изображений (а не отбрасывать их, как в текущем методе базового класса).

Последний ряд упражнений относится к файлу `parse_links.py`, который рассматривался ранее в этой главе в примере 9.4.

- 9.15. Параметры командной строки.** Добавьте параметры командной строки, которые позволили бы пользователю просматривать вывод одного или нескольких средств синтаксического анализа (а не той части из них, которые применяются по умолчанию).
- 9.16. Средство синтаксического анализа `lxml`.** Загрузите и установите модуль `lxml`, а затем добавьте поддержку для `lxml` к `parse_links.py`.
- 9.17. Средства синтаксического анализа разметки.** Внесите изменения в каждое средство синтаксического анализа в поисковом роботе, заменив `htmllib.HTMLParser`.
- а) `HTMLParser.HTMLParser`
  - б) `html5lib`
  - в) `BeautifulSoup`
  - г) `lxml`
- 9.18. Применение рефакторинга.** Внесите изменения в функцию `output()`, чтобы она позволяла поддерживать другие формы вывода.
- а) Запись в файл
  - б) Отправка в другой процесс (т.е. запись в сокет)

- 9.19.** *Разработка кода в стиле Python.* В построчном объяснении сценария `parse_links.py` было показано, как развернуть код функции `simpleBS()`, чтобы он превратился из сложной для восприятия однострочной команды в блок отформатированного должным образом кода Python. Прделайте то же с функциями `fasterBS()` и `html5libparse()`.
- 9.20.** *Производительность и профилирование.* Выше было показано, почему быстроедействие `fasterBS()` выше по сравнению с `simpleBS()`. Воспользуйтесь функцией `timeit` для демонстрации разности в быстродействии, затем найдите в Интернете инструмент для работы с памятью Python, который показал бы, какая функция требует меньше памяти. Опишите, что это за инструмент профилировщика памяти и где вы его нашли. Показывает ли какой-либо из трех стандартных библиотечных профилировщиков (`profile`, `hotshot`, `cProfile`) информацию об использовании памяти?
- 9.21.** *Рекомендации.* Применительно к функции `htmlparser()` предположим, что решено отказаться от идеи создания пустого списка и повторного вызова метода `append()` для пополнения списка. Вместо этого должен использоваться способ непосредственного формирования списка, который позволяет заменить строки 35–39 следующей строкой кода:

```
self.data = [v for k, v in attrs if k == 'href']
```

Допустима ли такая замена? Иными словами, можем ли мы внести это изменение и по-прежнему рассчитывать на то, что сценарий будет продолжать работать правильно? Почему его следует (или не следует) использовать?

- 9.22.** *Манипулирование данными.* В сценарии `parse_links.py` применяется сортировка URL в алфавитном порядке (фактически в лексикографическом порядке). Но этот способ организации ссылок может оказаться не самым лучшим:

```
http://python.org/psf/
http://python.org/search
http://roundup.sourceforge.net/
http://sourceforge.net/projects/mysql-python
http://twistedmatrix.com/trac/
http://wiki.python.org/moin/
http://wiki.python.org/moin/CgiScripts
http://www.python.org/
```

Вместо этого в большей степени оправдано применение сортировки по именам доменов:

```
http://python.org/psf/
http://python.org/search
http://wiki.python.org/moin/
http://wiki.python.org/moin/CgiScripts
http://www.python.org/
http://roundup.sourceforge.net/
http://sourceforge.net/projects/mysql-python
http://twistedmatrix.com/trac/
```

Предусмотрите в своем сценарии возможность проводить сортировку не только в алфавитном (лексикографическом) порядке, но и по именам доменов.

# Веб-программирование: интерфейсы CGI и WSGI

*В этой главе...*

- Введение
- Средства обработки клиентских данных на веб-сервере
- Создание приложений CGI
- Использование стандарта кодирования Юникод с интерфейсом CGI
- Расширения CGI
- Введение в WSGI
- Практическая разработка для веб
- Связанные модули

*Интерфейс WSGI прежде всего предназначен для разработчиков интернет-платформ и веб-серверов, а не веб-приложений. Это — не API приложений, а API сопряжений платформы с сервером.*  
Филип Дж. Эби (Phillip J. Eby),  
август 2004 года

## 10.1. Введение

Это вступительная глава по программированию для веб, в которой содержится краткий и общий обзор возможностей веб-программирования в языке Python, начиная от навигации и заканчивая созданием форм отзывать пользователей, распознаванием указателей URL и разработкой динамически формируемых веб-страниц. В этой главе вначале рассматривается *интерфейс общего шлюза* (Common Gateway Interface — CGI), а затем описывается *интерфейс шлюза веб-сервера* (Web Server Gateway Interface — WSGI).

## 10.2. Средства обработки клиентских данных на веб-сервере

В разделе приводятся общие сведения об интерфейсе CGI, в частности, объясняется, что означает аббревиатура CGI, для чего предназначен данный интерфейс и как он применяется на веб-серверах. После этого демонстрируется использование языка Python для создания приложений CGI.

### 10.2.1. Введение в интерфейс CGI

Изначально веб разрабатывался как глобальный оперативный репозиторий, или архив документов (главным образом для образовательных и научных учреждений). При этом информация в основном была представлена в текстовой форме, как правило, с разметкой HTML.

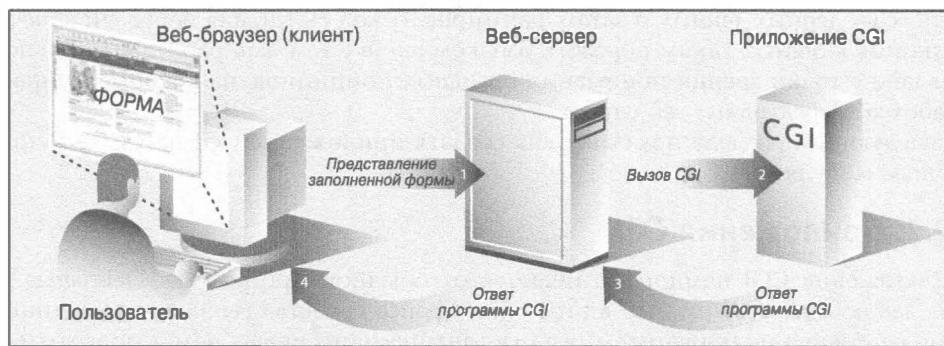
Язык HTML можно рассматривать как инструмент форматирования текста, позволяющий задавать шрифты, размеры и стили. Основной особенностью HTML является поддержка гипертекста. Под этим подразумевается возможность использовать определенный текст (обычно как-то выделенный) или даже графические элементы в качестве ссылок, указывающих на другие документы или на места в Интернете и в веб, которые связаны с контекстом оригинала. Для доступа к документу достаточно щелкнуть кнопкой мыши или воспользоваться другим механизмом выделения содержимого, предназначенным для пользователей. Документы HTML являются статическими, хранятся на веб-сервере и передаются клиентам по требованию.

По мере развития служб, поддерживающих Интернет и веб-службы, возрастала потребность в обработке данных, вводимых пользователями. Владельцам интернет-магазинов необходимо было обрабатывать отдельные заказы, а в приложениях интернет-банкинга и поисковых машинах все чаще стали применяться учетные записи, относящиеся к конкретным пользователям. В связи с этим были созданы заполняемые формы; они представляли собой единственное средство, с помощью которого веб-сайт мог получать информацию от пользователей (пока не появились апплеты Java). Это, в свою очередь, потребовало разработки средств динамического генерирования кода HTML для каждого клиента, передающего данные.



В действительности веб-серверы легко решают только одну задачу: получение пользовательского запроса на выдачу файла и возврат этого файла (т.е. HTML-файла) клиенту. Веб-серверы не обладают таким “интеллектом”, который позволял бы обрабатывать данные, связанные с конкретными пользователями, в частности, поля заполняемых форм. По этой причине веб-серверы передают полученные запросы во внешние приложения, в которых создается динамически генерируемый код HTML, возвращаемый клиенту.

Весь процесс начинается с того, что веб-сервер получает клиентский запрос (GET или POST) и вызывает соответствующее приложение. Затем сервер ожидает поступления результата в виде кода HTML для последующей передачи клиенту. После завершения своей работы приложение передает динамически сгенерированный код HTML на сервер, который, в свою очередь, возвращает его пользователю. Процесс получения формы, передачи данных внешнему приложению, получения и возврата кода HTML осуществляется с помощью интерфейса CGI. Общая схема работы интерфейса CGI представлена на рис. 10.1. На этом рисунке поэтапно показаны все выполняемые операции и поток данных, начиная от отправки пользователем формы и заканчивая возвратом результирующей веб-страницы.



**Рис. 10.1.** Общая схема работы интерфейса CGI, обеспечивающего взаимодействие веб-сервера и приложения, необходимое для обработки формы пользователя и подготовки возвращаемого в конечном итоге динамического кода HTML

Данные, введенные в форму с помощью клиентской программы и отправленные на веб-сервер, могут требовать обработки, а также, возможно, применения той или иной системы хранения данных в серверной базе данных. Следует помнить, что если веб-страница содержит элементы, требующие ввода данных пользователем (текстовые поля, переключатели и т.д.), а также кнопку для отправки данных или изображение, то по всей вероятности для обработки этой формы необходимо предусмотреть выполнение тех или иных операций CGI.

Приложения CGI, в которых формируется код HTML, обычно разрабатываются на одном из многих высокоуровневых языков программирования, позволяющих принимать данные пользователя, а затем обрабатывать их и возвращать код HTML на сервер. Прежде чем приступить к описанию CGI, мы должны подчеркнуть, что веб-приложения производственного назначения, как правило, больше не создаются с применением этого интерфейса.

Интерфейс CGI имеет существенные ограничения и не может применяться на веб-серверах для одновременной обработки большого количества клиентских

соединений, поэтому рассматривается как устаревший. При создании веб-служб, на которые возложены ответственные задачи, принято применять компилируемые языки наподобие C/C++, которые обеспечивают необходимое масштабирование. Окружение современного веб-сервера, как правило, состоит из сервера Apache, встроенных компонентов для доступа к базе данных (MySQL или PostgreSQL), модулей поддержки языков Java (Tomcat), PHP и других интерпретируемых языков, таких как Python или Ruby, а также средств безопасности, например, на основе протокола SSL. Разработку небольшого веб-сайта для себя лично или для какой-либо организации, где не требуются значительные мощь и сложность, связанные с созданием ответственных веб-служб, вполне можно начать с применения CGI. Этот интерфейс может также использоваться для проверки замысла будущего веб-сайта.

Тем не менее в настоящее время для создания веб-служб предусмотрено большое количество разнообразных платформ разработки веб-приложений и систем управления информационным наполнением, поэтому CGI окончательно становится пережитком прошлого. Вместе с тем, даже в самой современной и развитой среде разработки, используется та же модель, которая была первоначально предусмотрена в CGI; любая веб-служба должна получать данные, введенные пользователем, выполнять код на основе введенных данных, а затем формировать код HTML для передачи конечных результатов клиенту. Таким образом, ознакомление с тем, как работает CGI, вполне оправдано с точки зрения понимания основных принципов, на которые опирается разработка эффективных веб-служб.

В следующем разделе показано, как создать приложение CGI на языке Python с помощью модуля `cgi`.

## 10.2.2. Приложения CGI

Приложение CGI немного отличается от обычной программы. Основные различия заключаются в том, что в этом интерфейсе средства ввода-вывода данных и взаимодействие с пользователем реализованы немного иначе, чем в программе, не предназначенной для работы в сети. После запуска сценария CGI веб-сервер получает данные, введенные пользователем, но они поступают от веб-клиента, а не с локального компьютера или из файла на диске. Это процесс называется *запросом* (request).

В отличие от обычного приложения, при выводе приложение CGI передает данные веб-клиенту, а не на экран, в окно графического пользовательского интерфейса или в файл на диске. Данные, передаваемые сервером клиенту, называются *ответом* (response). Эти данные должны быть снабжены правильно оформленными заголовками, за которыми следует текст с разметкой HTML. Иначе, если веб-клиентом является браузер, то возникнет внутренняя ошибка сервера (Internal Server Error), потому что веб-клиенты понимают только корректные данные HTTP (например, заголовки MIME и код HTML).

Наконец, следует отметить, что непосредственно в самом сценарии взаимодействия с пользователем не предусмотрено. Весь обмен данными происходит между веб-клиентом, действующим от имени пользователя, веб-сервером и приложением CGI.

## 10.2.3. Модуль `cgi`

В модуле `cgi` предусмотрен основной класс `FieldStorage`, который выполняет всю работу. Этот класс считывает всю необходимую информацию о пользователе,

передаваемую веб-клиентом (через веб-сервер), поэтому экземпляр этого класса должен быть создан сразу после запуска сценария CGI на языке Python. Экземпляр указанного класса включает объект, напоминающий словарь, который содержит набор пар “ключ–значение”. Ключами являются имена элементов ввода, передаваемых из заполненной формы. Значения содержат соответствующие данные.

Значения могут представлять собой объекты одного из трех типов. Во-первых, экземпляры класса `FieldStorage`. Во-вторых, экземпляры аналогичного класса `MiniFieldStorage`, который используется в тех случаях, если не требуется передача (upload) файлов или ведется обработка данных формы, не состоящей из нескольких частей. Экземпляры `MiniFieldStorage` содержат только пары “ключ–значение”, состоящие из имени и данных. В-третьих, передаваемые значения могут представлять собой список указанных объектов. Такие списки формируются, если форма содержит несколько элементов ввода с одинаковым именем поля.

Для простых веб-форм обычно достаточно применять лишь экземпляры `MiniFieldStorage`. Все приведенные ниже примеры относятся только к этому общему случаю.

### 10.2.4. Модуль `cgitb`

Как было указано ранее, веб-приложение должно вернуть веб-серверу для дальнейшей передачи пользователю или браузеру правильно сформированный ответ, содержащий допустимые заголовки HTTP и данные в формате HTML. В случае аварийного завершения работы приложения CGI ни о каком возврате правильно оформленных данных не может быть и речи. Происходит то же, что и при выполнении любого сценария Python, который завершается с ошибками: выводится сообщение об ошибке и формируется протокол обратной трассировки. Может ли текст обратной трассировки представлять собой правильно оформленные заголовки HTTP или код HTML? Разумеется, нет.

Веб-сервер, получивший ответ, который не может быть им распознан, не принимает никаких усилий по исправлению такой ситуации и просто возвращает код 500, которая означает, что произошла внутренняя ошибка веб-сервера, скорее всего связанная с выполнением текущего приложения. На основании этого кода разработчик вряд ли сможет определить, что произошло, и ему не может помочь даже вывод в браузере, поскольку обычно экран остается пустым или содержит сообщение “Internal Server Error” или нечто подобное.

Если программа на языке Python выполняется в командной строке или в *интегрированной среде разработки*, то ошибки приводят к получению обратной трассировки, которую можно изучить и найти причины сбоя. В браузере все происходит иначе, поэтому необходимо вывести на экран браузера полный набор сообщений обратной трассировки веб-приложения, а не одно лишь указание на то, что произошла внутренняя ошибка сервера. Именно для этого предназначен модуль `cgitb`.

Для того чтобы получить вывод обратной трассировки, достаточно предусмотреть в сценарии приложения CGI следующие инструкции импорта модуля и вызова метода:

```
import cgitb
cgitb.enable()
```

Читатель получит много шансов исследовать возможности CGI в первой половине главы. Пока отложим дальнейший анализ этих двух строк, поскольку первые

несколько примеров для этого слишком просты. Вначале рассмотрим более сложный метод устранения ошибок, связанных с появлением сообщения “Internal Server Error”. Разработчики, хорошо знающие, насколько сложным является устранение нарушений в работе сервера, неукоснительно вводят эти две строки в код отлаживаемых программ.

## 10.3. Создание приложений CGI

В данном разделе мы перейдем к рассмотрению практических примеров. Вначале покажем, как установить веб-сервер, а затем подробно опишем приложение CGI на языке Python. Начнем с простого сценария, после чего будем последовательно его усложнять. Освоенные при этом методы могут использоваться для разработки приложений с использованием практически любой веб-платформы.

### 10.3.1. Установка веб-сервера

Для проведения экспериментов по разработке сценариев CGI на языке Python вначале необходимо установить веб-сервер, выполнить его настройку для обработки запросов CGI в среде Python, а затем предоставить веб-серверу доступ к сценариям CGI. Для решения некоторых из рассматриваемых задач может потребоваться помощь системного администратора.

### Серверы производственного назначения

Для экспериментирования с веб-сервером, который в дальнейшем можно будет применить в реальном приложении, следует загрузить и установить программное обеспечение Apache, ligHTTPD или thttpd. Для веб-сервера Apache предусмотрено несколько подключаемых программных модулей для обработки CGI Python, но в рассматриваемых примерах они не потребуются. Необходимость в установке этих модулей возникает, если устанавливаемый веб-сервер предназначен для обработки запросов, поступающих из-за пределов локальной сети. Однако и в таком случае применение подключаемых модулей может стать нецелесообразным.

### Серверы для разработчика

Для обучения или поддержки простых веб-сайтов может оказаться достаточным использование веб-серверов, которые входят в состав библиотеки Python. Сведения о том, как создавать и настраивать простые веб-серверы на основе Python, приведены в главе 9. В этой главе будут рассматриваться более простые примеры, в которых используется только веб-сервер Python с поддержкой CGI.

**2.x**

Чтобы запустить этот самый простой веб-сервер, в версии Python 2.x достаточно вызвать его непосредственно на выполнение следующим образом:

```
$ python -m CGIHTTPServer [port]
```

**3.x**

В версии Python 3 эта задача немного усложняется, поскольку все три веб-сервера и связанные с ними обработчики объединены в отдельный модуль (`http.server`), в котором определен один основной сервер и три класса обработчиков запросов (`BaseHTTPRequestHandler`, `SimpleHTTPRequestHandler` и `CGIHTTPRequestHandler`).

## 2.4

Параметр с номером порта сервера является необязательным и в случае его отсутствия по умолчанию устанавливается порт 8000. Кроме того, в версии 2.4 впервые появилась опция `-m`. Тем, кто использует более старую версию Python или желает ознакомиться с альтернативными версиями запуска сервера, целесообразно рассмотреть следующие темы.

- Вызов модуля на выполнение с помощью команды командного интерпретатора

Применение этого метода связано с определенными затруднениями, поскольку прежде необходимо узнать, где физически находится файл `CGIHTTPServer.py`. В персональных компьютерах с операционной системой Windows это проще, поскольку для установки, как правило, применяется папка `C:\Python2X`:

```
C:\>python C:\Python27\Lib\CGIHTTPServer.py
Serving HTTP on 0.0.0.0 port 8000 ...
```

В системах POSIX приходится заниматься дополнительными проверками:

```
>>> import sys, CGIHTTPServer
>>> sys.modules['CGIHTTPServer']
<module 'CGIHTTPServer' from '/usr/local/lib/python2.7/
CGIHTTPServer.py'>
>>> ^D
$ python /usr/local/lib/python2.7/CGIHTTPServer.py
Serving HTTP on 0.0.0.0 port 8000 ...
```

- Использование опции `-c`

С использованием опции `-c` можно выполнять инструкции Python записанные в строке. Таким образом, чтобы импортировать модуль `CGIHTTPServer` и выполнить функцию `test()`, необходимо сделать следующее:

```
$ python -c "import CGIHTTPServer; CGIHTTPServer.test()"
Serving HTTP on 0.0.0.0 port 8000 ...
```

## 3.x

В версиях Python 3.x модуль `CGIHTTPServer` вошел в состав `http.server`, поэтому можно выполнить эквивалентный вызов (работая, например, в версии Python 3.2) следующим образом:

```
$ python3.2 -c "from http.server import CGIHTTPRequestHandler; test(CGIHTT
PRequestHandler)"
```

- Создание несложного сценария

Прежде всего вставим вызовы `import` и `test()` из предыдущего примера в файл с произвольным именем, скажем, `cgihttpd.py` (Python 2 или 3). Поскольку в Python 3 не предусмотрен модуль `CGIHTTPServer.py`, который может быть вызван, единственный способ ввода в действие сервера для запуска из командной строки с портом, отличным от 8000, состоит в использовании следующего сценария:

```
$ python3.2 cgihttpd.py 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

При использовании любого из этих четырех способов произойдет запуск веб-сервера в порту 8000 (или в другом, который был выбран отдельно) на определенном компьютере из текущего каталога. После этого остается лишь создать подкаталог `cgi-bin` в каталоге, из которого запущен сервер, и поместить в него сценарии CGI Python. В каталог `cgi-bin` достаточно поместить необходимые файлы HTML и сценарии CGI с расширением `.py`, после чего можно переходить на вновь созданный веб-сайт, указывая примерно такие адреса:

```
http://localhost:8000/friends.htm
http://localhost:8080/cgi-bin/friendsB.py
```

Следует обязательно запускать сервер из каталога, в котором имеется подкаталог `cgi-bin`, и проверять наличие файлов `.py`; в противном случае сервер, применяемый для разработки, вернет файлы Python в виде статического текста вместо их выполнения.

### 10.3.2. Создание страницы формы

В примере 10.1 представлен код простой веб-формы, `friends.htm`. В этом коде HTML можно видеть, что форма содержит две входные переменные: `person` и `howmany`. Значения этих двух полей будут переданы в сценарий CGI, `friendsA.py`.

#### Пример 10.1. Статическая веб-страница формы (`friends.htm`)

С помощью этого файла HTML перед пользователем разворачивается форма с пустым полем ввода имени пользователя и с рядом переключателей, предоставляющих пользователю на выбор несколько значений.

```
1 <HTML><HEAD><TITLE>
2 Friends CGI Demo (static screen)
3 </TITLE></HEAD>
4 <BODY><H3>Friends list for: <I>NEW USER</I></H3>
5 <FORM ACTION="/cgi-bin/friendsA.py">
6 Enter your Name:
7 <INPUT TYPE=text NAME=person VALUE="NEW USER" SIZE=15>
8 <P>How many friends do you have?
9 <INPUT TYPE=radio NAME=howmany VALUE="0" CHECKED> 0
10 <INPUT TYPE=radio NAME=howmany VALUE="10"> 10
11 <INPUT TYPE=radio NAME=howmany VALUE="25"> 25
12 <INPUT TYPE=radio NAME=howmany VALUE="50"> 50
13 <INPUT TYPE=radio NAME=howmany VALUE="100"> 100
14 <P><INPUT TYPE=submit></FORM></BODY></HTML>
```

В данном примере заслуживает внимание тот факт, что сценарий CGI установлен в применяемом по умолчанию каталоге `cgi-bin` на локальном узле (см. ссылку ACTION). (Если в применяемой вами среде разработки выбраны другие варианты, обновите действие формы, прежде чем приступить к проверке сценария CGI и веб-страницы.) Кроме того, в действии формы не указан вторичный дескриптор METHOD, поэтому все запросы направляются с применяемым по умолчанию типом, GET. Здесь выбран метод GET, поскольку количество полей формы весьма невелико; кроме того, желательно, чтобы строка запроса появилась на панели Location (или Address, Go To). Это позволяет наблюдать за отправкой указателя URL на сервер.

На рис. 10.2 и 10.3 показан экран, подготавливаемый для просмотра с помощью файла `friends.htm` в клиентских программах, которые выполняются на компьютерах Mac и Windows.



Рис. 10.2. Страница формы Friends в анонимном режиме браузера Chrome на Mac OS X

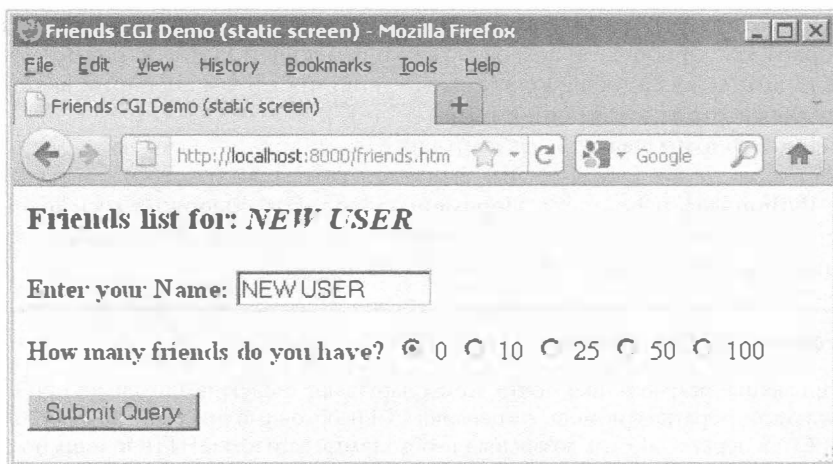


Рис. 10.3. Страница формы Friends в браузере Firefox 6 на Windows

### 10.3.3. Формирование страницы с результатами

Данные, введенные пользователем, передаются на сервер после щелчка на кнопке Submit (Отправить). (Чтобы выполнить то же действие, пользователь может также нажать клавишу <Return> или <Enter> в текстовом поле.) При передаче формы с помощью интерфейса CGI выполняется сценарий, приведенный в примере 10.2, `friendsA.py`.

**Пример 10.2. Экран с результатами выполнения кода CGI (friendsA.py)**

В этом сценарии CGI происходит считывание значений полей `person` и `howmany` формы и использование этих данных для создания динамически сформированного экрана результатов. В версии для Python 3, `friendsA3.py` (здесь не приведена) достаточно добавить круглые скобки в инструкции `print` в строке 17. Оба эти сценария можно найти по адресу `corepython.com`.

```

1 #!/usr/bin/env python
2
3 import cgi
4
5 reshtml = '''Content-Type: text/html\n
6 <HTML><HEAD><TITLE>
7 Friends CGI Demo (dynamic screen)
8 </TITLE></HEAD>
9 <BODY><H3>Friends list for: <I>%s</I></H3>
10 Your name is: %s<P>
11 You have %s friends.
12 </BODY></HTML>'''
13
14 form = cgi.FieldStorage()
15 who = form['person'].value
16 howmany = form['howmany'].value
17 print reshtml % (who, who, howmany)

```

Сценарий содержит все программные средства, необходимые для чтения входных данных формы, их обработки и возврата результирующей HTML-страницы пользователю. В действительности все нетривиальные действия в этом сценарии выполняются лишь в четырех строках кода Python (строки 14–17).

Переменной формы является экземпляр `FieldStorage`, который содержит значения полей `howmany` и `person`. Эти значения считываются, соответственно, в переменные Python `who` и `howmany`. Переменная `reshtml` содержит весь текст HTML, подлежащий возврату, в котором некоторые поля заполняются динамически с использованием данных, только что считанных из формы.

**Заголовки HTTP отделены от кода HTML**

Начинающие разработчики почти всегда нарушают следующее правило: при отправке результатов обратно с помощью сценария CGI необходимо предусмотреть, чтобы сценарий CGI в первую очередь возвращал необходимые заголовки HTTP и лишь после этого код HTML. Кроме того, чтобы можно было определить, где заканчиваются заголовки и начинается результирующий код HTML, необходимо вставить одну пустую строку (пара символов `NEWLINE`) между обоими наборами данных, как показано в строке 5 нашего примера `friendsA.py` (один символ `\n` передается явно, а другой — неявно, в конце строки 5). Об этом следует помнить и при рассмотрении других примеров.

Один из возможных экранов с результатами показан на рис. 10.4 (при условии, что пользователь ввел имя “Annalee Lenday” и щелкнул на переключателе “25 friends”).

Разработчику веб-сайта может прийти в голову идея, что было бы неплохо предусмотреть автоматическое преобразование в прописные первых букв введенных имен



и фамилий на тот случай, что они введены как строчные. С помощью интерфейса CGI в языке Python это можно сделать легко. (Ниже приведен соответствующий пример.)

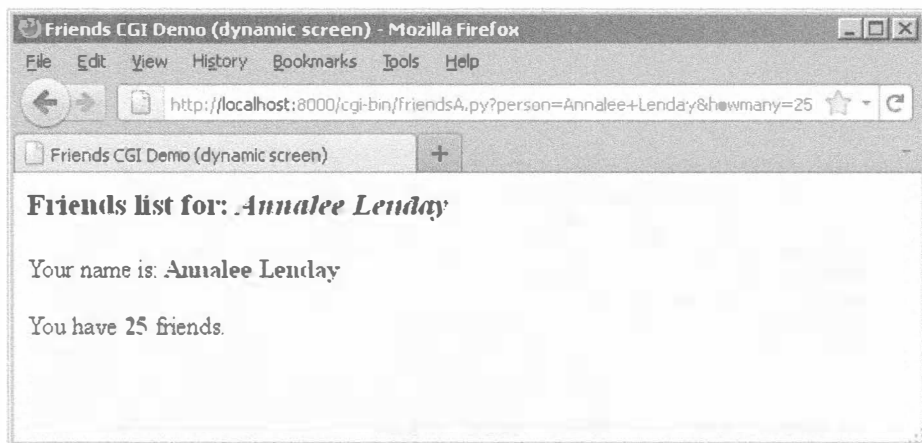


Рис. 10.4. Страница с результатами сценария Friends после отправки данных об имени и количестве друзей

Заслуживает внимания то, что в запросе GET к указателю URL в поле адреса добавлены переменные формы и их значения. Кроме того, можно отметить, что в заголовке страницы `friends.htm` имеется слово “static”, а на экране вывода результатов `friends.py` в заголовке присутствует слово “dynamic”. Это сделано намеренно, чтобы указать, что `friends.htm` — это файл со статическим текстом, а страница с результатами формируется динамически. Иными словами, код HTML страницы с результатами не присутствует на диске в виде текстового файла, а формируется в сценарии CGI, который возвращает этот текст, как будто он представляет собой содержимое обычного локального файла.

В следующем примере мы вообще обойдемся без статических файлов, поскольку рассматриваемый сценарий CGI станет немного более разноплановым.

### 10.3.4. Формирование страниц с формой и результатами

На следующем этапе мы откажемся от использования файла `friends.html` и введем весь необходимый код в файл `friendsB.py`. В рассматриваемом сценарии теперь будут формироваться и страница с формой, и страница с результатами. Как же сообщить сценарию, какая страница должна быть сформирована? Итак, если в сценарий поступают данные формы, это означает, что должна быть создана страница с результатами. Если же не в сценарий не поступает вообще никакой информации, то можно сделать вывод, что следует сформировать страницу с формой для пользователя, в которую он может ввести свои данные. Новый вариант сценария, `friendsB.py`, представлен в примере 10.3.

#### Пример 10.3. Формирование страницы с формой и страницы с результатами (`friendsB.py`)

Оба сценария, `friends.htm` и `friendsA.py`, объединены в сценарий `friendsB.py`, который теперь может выводить и страницу формы, и страницу с результатами в виде динамически формируемого кода HTML, а также автоматически выбирать,

какая страница должна быть выведена. Для преобразования этого сценария в версию для Python 3, friendsB3.py необходимо добавить круглые скобки к обоим инструкциям **print** и изменить действие формы для friendsB3.py.

```

1 #!/usr/bin/env python
2
3 import cgi
4
5 header = 'Content-Type: text/html\n\n'
6
7 formhtml = '''<HTML><HEAD><TITLE>
8 Friends CGI Demo</TITLE></HEAD>
9 <BODY><H3>Friends list for: <I>NEW USER</I></H3>
10 <FORM ACTION="/cgi-bin/friendsB.py">
11 Enter your Name:
12 <INPUT TYPE=hidden NAME=action VALUE=edit>
13 <INPUT TYPE=text NAME=person VALUE="NEW USER" SIZE=15>
14 <P>How many friends do you have?
15 %s
16 <P><INPUT TYPE=submit></FORM></BODY></HTML>'''
17
18 fradio = '<INPUT TYPE=radio NAME=howmany VALUE="%s" %s> %s\n'
19
20 def showForm():
21 friends = []
22 for i in (0, 10, 25, 50, 100):
23 checked = ''
24 if i == 0:
25 checked = 'CHECKED'
26 friends.append(fradio % (str(i), checked, str(i)))
27
28 print '%s' % (header, formhtml % ''.join(friends))
29
30 reshtml = '''<HTML><HEAD><TITLE>
31 Friends CGI Demo</TITLE></HEAD>
32 <BODY><H3>Friends list for: <I>%s</I></H3>
33 Your name is: %s<P>
34 You have %s friends.
35 </BODY></HTML>'''
36
37 def doResults(who, howmany):
38 print header + reshtml % (who, who, howmany)
39
40 def process():
41 form = cgi.FieldStorage()
42 if 'person' in form:
43 who = form['person'].value
44 else:
45 who = 'NEW USER'
46
47 if 'howmany' in form:
48 howmany = form['howmany'].value
49 else:
50 howmany = 0
51
52 if 'action' in form:
53 doResults(who, howmany)

```

```
54 else:
55 showForm()
56
57 if __name__ == '__main__':
58 process()
```

## Построчное объяснение

### Строки 1–5

Кроме обычных строк запуска и импорта модулей, добавлена инструкция, отделяющая заголовок MIME HTTP от основного текста HTML, причем эта инструкция используется для страниц обоих типов (страницы формы и страницы с результатами), поскольку это позволяет сократить объем кода сценария. Строка заголовка добавляется к соответствующему тексту HTML во время формирования вывода.

### Строки 7–28

Весь этот код относится к странице формы `friends.htm`, которая теперь вошла в состав сценария CGI. Для текста страницы формы предусмотрена переменная `formhtml`, кроме того, отдельная строка применяется для построения списка переключателей, `fradio`. Можно было бы продублировать код HTML с определением переключателей, как в файле `friends.htm`, но мы хотели показать, как использовать язык Python для более динамичного формирования вывода (см. цикл `for` в строках 22–26).

Функция `showForm()` предназначена для формирования формы, в которую пользователь вводит данные. В ней формируется ряд инструкций с определением переключателей, эти строки кода HTML объединяются с основной частью переменной `formhtml`, к форме добавляется заголовок, а затем происходит возврат всей коллекции данных клиенту путем отправки окончательно сформированной строки на стандартное устройство вывода.

В этом коде заслуживают внимания несколько интересных особенностей. Прежде всего рассмотрим скрытую переменную в форме `action`, которая содержит значение `edit` в строке 12. Лишь с помощью этого поля можно узнать, какая страница должна быть сформирована (страница с формой или страница с результатами). Как применяется это поле, можно видеть в строках 53–56.

Кроме того, отметим, что переключатель “0” задан как применяемый по умолчанию путем установки на нем отметки при последовательном формировании кнопок с помощью цикла. Это также позволяет обновлять компоновку переключателей и (или) их значений в одной строке кода (строка 18), а не заниматься правкой нескольких строк текста. К тому же достигается большая гибкость, поскольку по условию можно определить, какой переключатель выбран (см. следующее обновление рассматриваемого сценария, `friendsC.py`).

У читателя может возникнуть вопрос: “Для чего требуется переменная `action`, если можно было бы просто проверить присутствие данных `person` или `howmany`?” Это резонный вопрос, поскольку в данной ситуации действительно можно было бы просто проверить значение `person` или `howmany`.

Переменная `action` более явно демонстрирует свое присутствие в коде, поскольку имеет имя, соответствующее ее назначению, и в конечном итоге код становится

проще для понимания. Переменные `person` и `howmany` используются как носители значений, а переменная `action` — как флажок.

Еще одной причиной введения в код переменной `action` является то, что с ее помощью можно проще определить, какая страница должна быть сформирована. В частности, если переменная `person` содержит значение, то должна быть сформирована страница с формой (а не страница с результатами). Код, который действует исключительно на основании проверки содержимого переменной `person`, не позволяет создать надежный сценарий.

### Строки 30–38

Код, предназначенный для отображения страницы с результатами, фактически идентичен тому, что применяется в сценарии `friendsA.py`.

### Строки 40–55

В связи с тем, что один и тот же сценарий служит для формирования разных страниц, было решено создать общую функцию `process()` для получения данных формы и определения того, какое действие должно быть выполнено. Основная часть функции `process()` также является аналогичной основной части кода сценария `friendsA.py`. Однако следует отметить два важных различия.

Дело в том, что сценарий на разных этапах может как получать, так и не получать поля формы (например, при первом вызове сценария для формирования страницы формы на сервер не передаются какие-либо поля), поэтому операторы выборки полей формы заключить необходимо в блоки с помощью инструкций `if` для проверки самого факта существования данных. Кроме того, как было указано выше, для определения страницы, формируемой на данном этапе, применяется поле `action`. Код, в котором происходит соответствующая проверка, находится в строках 52–55.

На рис. 10.5 показано, что автоматически сформированная форма является идентичной статической форме, представленной на рис. 10.2, но ссылка для вызова этой формы оканчивается суффиксом `.py`, а не `.html`. После ввода имени Cynthia Gilbert, выбора 50 друзей и щелчка на кнопке `Submit` создается форма, показанная на рис. 10.6.

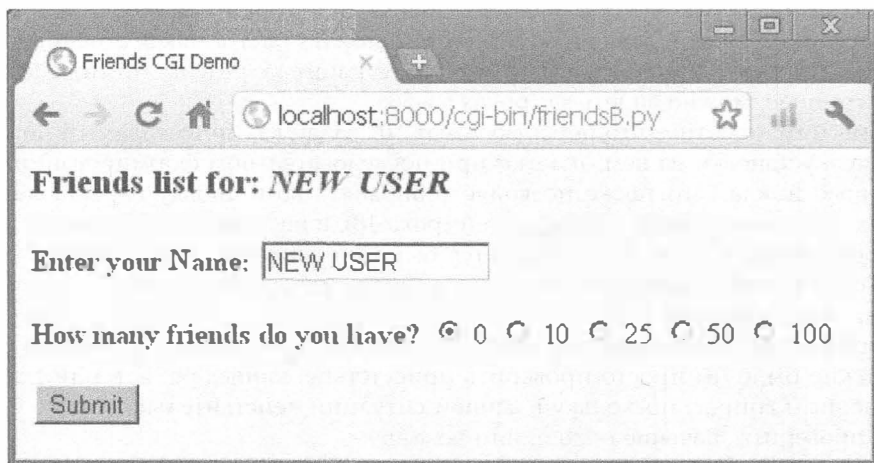


Рис. 10.5. Автоматически сформированная страница формы `Friends` в браузере Chrome в Windows

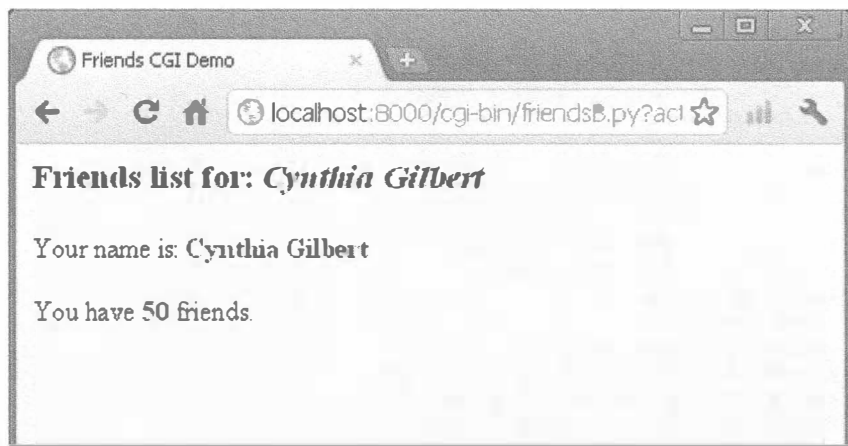


Рис. 10.6. Страница с результатами Friends после передачи имени и числа друзей

Заслуживает внимания то, что в указателе URL не отображается имя статической страницы `friends.htm`, поскольку формирование и страницы формы, и страницы с результатами обеспечивает сценарий `friendsB.py`.

### 10.3.5. Полностью интерактивные веб-сайты

Рассмотрим заключительный пример, который должен наилучшим образом завершить эту тему. Как и перед этим, пользователь вводит информацию на странице формы. После этого данные обрабатываются и формируется страница с результатами. На этот раз на странице с результатами будет добавлена ссылка, которая позволяет пользователю вернуться на страницу формы, причем теперь эта страница будет не пустой, а заполненной данными, представленными ранее пользователем. Кроме того, будут введены определенные средства обработки ошибок в качестве примера того, как решить эту задачу. Новый сценарий `friendsC.py` показан в примере 10.4.

#### Пример 10.4. Сценарий, обеспечивающий полное взаимодействие с пользователем и обработку ошибок (`friendsC.py`)

Добавляя ссылку для возврата на страницу формы с представленной ранее информацией, мы создаем полный цикл, который позволяет пользователю полноценно выполнять свою задачу с помощью веб-интерфейса. Кроме того, в приложении теперь осуществляется несложная проверка ошибок, с помощью которой можно определить, что пользователь не выбрал ни один из переключателей, и сообщить ему об этом.

```

1 #!/usr/bin/env python
2
3 import cgi
4 from urllib import quote_plus
5
6 header = 'Content-Type: text/html\n\n'
7 url = '/cgi-bin/friendsC.py'
8
9 errhtml = '''<HTML><HEAD><TITLE>
10 Friends CGI Demo</TITLE></HEAD>

```

```

11 <BODY><H3>ERROR</H3>
12 B>%s<P>
13 FORM<INPUT TYPE=button VALUE=Back
14 ONCLICK="window.history.back()"></FORM>
15 </BODY></HTML>'''
16
17 def showError(error_str):
18 print header + errhtml % error_str
19
20 formhtml = '''<HTML><HEAD><TITLE>
21 Friends CGI Demo</TITLE></HEAD>
22 <BODY><H3>Friends list for: <I>%s</I></H3>
23 <FORM ACTION="%s">
24 Enter your Name:
25 <INPUT TYPE=hidden NAME=action VALUE=edit>
26 <INPUT TYPE=text NAME=person VALUE="%s" SIZE=15>
27 <P>How many friends do you have?
28 %s
29 <P><INPUT TYPE=submit></FORM></BODY></HTML>'''
30
31 fradio = '<INPUT TYPE=radio NAME=howmany VALUE="%s" %s> %s\n'
32
33 def showForm(who, howmany):
34 friends = []
35 for i in (0, 10, 25, 50, 100):
36 checked = ''
37 if str(i) == howmany:
38 checked = 'CHECKED'
39 friends.append(fradio % (str(i), checked, str(i)))
40 print '%s' % (header, formhtml % (
41 who, url, who, ''.join(friends)))
42
43 reshtml = '''<HTML><HEAD><TITLE>
44 Friends CGI Demo</TITLE></HEAD>
45 <BODY><H3>Friends list for: <I>%s</I></H3>
46 Your name is: %s<P>
47 You have %s friends.
48 <P>Click here to edit your data again.
49 </BODY></HTML>'''
50
51 def doResults(who, howmany):
52 newurl = url + '?action=reedit&person=%s&howmany=%s' \
53 (quote_plus(who), howmany)
54 print header + reshtml % (who, who, howmany, newurl)
55
56 def process():
57 error = ''
58 form = cgi.FieldStorage()
59
60 if 'person' in form:
61 who = form['person'].value.title()
62 else:
63 who = 'NEW USER'
64
65 if 'howmany' in form:
66 howmany = form['howmany'].value
67 else:
68 if 'action' in form and \

```

```
69 form['action'].value = 'edit':
70 error = 'Please select number of friends.'
71 else:
72 howmany = 0
73
74 if not error:
75 if 'action' in form and \
76 form['action'].value != 'reedit':
77 doResults(who, howmany)
78 else:
79 showForm(who, howmany)
80 else:
81 showError(error)
82
83 if __name__ == '__main__':
84 process()
```

---

Сценарий `friendsC.py` не содержит существенных отличий от `friendsB.py`. Предлагаем вначале сравнить два сценария самому читателю. Затем будет приведена краткая сводка основных расхождений между этими двумя вариантами.

## Сокращенное построчное объяснение

### Строка 7

Из формы изъята строка с указателем URL, поскольку теперь она нужна в двух местах; вызов страницы с результатами происходит отдельно от страницы формы ввода данных пользователем.

### Строки 9–18, 68–70, 74–81

Все эти строки относятся к новой функции сценария — обработке и выводу ошибок. Если пользователь не выбрал переключатель, указывающий количество друзей, то на сервер не поступает значение поля `howmany`. В таком случае функция `showError()` возвращает пользователю страницу с ошибкой.

На странице с сообщением об ошибке находится кнопка “Назад”, поддерживаемая кодом JavaScript. Кнопки относятся к элементам управления, предназначенным для ввода, поэтому для кнопки “Назад” должна быть предусмотрена форма, но какое-либо результативное действие не требуется, поскольку должен быть всего лишь выполнен возврат на предыдущую строку журнала просмотра, которая относится к отображенной ранее странице. Рассматриваемый сценарий в настоящее время выполняет обработку (проверку) ошибок только одного типа, но все равно в нем предусмотрена универсальная переменная `error` на тот случай, если будет решено продолжить разработку сценария для расширения состава обнаруживаемых ошибок.

### Строки 26–28, 37–40, 47 и 51–54

Одно из назначений этого сценария состоит в формировании значимой обратной ссылки со страницы с результатами на страницу формы. Это реализовано с применением ссылки, которая позволяет пользователю вернуться на страницу формы, чтобы исправить или обновить введенные им данные. Применение этой новой страницы формы имеет смысл, только если в ней содержится информация, относящаяся к тем данным, которые введены ранее пользователем. (Пользователей очень раздражает,

когда им приходится повторно вводить одну и ту же информацию после того, как форма, к которой они вернулись, оказывается пустой!)

Чтобы решить эту задачу, необходимо предусмотреть ввод текущих значений в обновленную форму. В строке 26 добавлено значение для имени. Это значение, если оно задано, будет вставлено в поле имени. Очевидно, что после первой прорисовки страницы формы это поле будет пустым. В строках 37-38 определяется поле переключателей, соответствующее количеству друзей, выбранному в настоящее время. Наконец, в строке 48 и строках 52-54, где определена обновленная функция `doResults()`, создается ссылка со всей существующей информацией, которая возвращает пользователя к обновленной странице формы.

## Строка 61

В конечном итоге происходит добавление простого средства, которое, по мнению автора, способствует повышению эстетической привлекательности пользовательского интерфейса. В формах, которые создаются с помощью сценариев `friendsA.py` и `friendsB.py`, текст, введенный пользователем в качестве своего имени, остается неизменным. И действительно, как показывают строки сценариев `friendsA.py` и `friendsB.py`, предназначенные для работы с полем ввода имени в форме, какие-либо дополнительные действия с введенными данными не выполняются. Таким образом, если пользователь вводит имя и фамилию строчными буквами, они так и отображаются, в нижнем регистре, и т.д. Поэтому добавлен вызов функции `str.title()`, которая выполняет автоматические преобразования с именем пользователя. Строковый метод `title()` выделяет первую букву переданного ему слова как прописную. Такое преобразование может потребоваться или не потребоваться в той или иной версии, но оно предусмотрено в данном примере как свидетельство того, что в сценарии вполне могут применяться дополнительные действия.

На рис. 10.7-10.10 демонстрируется ход взаимодействия пользователя с формой CGI и сценарием.

На рис. 10.7 показан результат выполнения сценария `friendsC.py` для формирования страницы формы. В качестве имени введены слова "foo bar", но намеренно не выбран ни один из переключателей. Это приводит к появлению сообщения об ошибке, которое отображается в форме, показанной на рис. 10.8.

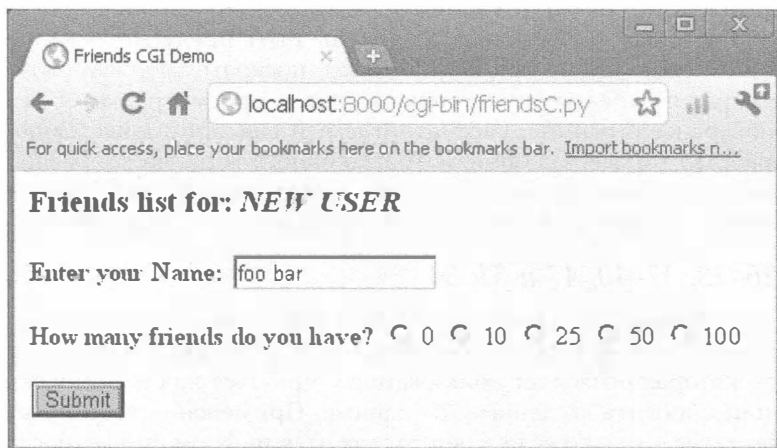


Рис. 10.7. Начальная страница формы Friends, на которой не указано количество друзей



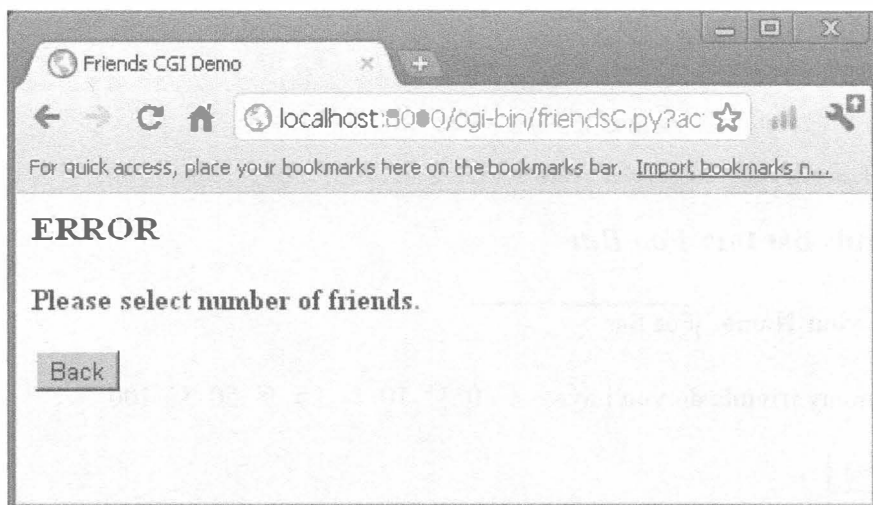


Рис. 10.8. Страница с сообщением об ошибке, которая формируется после обнаружения неправильного ввода данных пользователем

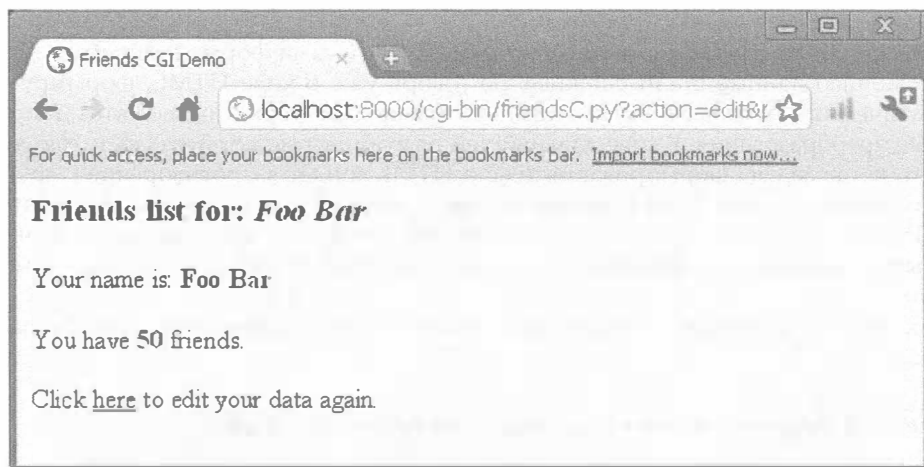


Рис. 10.9. Страница с результатами обработки правильно заполненной формы Friends

Затем производится щелчок на кнопке "Назад", щелчок на переключателе 50 и повторная отправка формы. Страница с результатами, показанная на рис. 10.9, также уже встречалась в данной главе, но теперь на ней имеется дополнительная ссылка в нижней части, с помощью которой можно вернуться к странице формы. Единственное различие между новой страницей формы и исходной страницей состоит в том, что теперь данные, введенные пользователем, рассматриваются как применяемые по умолчанию, иными словами, снова появляются в форме после возвращения на предыдущий шаг. (Заслуживает внимания то, что сохраняется и результат автоматического преобразования первых букв имени в прописные.) Сформированная страница показана на рис. 10.10.

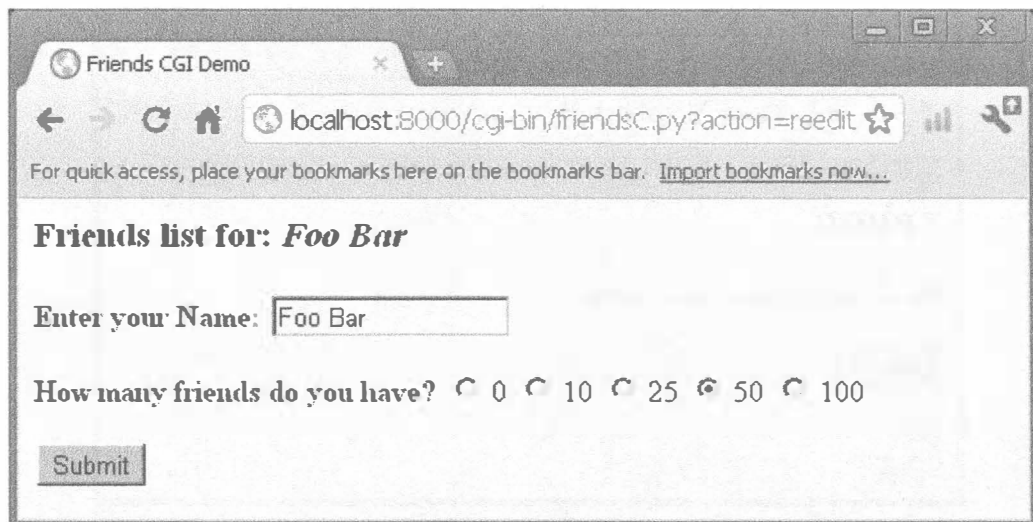


Рис. 10.10. Новая версия страницы формы Friends

Теперь пользователь получил возможность вносить изменения в любые поля формы и повторно отправлять эту форму.

Однако любой разработчик сразу же заметит, что сама форма и обрабатываемые с ее помощью данные стали сложнее, не говоря уже о коде HTML, формируемом автоматически, тем более в связи с тем, что с данными формы проводятся дополнительные преобразования. В ходе дальнейшей разработки может быть достигнут такой момент, когда задача формирования текста HTML войдет в противоречие с другими задачами приложения. В этом случае можно прибегнуть к использованию таких пакетов Python, как HTMLgen, xist или HSC. Это инструменты сторонних разработчиков, которые специально предназначены для создания кода HTML с помощью объектов Python.

Наконец, в примере 10.5 показан эквивалентный сценарий для Python 3, friendsC3.py.

#### Пример 10.5. Результат переноса friendsC.py в Python 3 (friendsC3.py)

Данный сценарий, предназначенный для тех же целей, что и friendsC.py, создан для версии Python 3. Рассмотрим, что в нем изменилось.

```

1 #!/usr/bin/env python
2
3 import cgi
4 from urllib.parse import quote_plus
5
6 header = 'Content-Type: text/html\n\n'
7 url = '/cgi-bin/friendsC3.py'
8
9 errhtml = '''<HTML><HEAD><TITLE>
10 Friends CGI Demo</TITLE></HEAD>
11 <BODY><H3>ERROR</H3>
12 %s<P>
13 <FORM><INPUT TYPE=button VALUE=Back

```

```

14 ONCLICK="window.history.back()" "></FORM>
15 </BODY></HTML>'''
16
17 def showError(error_str):
18 print(header + errhtml % (error_str))
19
20 formhtml = '''<HTML><HEAD><TITLE>
21 Friends CGI Demo</TITLE></HEAD>
22 <BODY><H3>Friends list for: <I>%s</I></H3>
23 <FORM ACTION="%s">
24 Enter your Name:
25 <INPUT TYPE=hidden NAME=action VALUE=edit>
26 <INPUT TYPE=text NAME=person VALUE="%s" SIZE=15>
27 <P>How many friends do you have?
28 %s
29 <P><INPUT TYPE=submit></FORM></BODY></HTML>'''
30
31 fradio = '<INPUT TYPE=radio NAME=howmany VALUE="%s" %s> %s\n'
32
33 def showForm(who, howmany):
34 friends = []
35 for i in (0, 10, 25, 50, 100):
36 checked = ''
37 if str(i) == howmany:
38 checked = 'CHECKED'
39 friends.append(fradio % (str(i), checked, str(i)))
40 print('%s%s' % (header, formhtml % (
41 who, url, who, ''.join(friends))))
42
43 reshtml = '''<HTML><HEAD><TITLE>
44 Friends CGI Demo</TITLE></HEAD>
45 <BODY><H3>Friends list for: <I>%s</I></H3>
46 Your name is: %s<P>
47 You have %s friends.
48 <P>Click here to edit your data again.
49 </BODY></HTML>'''
50
51 def doResults(who, howmany):
52 newurl = url + '?action=reedit&person=%s&howmany=%s'%(
53 quote_plus(who), howmany)
54 print(header + reshtml % (who, who, howmany, newurl))
55
56 def process():
57 error = ''
58 form = cgi.FieldStorage()
59
60 if 'person' in form:
61 who = form['person'].value.title()
62 else:
63 who = 'NEW USER'
64
65 if 'howmany' in form:
66 howmany = form['howmany'].value
67 else:
68 if 'action' in form and \
69 form['action'].value == 'edit':
70 error = 'Please select number of friends.'
71 else:
72 howmany = 0

```

```

73
74 if not error:
75 if 'action' in form and \
76 form['action'].value != 'reedit':
77 doResults(who, howmany)
78 else:
79 showForm(who, howmany)
80 else:
81 showError(error)
82
83 if __name__ == '__main__':
84 process()

```

## 10.4. Использование стандарта кодирования Юникод с интерфейсом CGI

В главе “Последовательности” книг *Core Python Programming* и *Core Python Language Fundamentals* показано, как используются строки данных в стандарте кодирования Юникоде. В одном из разделов приведен простой пример сценария, в котором берется строка данных в Юникоде, записывается в файл и снова считывается в память. В данном разделе демонстрируется аналогичный сценарий CGI, в котором осуществляется вывод в Юникоде. Кроме того, показано, как передать в браузер указания, необходимые для того, чтобы отображение символов Юникода на экране осуществлялось должным образом. Одно из требований состоит в том, что на компьютере должны быть установлены восточноазиатские шрифты, чтобы браузер мог их отображать.

Для того чтобы понять, как обрабатывается Юникод, создадим сценарий CGI для формирования многоязычной веб-страницы. Вначале зададим сообщение в виде строки данных в Юникоде. Предполагается, что текстовый редактор, применяемый читателем, предназначен для работы только с кодом ASCII. Поэтому ввод символов, отличных от ASCII, осуществляется с помощью экранирующего символа `\u`. На практике сообщение может быть также прочитано из файла или из базы данных.

```

Greeting in English, Spanish,
Chinese and Japanese.
UNICODE_HELLO = u"""
Hello!
\u00A1Hola!
\u4F60\u597D!
\u3053\u3093\u306B\u3061\u306F!
"""

```

В первую очередь сценарий CGI формирует заголовок HTTP с указанием типа содержимого, `content-type`. В этом заголовке очень важно объявить, что содержимое передается в кодировке UTF-8, чтобы браузер мог правильно его интерпретировать:

```

print 'Content-type: text/html; charset=UTF-8\r'
print '\r'

```

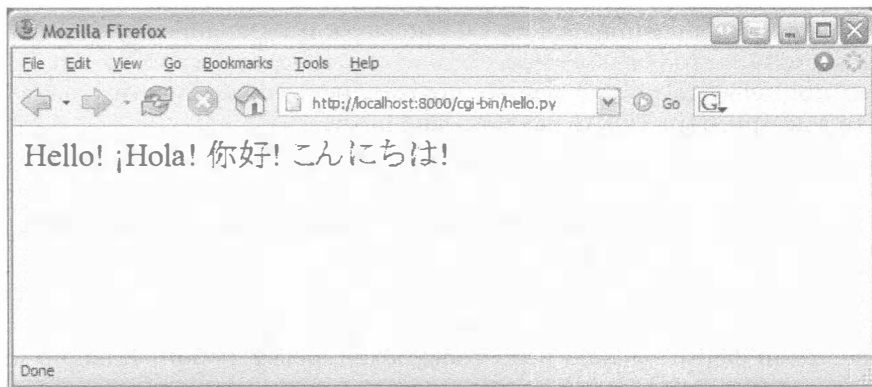
Затем выводится само сообщение. В первую очередь воспользуемся строковым методом `encode()` для преобразования строки в последовательность символов UTF-8:

```

print UNICODE_HELLO.encode('UTF-8')

```

Ознакомьтесь с кодом в примере 10.6, в результате выполнения которого формируется примерно такой вывод, как показано в окне обозревателя на рис. 10.11.



**Рис. 10.11.** Простой пример с демонстрацией вывода данных в Юникоде с помощью CGI в браузере Firefox

#### Пример 10.6. Простой вывод данных в Юникоде с помощью CGI (uniCGI.py)

В этом сценарии формируются строки данных в Юникоде для вывода в веб-браузер.

```

1 #!/usr/bin/env python
2
3 CODEC = 'UTF-8'
4 UNICODE_HELLO = u'''
5 Hello!
6 \u00A1Hola!
7 \u4F60\u597D!
8 \u3053\u3093\u306B\u3061\u306F!
9 '''
10
11 print 'Content-Type: text/html; charset=%s\r' % CODEC
12 print '\r'
13 print '<HTML><HEAD><TITLE>Unicode CGI Demo</TITLE></HEAD>'
14 print '<BODY>'
15 print UNICODE_HELLO.encode(CODEC)
16 print '</BODY></HTML>'

```

## 10.5. Расширения CGI

Теперь перейдем к рассмотрению более сложных аспектов программирования интерфейса CGI. К этому относится использование cookie-файлов (кэшированных данных, сохраняемых на клиентском компьютере), сохранение нескольких значений для одного и того же поля CGI, а также передача файла на сервер с использованием средств отправки многокомпонентной формы. Для экономии места ниже в данной главе показана реализация всех этих трех средств в одном приложении. Вначале рассмотрим передачу многокомпонентной формы.

### 10.5.1. Передача многокомпонентной формы и передача файла

В настоящее время спецификация CGI поддерживает кодировки формы только двух типов: `application/x-www-form-urlencoded` и `multipart/form-data`. Первая кодировка применяется по умолчанию, поэтому нет необходимости указывать тип кодировки в дескрипторе FORM, как показано ниже.

```
<FORM enctype="application/x-www-form-urlencoded" ...>
```

Если форма является многокомпонентной, то ее кодировка должна быть задана явно:

```
<FORM enctype="multipart/form-data" ...>
```

Для передачи формы можно использовать кодировку любого типа, но при выгрузке файла может применяться только многокомпонентная кодировка. Многокомпонентная кодировка была изобретена компанией Netscape на ранних этапах развития веб-технологий и с тех пор нашла широкое распространение, и применяется во всех ведущих версиях браузеров.

Передача файла выполняется с применением поля с именем файла:

```
<INPUT type=file name=...>
```

При обработке этого кода на странице формируется пустое текстовое поле с кнопкой сбоку, которая позволяет просматривать структуру каталогов и выполнять поиск файла, подлежащего выгрузке. Если используется многокомпонентная кодировка, то клиентская веб-форма, подготовленная к отправке на сервер, весьма напоминает (многокомпонентные) сообщения электронной почты с вложениями. Требуется отдельная кодировка, поскольку вряд ли возможно передать файл через URL, особенно двоичный файл. И в том и в другом случае данные доставляются на сервер, но их упаковка происходит по-разному.

Независимо от того, используется ли кодировка по умолчанию или многокомпонентная кодировка, модуль `cgi` обрабатывает данные одинаковым образом, подготавливая перед передачей формы ключи и соответствующие значения. Доступ к данным, как и прежде, осуществляется с помощью экземпляра `FieldStorage`.

### 10.5.2. Многозначные поля

После изучения процесса передачи файлов, перейдем к описанию особенностей обработки полей с несколькими значениями. Наиболее часто применяется способ работы со страницей, предусматривающий предоставление пользователю флажков для выбора нужных ему вариантов. Каждый из флажков обозначен меткой с именем поля, но эти метки одинаковы, поэтому для проведения различия между ними с каждым конкретным флажком связано отдельное значение.

Как уже было сказано, при передаче формы данные, введенные пользователем, передаются на сервер в виде пар “ключ–значение”. Если происходит отправка значения больше чем одного флажка, то обнаруживается, что с одинаковым ключом связано несколько значений. В этих случаях в модуле `cgi` создается не один экземпляр `MiniFieldStorage`, а список таких экземпляров, который можно обрабатывать в цикле для выборки различных значений. Вообще говоря, это не так уж сложно.

### 10.5.3. Cookie-файлы

Наконец, в этом примере рассмотрим использование cookie-файлов. Кратко можно отметить, что cookie-файлы — всего лишь фрагменты информации, которые сохраняются веб-сервером на клиентском компьютере (с помощью браузера), а затем запрашиваются.

Протокол HTTP не сохраняет состояние. Это означает, что каждая операция выполняется с его помощью без учета результатов предыдущих операций. Поэтому для переноса информации с одной страницы на другую применяются пары “ключ–значение” в запросе, как показано в запросах GET и на страницах, которые приведены выше в этой главе. Еще один способ выполнения задачи связывания форм, который также был показан выше, состоит в использовании скрытых полей формы, таких как поле для переменной `action`, применявшееся в некоторых из последних сценариев `friends*.py`. Этими переменными и их значениями управляет сервер, поскольку именно он формирует страницы, отправляемые клиенту, в которых должны находиться указанные данные.

Еще один способ сохранения состояния при переходе от одной страницы к другой состоит в сохранении данных о состоянии на клиенте. Для этого применяются cookie-файлы. При этом вместо сохранения данных в возвращаемых веб-страницах сервер передает запрос клиенту на сохранение cookie-файла. Cookie-файл связан с доменом сервера-отправителя (поэтому сервер не может задавать или перекрывать cookie-файлы с других веб-сайтов) и имеет дату истечения срока действия. Таким образом cookie-файлы со временем уничтожаются и не происходит их неконтролируемое накопление на клиентском компьютере.

Кроме этих двух отличительных особенностей, cookie-файлы отличаются тем, что состоят из элементов “ключ–значение”, представляющих требуемые данные. Другими атрибутами cookie-файлов являются уточненное обозначение пути к домену или запрос, который cookie-файл должен передавать только в безопасной среде.

Использование cookie-файлов позволяет избавиться от необходимости передавать данные со страницы на страницу для отслеживания пользователя. Безусловно, идея применения cookie-файлов подвергалась критике в связи с тем, что эта технология может стать причиной нарушения конфиденциальности, но на большинстве веб-сайтов специально предусмотрено их использование. Прежде чем приступить к рассмотрению кода, отметим, что веб-сервер отправляет запрос клиенту на сохранение cookie-файла в виде заголовка `Set-Cookie`, за которым непосредственно следует указанный файл.

После сохранения cookie-файла на клиентском компьютере в запросы к серверу автоматически включается этот cookie-файл с использованием сервера переменной окружения `HTTP_COOKIE`. В cookie-файлах разграничителями служат точки с запятой (;), а каждая пара “ключ–значение” разделяется знаком равенства (=). Для получения доступа к этим значениям данных в приложении достаточно провести разбиение строки с данными на компоненты (например, с помощью функции `str.split()` или обработки вручную).

Cookie-файлы, как и многокомпонентная кодировка, впервые были разработаны компанией Netscape. Представители этой компании подготовили также первый вариант спецификации для этой технологии, который до сих пор почти не утратил своей значимости. Доступ к этому документу можно получить на следующем веб-сайте:

В дальнейшем была проведена стандартизация технологии cookie-файлов, и указанный документ окончательно стал устаревшим. Для получения более актуальной информации необходимо воспользоваться документами RFC, так называемыми “предложениями к обсуждению”. Первой официальной публикацией, касающейся cookie-файлов, был документ RFC 2109, который вышел в 1997 г. Несколько лет спустя, в 2000 г., он был заменен документом RFC 2965. Самой современной версией (которая заменила две другие) ко времени написания этой книги является RFC 6265, опубликованной в апреле 2011 г.

### 10.5.4. Cookie-файлы и передача файлов

Теперь приступим к рассмотрению приложения CGI, `advcgi.py`, код и функционирование которого в определенной степени напоминает сценарий `friendsC.py`, приведенный ранее в этой главе. Первой страницей, применяемой по умолчанию, является форма, заполняемая пользователем, которая состоит из четырех основных частей: задаваемая пользователем строка cookie-файла, поле с именем, список выбора языков программирования и поле с указанием отправляемого файла. На рис. 10.12 показаны внешний вид этой формы и пример частичного заполнения.

Advanced CGI Demo Form

My Cookie Setting

• CPPuser = (cookie has not been set yet)

Enter cookie value

Hey, I got a cookie! (optional)

Enter your name

Wesley (required)

What languages can you program in? (at least one required)

☒ Python ☐ Ruby ☐ Java ☐ C++ ☐ PHP ☒ C ☐ JavaScript

Enter file to upload (max size 4K)

Choose File bio.txt

Submit

Рис. 10.12. Расширенный cookie-файл CGI, поле для указания передаваемого файла и страница с многокомпонентной формой



Все данные передаются на сервер с использованием многокомпонентной кодировки, а получение этих данных на сервере происходит, как и в предыдущих примерах, с применением экземпляра `FieldStorage`. Единственная сложность состоит в том, как получить переданный файл. В данном приложении решено организовать построчную обработку файла в цикле. Кроме того, если известно, что файл не слишком велик, то можно прочитать сразу все его содержимое.

Это первое событие получения данных сервером, поэтому именно сейчас при возврате страницы с результатами клиенту необходимо задать заголовок "Set-Cookie:" для кэширования данных в cookie-файле браузера.

На рис. 10.13 показаны результаты, полученные после передачи данных формы. На странице показаны значения всех полей, введенных пользователем. Как показано в последнем диалоговом окне, данный файл загружается на сервер и отображается.



Рис. 10.13. Страница с результатами расширенного приложения CGI

Заслуживает также внимания ссылка в нижней части страницы, которая позволяет вернуться на страницу форму, опять-таки с применением того же сценария CGI.

После щелчка на этой ссылке в нижней части не происходит отправка данных формы в сценарий, а отображается страница формы. Тем не менее, как показано на рис. 10.14, отображаемая форма больше не пуста; на ней уже присутствует информация, ранее введенная пользователем. Как удалось этого достичь без применения данных формы (представленных как скрытые поля формы или как параметры запроса в URL)? Секрет заключается в том, что данные сохранены на клиентском компьютере в cookie-файле (а фактически в двух cookie-файлах).

Advanced CGI Demo

localhost:8000/cgi-bin/advcgi.py

## Advanced CGI Demo Form

### My Cookie Setting

• CPPuser = Hey, I got a cookie!

Enter cookie value

Hey, I got a cookie! (optional)

Enter your name

Wesley (required)

What languages can you program in? (at least one required)

☒ Python ☐ Ruby ☐ Java ☐ C++ ☐ PHP ☒ C ☐ JavaScript

Enter file to upload

Choose File No file chosen

Submit

Рис. 10.14. Новая страница формы с данными, загруженными из cookie-файлов, в которых отсутствует лишь имя переданного файла

Пользовательский cookie-файл хранит строку данных, введенную пользователем в поле формы “Enter cookie value”, а имя пользователя, известные ему языки и имена выгруженных файлов сохраняются в информационном cookie-файле.

После того как в сценарии обнаруживается, что данные формы отсутствуют, проводится подготовка к выводу страницы формы, но перед этим происходит чтение на клиентском компьютере cookie-файлов (которые автоматически передаются клиентом после того, как пользователь щелкает на ссылке) и заполнение формы. Поэтому в форме, отображаемой в конечном итоге, как по волшебству появляется вся ранее введенная пользователем информация.

Несомненно, это приложение заинтересует читателя, поэтому его код представлен в примере 10.7.

### Пример 10.7. Расширенное приложение CGI (advcgi.py)

В этом сценарии содержится один основной класс AdvCGI.py, имеющий немного более широкое назначение. В нем представлены методы, предназначенные для отображения страниц с формой, сообщением об ошибке или результатами, а также методы чтения или записи cookie-файлов на клиентском компьютере (где развернут веб-браузер).

```

1 #!/usr/bin/env python
2
3 from cgi import FieldStorage
4 from os import environ
5 from cStringIO import StringIO
6 from urllib import quote, unquote
7
8 class AdvCGI(object):
9 header = 'Content-Type: text/html\n\n'
10 url = '/cgi-bin/advcgi.py'
11
12 formhtml = '''<HTML><HEAD><TITLE>
13 Advanced CGI Demo</TITLE></HEAD>
14 <BODY><H2>Advanced CGI Demo Form</H2>
15 <FORM METHOD=post ACTION="%s" ENCTYPE="multipart/form-data">
16 <H3>My Cookie Setting</H3>
17 <CODE>CPPuser = %s</CODE>
18 <H3>Enter cookie value

19 <INPUT NAME=cookie value="%s"> (<I>optional</I>)</H3>
20 <H3>Enter your name

21 <INPUT NAME=person VALUE="%s"> (<I>required</I>)</H3>
22 <H3>What languages can you program in?
23 (<I>at least one required</I>)</H3>
24 %s
25 <H3>Enter file to upload <SMALL>(max size 4K)</SMALL></H3>
26 <INPUT TYPE=file NAME=upfile VALUE="%s" SIZE=45>
27 <P><INPUT TYPE=submit>
28 </FORM></BODY></HTML>'''
29
30 langSet = ('Python', 'Ruby', 'Java', 'C++', 'PHP', 'C', 'JavaScript')
31 langItem = '<INPUT TYPE=checkbox NAME=lang VALUE="%s"%s %s\n'
32
33 def getCPPCookies(self): # считываем cookie-файлы у клиента
34 if 'HTTP_COOKIE' in environ:
35 cookies = [x.strip() for x in environ['HTTP_COOKIE'].split(';')]
36 for eachCookie in cookies:
37 if len(eachCookie)>6 and eachCookie[:3]=='CPP':
38 tag = eachCookie[3:7]
39 try:
40 self.cookies[tag] = eval(unquote(eachCookie[8:]))
41 except (NameError, SyntaxError):
42 self.cookies[tag] = unquote(eachCookie[8:])
43 if 'info' not in self.cookies:
44 self.cookies['info'] = ''
45 if 'user' not in self.cookies:
46 self.cookies['user'] = ''

```

```

47 else:
48 self.cookies['info'] = self.cookies['user'] = ''
49
50 if self.cookies['info'] != '':
51 self.who, langStr, self.fn = self.cookies['info'].split(':')
52 self.langs = langStr.split(',')
53 else:
54 self.who = self.fn = ' '
55 self.langs = ['Python']
56
57 def showForm(self):
58 self.getCPPCookies()
59
60 # объединяем флажки выбора языка
61 langStr = []
62 for eachLang in AdvCGI.langSet:
63 langStr.append(AdvCGI.langItem % (eachLang,
64 ' CHECKED' if eachLang in self.langs else '',
65 eachLang))
66
67 # Проверяем, установлены ли еще cookie-файлы пользователя
68 if not ('user' in self.cookies and self.cookies['user']):
69 cookStatus = '<I>(cookie has not been set yet)</I>'
70 userCook = ''
71 else:
72 userCook = cookStatus = self.cookies['user']
73
74 print '%s%s' % (AdvCGI.header, AdvCGI.formhtml % (
75 AdvCGI.url, cookStatus, userCook, self.who,
76 ''.join(langStr), self.fn))
77
78 errhtml = '''<HTML><HEAD><TITLE>
79 Advanced CGI Demo</TITLE></HEAD>
80 <BODY><H3>ERROR</H3>
81 %s<P>
82 <FORM><INPUT TYPE=button VALUE=Back
83 ONCLICK="window.history.back()"></FORM>
84 </BODY></HTML>'''
85
86 def showError(self):
87 print AdvCGI.header + AdvCGI.errhtml % (self.error)
88
89 reshtml = '''<HTML><HEAD><TITLE>
90 Advanced CGI Demo</TITLE></HEAD>
91 <BODY><H2>Your Uploaded Data</H2>
92 <H3>Your cookie value is: %s</H3>
93 <H3>Your name is: %s</H3>
94 <H3>You can program in the following languages:</H3>
95 %s
96 <H3>Your uploaded file...

97 Name: <I>%s</I>

98 Contents:</H3>
99 <PRE>%s</PRE>
100 Click here to return to form.
101 </BODY></HTML>'''
102
103 def setCPPCookies(self): # просим клиента сохранить cookie-файлы
104 for eachCookie in self.cookies.keys():
105 print 'Set-Cookie: CPP%s%s; path=/' % \

```

```

106 (eachCookie, quote(self.cookies[eachCookie]))
107
108 def doResults(self): # выводим итоговую страницу
109 MAXBYTES = 4096
110 langList = ''.join(
111 '%s
' % eachLang for eachLang in self.langs)
112 filedata = self.fp.read(MAXBYTES)
113 if len(filedata) == MAXBYTES and f.read():
114 filedata = '%s%s' % (filedata,
115 '... <I>(file truncated due to size)</I>')
116 self.fp.close()
117 if filedata == '':
118 filedata = '<I>(file not given or upload error)</I>'
119 filename = self.fn
120
121 # Проверяем, установлены ли еще cookie-файлы пользователя
122 if not ('user' in self.cookies and self.cookies['user']):
123 cookStatus = '<I>(cookie has not been set yet)</I>'
124 userCook = ''
125 else:
126 userCook = cookStatus = self.cookies['user']
127
128 # устанавливаем cookie-файлы
129 self.cookies['info'] = ':'.join(
130 (self.who, '', '.join(self.langs, ','), filename))
131 self.setCPPCookies()
132
133 print '%s%s' % (AdvCGI.header, AdvCGI.reshtml % (
134 cookStatus, self.who, langList,
135 filename, filedata, AdvCGI.url))
136
137 def go(self): # определяем, какую страницу вернуть
138 self.cookies = {}
139 self.error = ''
140 form = FieldStorage()
141 if not form.keys():
142 self.showForm()
143 return
144
145 if 'person' in form:
146 self.who = form['person'].value.strip().title()
147 if self.who == '':
148 self.error = 'Your name is required. (blank)'
149 else:
150 self.error = 'Your name is required. (missing)'
151
152 self.cookies['user'] = unquote(form['cookie'].value.strip()) if
'cookie' in form else ''
153 if 'lang' in form:
154 langData = form['lang']
155 if isinstance(langData, list):
156 self.langs = [eachLang.value for eachLang in langData]
157 else:
158 self.langs = [langData.value]
159 else:
160 self.error = 'At least one language required.'
161
162 if 'upfile' in form:
163 upfile = form['upfile']

```

---

```

164 self.fn = upfile.filename or ''
165 if upfile.file:
166 self.fp = upfile.file
167 else:
168 self.fp = StringIO('no data')
169 else:
170 self.fp = StringIO('no file')
171 self.fn = ''
172
173 if not self.error:
174 self.doResults()
175 else:
176 self.showError()
177
178 if __name__ == '__main__':
179 page = AdvCGI()
180 page.go()

```

---

Сценарий `advcgi.py` весьма похож на сценарии `CGI friendsC.py`, которые рассматривались ранее в этой главе. Он также возвращает страницы с формой, результатами и сообщениями об ошибках. В новом сценарии, кроме дополнительных функций CGI, введены некоторые объектно-ориентированные средства. В частности, вместо некоторых функций используются методы класса. Текст HTML страниц теперь представлен в виде статических данных для класса, а это означает, что текст будет оставаться неизменным во всех экземплярах; причем не имеет значения, что в рассматриваемом примере фактически применяется только один экземпляр.

## Построчное объяснение

### Строки 1–6

Здесь представлены обычные строки начала сценария и импорта. Для тех, кто не знаком с классом `StringIO`, отметим, что это структура данных, подобная файлу, основным элементом которой является строка; этот аналог файла можно рассматривать как поток текстовых данных в оперативной памяти.

В версии Python 2 этот класс представлен в модуле `StringIO` и в эквивалентном ему модуле на языке C, `cStringIO`. В версии Python 3 этот класс перемещен в пакет `io`. Аналогичным образом функции `urllib.quote()` и `urllib.unquote()` версии Python 2 перемещены в Python 3 в пакет `urllib.parse`.

### Строки 8–28

После объявления класса `AdvCGI` создаются переменные `header` и `url` (статический класс) для использования в методах, обеспечивающих вывод всех разнообразных страниц. Далее следуют статический текст HTML формы, определение языка программирования и элемент HTML для каждого языка.

### Строки 33–55

В этом примере используются cookie-файлы. Далее в приложении представлен метод `setCPPCookies()`, который служит для отправки cookie-файлов (с веб-сервера) в браузер и сохранения их на клиентском компьютере.

Метод `getCPPOokies()` выполняет противоположные действия. При выполнении браузером последующих вызовов в приложении этот метод отправляет ранее созданные cookie-файлы назад на сервер с помощью заголовков HTTP. При выполнении приложения доступ к этим значениям (из приложения) обеспечивается с помощью переменной окружения `HTTP_COOKIE`.

Метод выполняет интерпретацию cookie-файлов и поиск тех из них, имена которых начинаются с CPP (строка 37). В данном приложении нас интересуют cookie-файлы с именами `CPPuser` и `CPPinfo`. В строке 38 извлекаются ключи `'user'` и `'info'`, знак равенства в позиции 7 пропускается, значение, начинающееся с позиции 8, избавляется от кавычек, затем в строках 39–42 происходит формирование объекта Python. С помощью обработчика исключений производится поиск полезных данных cookie-файла, не являющихся действительными объектами Python, после чего полученное строковое значение просто сохраняется. Если какой-либо из cookie-файлов отсутствует, значениям данных присваивается пустая строка (строки 43–48). Метод `getCPPOokies()` вызывается только из `showForm()`.

В этом простом примере мы сами интерпретируем cookie-файлы, но в более сложных случаях может потребоваться использовать для выполнения этой задачи модуль `Cookie` (переименованный в Python 3 в `http.cookies`).

Аналогичным образом, если при разработке веб-клиентов потребуется организовать управление всеми cookie-файлами, хранимыми в браузере (в архиве cookie-файлов), и связь с веб-серверами, то возникнет необходимость использовать модуль `cookielib` (переименованный в Python 3 в `http.cookiejar`).

## Строки 57–76

Метод `checkUserCookie()` используется и в `showForm()`, и в `doResults()` для проверки того, было ли задано пользовательское значение cookie-файла. Это значение отображается и в форме, и в шаблонах HTML результатов.

Единственное назначение метода `showForm()` состоит в отображении формы для пользователя. В этом методе применяется `getCPPOokies()` для получения cookie-файлов из предыдущих запросов (если таковые имеются) и прорисовки формы должным образом.

## Строки 78–87

В данном блоке кода формируются также страницы с сообщениями об ошибках.

## Строки 89–101

Это просто шаблон HTML для страницы с результатами. Он используется в методе `doResults()`, который задает все требуемые данные.

## Строки 102–135

С применением этих блоков кода создается страница с результатами. Метод `setCPPOokies()` сохраняет cookie-файлы для приложения клиентским компьютером, а метод `doResults()` собирает все данные и отправляет вывод назад клиенту.

Последний метод, вызываемый из метода `go()`, выполняет основную работу по подготовке выходных данных. В первом блоке кода в этом методе (строки 109–119) обрабатываются данные, введенные пользователем: набор выбранных языков программирования (должен быть выбран по крайней мере один; см. метод `go()`), переданный

файл и предоставляемое пользователем значение cookie-файла; последние два параметра являются необязательными.

В завершение метод `doResults()` (строки 128–135) вносит все необходимые данные в отдельный cookie-файл (CPPinfo) для использования в будущем, а затем прорисовывает шаблон результатов со всеми данными.

## Строки 137–180

Сценарий начинается с создания объекта страницы `AdvCGI` и вызова его метода `go()`, который запускает обработку. Метод `go()` читает все входящие данные и решает, какая страница должна быть показана.

Если не задано имя или не выбран ни один язык, отображается страница с сообщением об ошибке. Если отсутствуют входные данные, вызывается метод `showForm()` для вывода формы; в противном случае вызывается метод `doResults()` для отображения страницы с результатами. Для активизации ошибок передается переменная `self.error`, которая выполняет две задачи. Она позволяет описать причину ошибки в виде строки и выполняет роль флага, указывающего, что произошла ошибка. Если это значение не пусто, пользователь перенаправляется к странице с ошибкой.

Обработка поля с личными сведениями (строки 145–150) выполняется так же, как и в предыдущих примерах: в виде отдельной пары “ключ–значение”. Действия по получению данных о языке (строки 153–160) немного сложнее, поскольку необходимо проверить либо экземпляр (`Mini`) `FieldStorage`, либо список таких экземпляров. В этом сценарии с данной целью применяется уже знакомая встроенная функция `isinstance()`. В конечном итоге формируется список из одного или нескольких имен языков, в зависимости от выбора, сделанного пользователем.

Этот пример использования cookie-файлов для хранения данных показывает, что cookie-файлы позволяют полностью избавиться от необходимости передачи полей по технологии CGI. В предыдущих примерах этой главы передача таких значений осуществлялась в виде переменных CGI. Теперь используются только cookie-файлы. Кроме того, приведенный здесь код показывает, что к получению требуемых данных вообще не привлекаются какие-либо средства обработки CGI, а это означает, что данные не поступают из объекта `FieldStorage`. Данные передаются на сервер веб-клиентам в каждом запросе, и для получения значений (не только данных, выбранных пользователем, но и ранее введенной информации, которая будет служить при следующем заполнении форм) применяются cookie-файлы.

Для получения данных, вновь введенных пользователем, служит метод `showResults()`, и этот же метод формирует cookie-файлы, например, с помощью вызова `setCPPCookies()`. Для отображения страницы формы с результатами текущего выбора пользователя метод `showForm()` должен считывать значения из cookie-файлов. Для этого в нем выполняется вызов метода `getCPPCookies()`.

Наконец, проводится подготовка к передаче файла (строки 162–171). Независимо от того, будет ли действительно проводиться передача файла, в `FieldStorage` в атрибуте файла передается дескриптор файла. Как показано в строке 171, если не задано имя файла, то в качестве его значения задается пустая строка. Лучший вариант может состоять в том, что производится доступ к указателю файла (атрибуту файла) и, возможно, чтение по одной строке одновременно или постепенная обработка другого рода.

В данном случае передача файлов составляет лишь часть действий по пересылке данных пользователем, поэтому для извлечения данных из файла указатель файла просто передается в функцию `doResults()`. С помощью `doResults()` для экономии



памяти отображаются только первые 4 КБайт содержимого файла (как указано в строке 112). Этот пример служит также демонстрацией того, как избежать необходимости отображать, допустим, двоичный файл объемом 4 ГБайт.

## 2.5

Читатели, уже знакомые с книгой *Core Python*, заметят, что код сценария подвергся существенному пересмотру по сравнению с предыдущими изданиями. Исходная версия была написана больше десяти лет тому назад и не отражала современные достижения Python. Вполне возможно даже, что эта версия `advcgi.py` не будет работать в версиях Python, предшествующих 2.5. На веб-сайте книги можно найти не только код из предыдущих изданий книги, но и эквивалентный вариант для версии Python 3.

## 10.6. Введение в WSGI

В этом разделе представлены вводные сведения о том, что нужно знать о технологии WSGI, начиная с идей, лежащих в ее основе, и кончая историей создания. Далее в этом разделе описано, как создавать веб-приложения, надежно эксплуатируемые в любых условиях.

### 10.6.1. Стимулы к дальнейшему развитию (альтернативы технологии CGI)

Выше было показано, что такое технология CGI и с чем связана необходимость в применении подобных технологий: серверы не могут создавать динамическое содержимое; они не имеют в своем распоряжении информации о конкретных пользовательских приложениях, необходимой для проверки подлинности, ведения банковских счетов, организации онлайн-торговли и т.д. Для того чтобы выполнить эти специализированные задания, веб-серверы должны обмениваться данными с внешними процессами.

Кроме того, в этой главе показано, как эти проблемы решаются с помощью технологии CGI и как она работает. Было также указано, что эта технология не имеет перспектив дальнейшего развития, поскольку не обеспечивает масштабирования; процессы CGI создаются для обработки каждого отдельного запроса, а затем уничтожаются (это можно сравнить с тем, как выполняются сценарии интерпретатором Python). Если в приложение должны приходить тысячи запросов и в ответ на это придется запускать такое же количество интерпретаторов, то будет создана такая нагрузка, с которой не справится ни один сервер. В связи с этим получили широкое распространение следующие два метода решения указанной проблемы производительности: интеграция сервера и внешние процессы. Ниже приведено краткое описание этих методов.

### 10.6.2. Интеграция сервера

Метод интеграции сервера известен также как применение *API сервера*. Этот метод реализован в виде таких собственных фирменных решений, как NSAPI (Netscape Server Application Programming Interface — интерфейс прикладного программирования сервера Netscape) и ISAPI (Internet Server Application Programming Interface — интерфейс прикладного программирования интернет-сервера) корпорации Microsoft. Наибольшее распространение в наши дни (начиная с середины 1990-х годов) получил

веб-сервер HTTP *Apache* с открытым исходным кодом. Этот веб-сервер также имеет интерфейс прикладного программирования (Application Programming Interface — API) и подключает *модули* (компилируемые сменные компоненты), позволяющие расширять набор его функций и возможностей.

Три перечисленных выше решения и подобные им устраняют основной недостаток CGI, низкую производительность, поскольку в них шлюз обмена данными встроен в сервер. Иными словами, сервер не запускает отдельный процесс для обработки запроса, а просто выполняет вызов функции, запуская тем самым прикладной код, и после этого вырабатывает ответ. В этих серверах применяемый интерфейс прикладного программирования может предусматривать выполнение работы с помощью ряда предварительно созданных подпроцессов, или потоков. Многие из них обеспечивают настройку в соответствии с конкретными требованиями поддерживаемых приложений. Серверы обычно поддерживают также такие функции, как сжатие, безопасность, поддержка прокси и виртуальный хостинг.

Безусловно, этот подход, как и любой другой, имеет свои недостатки. В частности, при использовании интерфейса прикладного программирования часто обнаруживается ошибочный код, применение которого снижает производительность сервера, не достигается полная совместимость применяемых языков, в связи с чем разработчики API вынуждены использовать такой же язык программирования, как и в реализации веб-сервера, а если API сервера с открытым исходным кодом не применяется, то приходится встраивать модули расширения в код, права на который принадлежат другой компании, поэтому должно соблюдаться требование потоковой безопасности приложений. Есть и другие недостатки.

### 10.6.3. Внешние процессы

Еще одним решением является применение внешних процессов. Такие процессы можно сравнить с приложениями CGI, которые работают постоянно вне серверного процесса. При поступлении запроса сервер передает его внешнему процессу. Такая организация работы обеспечивает лучшее масштабирование по сравнению с технологией CGI в чистом виде, поскольку внешние процессы действуют постоянно, а не порождаются для обработки отдельных запросов и последующего завершения. Наиболее широко известным решением на основе внешних процессов является FastCGI. Внешние процессы позволяют достичь тех же преимуществ, что и API сервера, но количество недостатков меньше. Например, из того, что внешние процессы выполняются вне сервера, следует, что код соответствующих модулей может быть реализован на любом языке программирования, сбой в приложении не затрагивает веб-сервер, не приходится вести разработку с учетом необходимости интеграции с кодом, принадлежащим другой компании, и т.д.

Безусловно, реализация FastCGI предусмотрена и на языке Python. Имеется также ряд модулей Python для Apache (PyApache, mod\_snake, mod\_python и т.д.), но некоторые из них больше не поддерживаются. Дополнительно к этому предусмотрено оригинальное решение исключительно в рамках технологии CGI, а это означает, что язык Python предоставляет целую гамму средств поддержки веб-приложений, в том числе на основе шлюза API для веб-сервера.

Тем не менее во всех этих инструментах предусмотрены разные механизмы вызова, поэтому разработчику приходится преодолевать затруднения, связанные с выбором наиболее подходящего решения. Приходится не только разрабатывать приложение, но и выбирать способ интеграции с разнообразными веб-серверами.

В действительности, прежде чем приступать к написанию веб-приложения, необходимо заранее точно выбрать один из механизмов, с помощью которого оно будет эксплуатироваться, и учитывать это в коде.

Эта проблема приобретает наибольшую остроту для разработчиков веб-платформ, поскольку они стремятся предоставить своим будущим пользователям максимальную гибкость. Чтобы при создании веб-приложений не приходилось подготавливать несколько версий для каждого типа сервера, применяемая веб-платформа должна предоставлять интерфейсы ко всем серверным решениям. Очевидно, что выполнение такого требования не может быть обеспечено исключительно средствами языка Python, поэтому был создан стандарт интерфейса шлюза веб-сервера (Web Server Gateway Interface — WSGI).

### 10.6.4. Основные сведения о стандарте WSGI

Стандарт WSGI не определяет ни сервер, или API, для которого ведется программирование, ни сам прикладной код. Этот стандарт задает лишь интерфейс. Спецификация WSGI подготовлена в ответ на предложение PEP 333 от 2003 года для стимулирования широкого распространения разнородных веб-платформ, веб-серверов и различных стилей вызова, о которых шла речь выше в этой главе (CGI в чистом виде, API сервера, внешние процессы).

Задача состояла в том, чтобы добиться функциональной совместимости и модульности на основе стандарта, направленного на создание общего API между веб-сервером и разными уровнями веб-платформы. За время, прошедшее после его создания, стандарт WSGI стал общепризнанным. Теперь почти все веб-серверы на основе Python совместимы с WSGI. Наличие такого стандарта, как WSGI, идет на пользу разработчикам приложений, помогает успешно справляться со своей работой создателям платформ, а также благоприятно для всего сообщества в целом.

Приложение WSGI определяется как вызываемый код, который всегда принимает следующие параметры: словарь переменных окружения сервера и еще один вызываемый фрагмент кода, который инициализирует подготовку ответа с кодом статуса HTTP и заголовками HTTP для возврата клиенту. Этот вызываемый код должен возвращать итерируемый объект, который составляет полезные данные.

В приведенном ниже приложении “Hello World” на основе WSGI эти переменные именуются соответственно `environ` и `start_response()`:

```
def simple_wsgi_app(environ, start_response):
 status = '200 OK'
 headers = [('Content-type', 'text/plain')]
 start_response(status, headers)
 return ['Hello world!']
```

Переменная `environ` содержит знакомые переменные окружения, такие как `HTTP_HOST`, `HTTP_USER_AGENT`, `SERVER_PROTOCOL` и т.д. Вызываемый код `start_response()`, который должен быть выполнен в приложении, подготавливает ответ, отправляемый в конечном итоге назад клиенту. Ответ должен включать код возврата HTTP (200, 300 и т.д.), а также заголовки ответа HTTP.

В этой первой версии стандарта WSGI, `start_response()` должен также возвращать функцию `write()`, которая требуется для серверов прежних версий, возвращающих результаты в виде потока. Рекомендуется отказаться от использования этой функции и возвращать только итерируемый объект, чтобы все операции по возврату результатов клиенту мог выполнять сам веб-сервер (это позволяет избавиться от

необходимости возлагать часть операций обработки возвращаемых данных на приложение, поскольку не все приложения к этому готовы). Если эта рекомендация не учитывается, то большинство приложений просто уничтожают значение, возвращаемое от `start_response()`, или не используют его и не сохраняют.

В предыдущем примере можно видеть, что устанавливается код статуса 200 и задается заголовок `Content-Type`. И то и другое передается в функцию `start_response()`, что позволяет формально приступить к подготовке ответа. За этим должна следовать подготовка итерируемого объекта, такого как список, генератор и т.д., иными словами, формирование полезных данных ответа. В этом примере происходит только возврат списка, содержащего одну строку, но ничто не препятствует применению такого кода, который возвращает намного больше данных. Кроме того, итерируемым объектом может быть не только список; превосходными альтернативами могут служить генератор или вызываемый экземпляр.

Наконец, в отношении кода `start_response()` можно отметить, что предусмотрен третий и необязательный параметр с информацией исключения, который обычно принято указывать по заданному для него сокращению, `exc_info`. Если в приложении заданы заголовки, скажем, с кодом "200 OK" (но фактически еще не отправлены), а затем во время выполнения обнаружены ошибки, то при желании можно изменить заголовки, установив в них другой код, например "403 Forbidden" или "500 Internal Server Error".

Для этого в приложении необходимо в начале выполнения вызвать функцию `start_response()`, как обычно, с двумя параметрами. При возникновении ошибок `start_response()` можно вызвать еще раз, но с параметром `exc_info`, новой информацией о состоянии и другими заголовками, которые заменят существующие.

Вызов функции `start_response()` во второй раз без параметра `exc_info` является ошибкой. Еще раз отметим, что все вызовы указанной функции должны происходить еще до отправки заголовков HTTP. Если заголовки уже отправлены, должно быть активизировано исключение, такое как `raise exc_info[0], exc_info[1]` или `exc_info[2]`.

Дополнительные сведения о вызове функции `start_response()` см. в документе PEP 333 по адресу <http://www.python.org/dev/peps/pep-0333/#the-start-response-callable>.

### 10.6.5. Серверы WSGI

Сервер должен действовать так: вызвать приложение, передать переменные окружения и вызываемую функцию `start_response()`, затем ожидать завершения приложения. После этого в качестве возвращаемого значения должен быть получен итерируемый объект, и эти данные должны быть отправлены назад клиенту. В следующем сценарии представлен упрощенный и ограниченный пример организации работы веб-сервера WSGI:

```
import StringIO
import sys

def run_wsgi_app(app, environ):
 body = StringIO.StringIO()

 def start_response(status, headers):
 body.write('Status: %s\r\n' % status)
 for header in headers:
```

```

 body.write('s: %s\r\n' % header)
 return body.write

iterable = app(envIRON, start_response)
try:
 if not body.getvalue():
 raise RuntimeError("start_response() not called by app!")
 body.write('\r\n%s\r\n' % '\r\n'.join(line for line in iterable))
finally:
 if hasattr(iterable, 'close') and callable(iterable.close):
 iterable.close()

sys.stdout.write(body.getvalue())
sys.stdout.flush()

```

Базовый сервер/шлюз получает приложение в том виде, в котором оно подготовлено разработчиком, помещает приложение в оперативную память и передает ему словарь `environ` с содержимым `os.environ()`, переменные окружения `wsgi.*`, предусмотренные технологией WSGI (см. PEP, но, как правило, `wsgi.input`, `wsgi.errors`, `wsgi.version` и т.д.), а также все переменные окружения платформы или промежуточного программного обеспечения. (Дополнительные сведения о промежуточном программном обеспечении приведены ниже.) Затем сервер вызывает функцию `run_wsgi_app()`, передавая ей приложение и определение окружения, а эта функция возвращает ответ клиенту.

В действительности для разработчика приложения эти нюансы организации работы не представляют особого интереса. Серверы WSGI создаются для предоставления единообразной платформы выполнения для приложений, для которых в качестве основы выбрана технология WSGI. Как показывает предыдущий пример, WSGI позволяет провести четкий водораздел между прикладной и серверной частями исполняемого кода. На сервер, описанный выше (или любой другой сервер WSGI), можно передать любое приложение, которое соответствует технологии WSGI. Аналогично, каким бы ни было приложение, при его разработке можно не задумываться над тем, какой сервер будет применяться для его эксплуатации. Необходимо лишь позаботиться о передаче требуемых переменных окружения и вызываемого кода `start_response()`, который должен быть выполнен перед возвратом данных клиенту.

### 10.6.6. Эталонный сервер

Как уже было сказано, разработчики приложений не должны заниматься также разработкой серверов. Это означает, что они не обязаны создавать и сопровождать код наподобие `run_wsgi_app()`, а должны иметь возможность выбрать для себя любой сервер WSGI. Если же не остается ничего иного, то можно воспользоваться простым эталонным сервером из стандартной библиотеки Python: `wsgiref.simple_server.WSGIServer`.

Сервер может быть создан непосредственно с использованием этого класса, но в пакете `wsgiref` предусмотрена вспомогательная функция `make_server()`, с помощью которой можно упростить создание эталонного сервера. Рассмотрим ее применение на примере приведенного выше приложения `simple_wsgi_app()`:

```

#!/usr/bin/env python

from wsgiref.simple_server import make_server

```

```

httpd = make_server('', 8000, simple_wsgi_app)
print "Started app serving on port 8000..."
httpd.serve_forever()

```

В данном случае берется созданное ранее приложение `simple_wsgi_app()`, заключается в оболочку в виде сервера, работающего в порту 8000, после чего производится запуск цикла сервера. При посещении в браузере адреса `http://localhost:8000` (или другого используемого сервера, адрес которого определяется сочетанием [узел, порт]) должен быть получен ответ в виде текста "Hello World!".

Чтобы еще больше упростить для себя знакомство с этой технологией, можно отказаться от написания не только сервера, но и приложения. В модуле `wsgiref` имеется также демонстрационное приложение `wsgiref.simple_server.demo_app()`. Функция `demo_app()` почти идентична `simple_wsgi_app()`, если не считать того, что дополнительно в ней предусмотрен вывод переменных окружения. Ниже приведен код вызова демонстрационного приложения с эталонным сервером.

```

#!/usr/bin/env python

from wsgiref.simple_server import make_server, demo_app

httpd = make_server('', 8000, demo_app)
print "Started app serving on port 8000..."
httpd.serve_forever()

```

Запустите сервер CGI, а затем направьте браузер по адресу приложения; на экране должен появиться вывод "Hello World!" вместе со списком переменных окружения.

Еще раз отметим, что это лишь эталонная модель сервера, совместимого с WSGI. Он не обладает полным набором функций и не предназначен для использования на производстве. Будущие создатели серверов могут взять его за основу для разработки собственных программных продуктов и обеспечить их совместимость с технологией WSGI. То же касается функции `demo_app()`, которая может служить эталоном приложения, совместимого с WSGI, для разработчиков приложений.

### 10.6.7. Примеры приложений WSGI

Как было отмечено ранее, теперь технология WSGI определена стандартом, и ее поддерживают почти все веб-платформы Python, даже если это не очевидно на первый взгляд. Например, класс обработчика App Engine Google, даже при обычном составе импортируемых модулей, может содержать код, который выглядит примерно так:

```

class MainHandler(webapp.RequestHandler):
 def get(self):
 self.response.out.write('Hello world!')

application = webapp.WSGIApplication([
 ('/', MainHandler)], debug=True)
run_wsgi_app(application)

```

Не на всех платформах можно обнаружить такое точное соответствие, как в данном случае, но всегда четко усматривается ссылка на WSGI. Чтобы провести более

подробное сравнение, можно перейти на один уровень ниже и рассмотреть функцию `run_bare_wsgi_app()`, которая находится в модуле `util.py` подпакета `webapp` в SDK App Engine Python. Вполне очевидно, что этот код весьма напоминает нечто производное от `simple_wsgi_app()`.

### 10.6.8. Промежуточное программное обеспечение и создание оболочек для приложений WSGI

Иногда возникает необходимость, не внося изменений в само приложение, ввести дополнительную предобработку или постобработку перед его выполнением (на этапе запроса) или после его выполнения (на этапе ответа). Для этого применяется так называемое промежуточное программное обеспечение, представляющее собой дополнительное функциональное средство, располагающееся между веб-сервером и веб-приложением. Промежуточное программное обеспечение позволяет, допустим, обработать данные, поступающие от пользователя, перед передачей их в приложение или провести некоторые заключительные корректировки результатов, полученных от приложения, перед возвратом полезных данных пользователю. Промежуточное программное обеспечение часто называют *надстройкой*, подразумевая под этим, что в основе лежит приложение, а подобные программные средства создают лишь дополнительные уровни.

Предварительная обработка может обеспечивать перехват параметров запроса, их модификацию, добавление или удаление, внесение изменений в окружение (включая все переданные пользователем переменные формы [CGI]), использование пути URL для планирования применения функций приложения, направление или перенаправление запросов, балансирование нагрузки с учетом сетевого трафика через входящий IP-адрес клиента, делегирование уточненных функциональных возможностей (например, использование заголовка `User-Agent` для перенаправления пользователей мобильных устройств к упрощенному пользовательскому интерфейсу/приложению) и т.д.

К примерам постобработки прежде всего относится манипулирование выводом приложения. В следующем сценарии показан сервер, аналогичный серверу отметок времени, который рассматривался в главе 2. В этом примере перед каждой строкой результатов в приложении выводится префикс в виде отметки времени. Безусловно, на практике соответствующий сценарий должен быть гораздо сложнее, но этот пример, как и другие, предназначен лишь для демонстрации основных особенностей приложения. В данном случае вызов `simple_wsgi_app()` заключается в оболочку в виде функции `ts_simple_wsgi_app()`, а эта функция устанавливается как приложение, регистрируемое сервером:

```
#!/usr/bin/env python

from time import ctime
from wsgiref.simple_server import make_server

def ts_simple_wsgi_app(environ, start_response):
 return ('[%s] %s' % (ctime(), x) for x in \
 simple_wsgi_app(environ, start_response))

httpd = make_server('', 8000, ts_simple_wsgi_app)
print "Started app serving on port 8000..."
httpd.serve_forever()
```

Читатели, которые стремятся преимущественно использовать средства объектно-ориентированного программирования, могут применить оболочку класса вместо оболочки функции. Следует также отметить, что два параметра, `environ` и `start_response()`, в вызове можно заменить одним кортежем с переменным числом элементов (см. приведенный далее пример использования переменной `stuff`) для уменьшения объема кода, поскольку он немного увеличился в связи с включением определения класса и нескольких методов:

```
class Ts_ci_wrapp(object):
 def __init__(self, app):
 self.orig_app = app

 def __call__(self, *stuff):
 return ('[%s] %s' % (ctime(), x) for x in
 self.orig_app(*stuff))

httpd = make_server('', 8000, Ts_ci_wrapp(simple_wsgi_app))
print "Started app serving on port 8000..."
httpd.serve_forever()
```

Классу присвоено имя `Ts_ci_wrapp`, представляющее собой сокращение от “timestamp callable instance wrapped application” (приложение с оболочкой в виде экземпляра вызываемой функции для работы с отметкой времени). Экземпляр создается при создании сервера. В ходе инициализации берется исходное приложение и кэшируется для дальнейшего использования. При выполнении приложения сервер, как и прежде, передает словарь `environ` и вызываемую функцию `start_response()`. Внесенное изменение позволяет вызывать сам экземпляр (поэтому и применяется соответствующее определение метода `__call__()`). Параметры `environ` и `start_response()` передаются в исходное приложение через параметр `stuff`.

В данном случае используется вызываемый экземпляр, а перед этим — функция, но следует учитывать, что приемлемым является любой вызываемый код. Следует также отметить, что в последних нескольких примерах функция `simple_wsgi_app()` каждый раз применялась без каких-либо изменений. Важная особенность технологии WSGI состоит в том, что она позволяет провести четкий водораздел между веб-приложением и веб-сервером. Это способствует лучшему распределению обязанностей между разработчиками, позволяет привлекать сразу несколько коллективов и распределять между ними работу, а также предоставляет единообразный и гибкий способ подготовки веб-приложений к эксплуатации под управлением любого сервера, совместимого с WSGI. Кроме того, разработчики веб-сервера избавляются от необходимости включать какие-либо определяемые пользователем или специализированные обработчики прерываний для тех прикладных программистов, которые предпочитают эксплуатировать свои приложения с использованием собственного программного обеспечения веб-сервера.

### 10.6.9. Изменения в интерфейсе WSGI в версии Python 3

3.x

В документе PEP 333 определен стандарт WSGI для Python 2, а в документе PEP 3333 предложены усовершенствования для PEP 333, которые позволяют распространить этот стандарт на Python 3. В частности, в последнем документе указано, что весь сетевой трафик осуществляется с применением байтовой организации передаваемых данных. В Python 2 предусмотрена



собственная поддержка байтовых строк, а в версии Python 3 строки по умолчанию имеют формат Юникода, поскольку в этой версии для представления текстовых данных применяется именно эта кодировка. С другой стороны, теперь в Python 3 текстовые строки (в байтовом представлении и в коде ASCII) определены как имеющие тип `bytes`.

В частности, документ PEP 3333 содержит разъяснение, что независимо от того, используется ли версия Python 2 или Python 3, для всех заголовков HTTP и соответствующих метаданных должен применяться изначально строковый тип данных `str`. Кроме того, в этом документе указано, что для полезных данных HTTP (запросов и ответов, входных данных GET/POST/PUT, вывода HTML и т.д.) должен применяться тип данных в виде байтовых строк. Дополнительные сведения о документе PEP 3333 можно найти в его определении по адресу [www.python.org/dev/peps/pep-3333/](http://www.python.org/dev/peps/pep-3333/).

Интересующиеся этой тематикой могут, кроме документа PEP 3333, ознакомиться с другими важными предложениями. Одним из них является документ PEP 444, представляющий собой первую попытку определить новую версию WSGI, которую можно условно назвать WSGI 2. Вообще говоря, сообщество разработчиков на языке Python рассматривает PEP 3333 как определение версии WSGI 1.0.1, или усовершенствование исходной спецификации PEP 333, а в PEP 444 речь идет о создании нового поколения WSGI.

## 10.7. Практическая разработка для веб

Технология CGI долгое время была основной “рабочей лошадкой”, и заложенные в ней концепции продолжают применяться в веб-программировании в наши дни. Вот почему мы уделили столько времени на ее изучение. С развитием WSGI появились новые перспективы создания программных средств производственного назначения.

Сегодня перед начинающими разработчиками веб-программ на Python открывается широкий набор возможностей. Кроме таких технологий, которые заняли прочное место среди веб-платформ, как Django, Pyramid и App Engine Google, есть еще так много вариантов выбора, что любой разработчик может найти для себя наиболее подходящий инструмент. При этом не обязательно даже прибегать к использованию платформ, поскольку есть возможность сразу же развернуть и применить веб-сервер, совместимый с WSGI, не привлекая дополнительные программные средства или платформы. Весьма велика вероятность того, что со временем разработчик перейдет к использованию одной из платформ, оценив все предоставляемые ею преимущества, такие как полный набор средств работы в Интернете.

Современная среда эксплуатации веб-приложений, как правило, включает в себя многопоточные или мультипроцессные серверы, имеющие цифровую подпись и безопасные cookie-файлы, основные средства проверки подлинности пользователей и управления сессиями. Вообще говоря, разработчики приложений уже регулярно используют эти средства. Функции проверки подлинности обеспечивают регистрацию пользователей с применением имени входа и пароля, cookie-файлы служат в качестве способа сопровождения сведений о пользователе и иногда информации о сессии и т.д. Еще одно важное направление деятельности состоит в обеспечении масштабирования приложений. Веб-серверы должны быть в состоянии обрабатывать запросы от большого числа пользователей, и в этом нельзя обойтись без применения потоков или процессов. Современные веб-технологии должны также обеспечивать поддержку сессий.

В этой главе весь прикладной код, эксплуатируемый с применением веб-серверов, наглядно показывает, что код сервера работает непрерывно и без ограничения по времени, а вызов веб-приложений (или, как в Java, сервлетов) происходит отдельно для каждого запроса. В коде не происходит сохранение информации о состоянии и, как уже было сказано, протокол HTTP также не поддерживает состояние. Иными словами, нельзя рассчитывать на то, что данные при переходе от одного запроса к другому будут автоматически сохранены в каких-то переменных, допустим, в глобальных. Поэтому каждый запрос должен рассматриваться как отдельная транзакция. Запрос поступает, обрабатывается, завершается, и все его следы исчезают, не оставляя ничего в программе.

Именно поэтому нужны средства управления сеансами, позволяющие сохранить пользовательскую информацию о состоянии от одного вызова к другому в течение четко определенного периода времени. Вообще говоря, такая задача решается с применением системы постоянного хранения данных того или иного типа на основе кэша памяти, плоских (или структурированных) файлов и даже базы данных. Несомненно, разработчики могут сами ввести в действие такую систему хранения данных, особенно если на них возложена реализация функций достаточно низкого уровня. Такие примеры также рассматривались в данной главе. Такие попытки можно назвать изобретением колеса, поскольку программное обеспечение управления сеансами уже входит в состав многих известных веб-платформ, включая Django. (Этой теме посвящена следующая глава.)

## 10.8. Связанные модули

В табл. 10.1 представлен список модулей, которые могут применяться для разработки веб-приложений. Другие полезные модули веб-приложений можно также найти в главе 3 и 13.

**Таблица 10.1.** Модули, применяемые в веб-приложениях

Модуль/пакет	Описание
<b>Веб-приложения</b>	
cgi	Получает данных формы CGI
cgitb <sup>c</sup>	Обрабатывает трассировки CGI
htmllib	Применявшееся ранее средство синтаксического анализа HTML для простых файлов HTML; класс HTML-Parser является расширением класса sgmlib.SGMLParser
HTMLparser <sup>c</sup>	Более новое, не основанное на SGML, средство синтаксического анализа для HTML и XHTML
htmlentitydefs	Общие определения компонентов HTML
Cookie	Серверные cookie-файлы для управления состоянием HTTP
cookielib <sup>c</sup>	Классы обработки cookie-файлов для клиентов HTTP
webbrowser <sup>b</sup>	Контроллер: запускает обработку веб-документов в браузере
sgmlib	Интерпретирует простые файлы SGML
robotparser <sup>a</sup>	Интерпретирует файлы robots.txt для анализа доступности URL
httplib <sup>a</sup>	Используется для создания клиентов HTTP

Модуль/пакет	Описание
<b>Веб-серверы</b>	
BaseHTTPServer	Абстрактный класс для разработки веб-серверов
SimpleHTTPServer	Позволяет обрабатывать простейшие запросы HTTP (HEAD и GET)
CGIHTTPServer	Кроме обработки таких веб-файлов, как SimpleHTTPServer, позволяет также обрабатывать запросы CG (HTTP POST)
http.server <sup>a</sup>	Новое имя для комбинированного пакета, объединяющего модули BaseHTTPServer, SimpleHTTPServer и CGIHTTPServer в Python 3
wsgiref <sup>f</sup>	Эталонный модуль WSGI
<b>Пакеты сторонних разработчиков (не находящиеся в стандартной библиотеке)</b>	
BeautifulSoup	Средство синтаксического анализа HTML и XML на основе регулярных выражений <a href="http://crummy.com/software/BeautifulSoup">http://crummy.com/software/BeautifulSoup</a>
html5lib	Средство синтаксического анализа HTML5 <a href="http://code.google.com/p/html5lib">http://code.google.com/p/html5lib</a>
lxml	Всесторонне развитое средство синтаксического анализа HTML и XML (поддерживает оба из вышеупомянутых средств синтаксического анализа), <a href="http://lxml.de">http://lxml.de</a>

<sup>a</sup> Новое в Python 1.6.<sup>e</sup> Новое в Python 2.4.<sup>b</sup> Новое в Python 2.0.<sup>f</sup> Новое в Python 2.5.<sup>c</sup> Новое в Python 2.2.<sup>g</sup> Новое в Python 3.0.<sup>d</sup> Новое в Python 2.3.

## 10.9. Упражнения

### CGI и веб-приложения

- 10.1.** *Модуль и файлы urllib.* Обновите сценарий friendsC.py, чтобы он сохранял имена и соответствующее количество друзей в виде текстового файла с двумя столбцами на диске и продолжал добавлять имена при каждом выполнении сценария.

**Дополнительные задания.** Добавьте код для вывода содержимого такого файла в веб-браузер (в формате HTML). Создайте ссылку для удаления всех имен в этом файле.

- 10.2.** *Проверка на наличие ошибок.* Сценарий friendsC.py сообщает об ошибке, если не выбран переключатель для указания количества друзей. Обновите сценарий CGI, чтобы он также сообщал об ошибке, если не введено имя (например, если поле содержит пустое значение или пробел).

**Дополнительное задание.** До сих пор мы рассматривали проверку на наличие ошибок с применением лишь серверного кода. Изучите вопросы программирования и реализации приложений на языке JavaScript с проверкой на наличие ошибок в клиентской части путем создания кода JavaScript для проверки того, возникают ли те или иные случаи ошибок, чтобы эти ошибки были устранены, прежде чем они достигнут сервера.

- 10.3.** *Простое приложение CGI.* Создайте страницу “Комментарии” или “Отзывы” для веб-сайта. Принимайте отзыв пользователя в виде формы, обрабатывайте данные отзыва с помощью сценария, затем прорисовывайте окно с благодарностью.
- 10.4.** *Простое приложение CGI.* Создайте книгу посетителей из Интернета. Принимайте имя, адрес электронной почты и отзыв пользователя, затем вносите эти данные в файл (в своем предпочтительном формате). Как и в упражнении 10.3, открывайте окно с благодарностью за внесение записи в книгу посетителей. Предусмотрите также ссылку, чтобы пользователи могли просматривать книги посетителей.
- 10.5.** *Cookie-файлы веб-браузера и регистрация на веб-сайте.* Создайте службу проверки подлинности пользователей для веб-сайта. Применяйте шифрование для управления именами и паролями пользователей. Возможно, вы уже подготовили чисто текстовую версию этого упражнения по книге *Core Python Programming* или *Core Python Language Fundamentals*. В таком случае при желании воспользуйтесь отдельными частями этого решения.

**Дополнительное задание.** Ознакомьтесь с принципами задания cookie-файлов веб-браузера и обеспечьте сопровождение сеанса регистрации по истечении четырех часов со времени последней успешной регистрации.

**Дополнительное задание.** Организуйте федеративную проверку подлинности с помощью OpenID, которая позволяет пользователям регистрироваться с помощью Google, Yahoo!, AOL, WordPress или даже через такую частную систему аутентификации, как Facebook Connect или вход в Twitter. Можно также использовать инструмент Google Identity Toolkit, который находится по адресу <http://code.google.com/apis/identitytoolkit>.

- 10.6.** *Ошибки.* Что происходит при аварийном завершении сценария CGI? В чем может помочь модуль `cgitb`?
- 10.7.** *CGI, обновления файлов и Zip-файлы.* Создайте приложение CGI, которое не только сохраняет файлы на диск сервера, но и применяет интеллектуальный способ распаковки Zip-файлов (или других архивов) в подкаталог с именем файла архива.
- 10.8.** *Веб-приложение для базы данных.* Рассмотрите схему базы данных, которую требуется ввести в состав веб-приложения для базы данных. Применительно к такому многопользовательскому приложению необходимо предоставлять всем доступ для чтения всего содержимого базы данных, но, возможно, лишь одному пользователю дать право на запись. Одним из примеров может быть адресная книга для своих членов семьи и родственников. Каждый член семьи после успешной регистрации получает веб-страницу с несколькими функциями, добавляет запись, просматривает свою запись, обновляет или удаляет, а также просматривает все записи (всю базу данных).

Спроектируйте класс `UserEntry`, который создает запись в базе данных для каждого экземпляра этого класса. Для реализации такой платформы регистрации можно использовать решение любой из предыдущих задач. Наконец, для базы данных можно использовать любой механизм хранения данных, например, реляционную базу данных, такую как MySQL, или один

из более простых модулей создания постоянного хранилища Python, допустим, `anydbm` или `shelve`.

- 10.9. Ядро электронной коммерции.** Создайте веб-службу электронной коммерции/онлайн-торговли, которая является универсальной и может быть расширена для поддержки большого числа клиентов. Добавьте собственную систему аутентификации, а также классы для пользователей и тележек для покупок (если у вас есть книга *Core Python Programming* или *Core Python Language Fundamentals*, то можете использовать классы, созданные в решениях к упражнениям 4 и 11 в главе “Объектно-ориентированное программирование”). Не забывайте, что потребуются также код управления предоставляемыми товарами или услугами. Может также потребоваться подключиться к платежной системе наподобие тех, которые предлагают сайты PayPal или Google. После чтения следующих глав перенесите это временное решение CGI на платформу Django, Pyramid или App Engine Google.

*Python 3.* Рассмотрите различия между `friendsC.py` и `friendsC3.py`. Опишите каждое изменение.

*Python 3.* Поддержка Юникода или ASCII, с одной стороны, и текстовых данных/байтов, с другой. Перенесите пример для Юникода, `uniCGI.py`, в версию Python 3.

## WSGI

- 10.10. Предыстория.** Что такое технология WSGI и в чем состоят некоторые причины ее создания?
- 10.11. Предыстория.** Назовите некоторые применяющиеся или применявшиеся методы преодоления проблемы масштабирования в CGI.
- 10.12. Предыстория.** Назовите некоторые известные платформы, совместимые с WSGI, и проведите исследование, которое помогло бы найти еще не получившие известность.
- 10.13. Предыстория.** В чем состоят различия между WSGI и CGI?
- 10.14. Приложения WSGI.** К какому типу объектов Python могут относиться приложения WSGI?
- 10.15. Приложения WSGI.** В чем состоят два обязательных аргумента для приложения WSGI? Опишите более подробно второй из них.
- 10.16. Приложения WSGI.** Укажите возможные возвращаемые типы для приложения WSGI.
- 10.17. Приложения WSGI.** Решения к упражнениям от 10.1 до 10.11 являются применимыми, только если сервер обрабатывает данные формы по такому же принципу, как и в технологии CGI. Выберите одно из решений для переноса в WSGI, где оно будет работать, независимо от выбранного сервера, совместимого с WSGI, не считая лишь небольших изменений.
- 10.18. Серверы WSGI.** Отличительной особенностью серверов WSGI, представленных в разделе 10.6.5, является применение типичной функции сервера `run_wsgi_app()`, которая выполняет приложение WSGI.

**а)** В функции `run_wsgi_app()` в настоящее время не предусмотрена возможность задавать необязательный третий параметр `exc_info`. Изучите документы PEP 333 и 3333 и добавьте поддержку для `exc_info`.

**б)** Перенесите эту функцию в версию Python 3.

**10.19. Практический пример.** Сравните и сопоставьте реализацию WSGI в следующих веб-платформах Python: Werkzeug, WebOb, Django, webapp для Google App Engine.

**10.20. Стандарты.** Документ PEP 3333 содержит разъяснения и усовершенствования, относящиеся к PEP 333 для Python 3, а документ PEP 444 вносит дополнения. Опишите назначение PEP 444 и его связь с существующими документами PEP.

# Веб-платформы: Django

## *В этой главе...*

- Введение
- Веб-платформы
- Введение в Django
- Проекты и приложения
- Первое приложение Hello World (блог)
- Создание модели для добавления службы базы данных
- Командный интерпретатор для приложений Python
- Приложение администрирования Django
- Создание пользовательского интерфейса для блога
- Усовершенствование вывода
- Работа с данными, введенными пользователем
- Простые и модельные формы
- Дополнительные сведения о представлениях
- Усовершенствования внешнего интерфейса
- Проверка компонентов
- Промежуточное приложение Django: TweetApprover
- Ресурсы

*Python — единственный язык, в котором количество поддерживаемых веб-платформ больше, чем количество ключевых слов.*

Харальд Армин Масса  
(Harald Armin Massa), декабрь 2005 года

## 11.1. Введение

В этой главе мы выйдем за рамки стандартной библиотеки Python и рассмотрим одну веб-платформу, широко применяемую в языке Python, — Django. Вначале будет проведен общий обзор веб-платформ, а затем речь пойдет о том, как разрабатывать приложения с помощью веб-платформы Django. Это описание начнется с основ и самого простого приложения, Hello World, после чего мы рассмотрим другие вопросы, с которыми чаще всего приходится сталкиваться при разработке реального приложения. Структуру главы можно вкратце определить с помощью следующего плана: фундаментальное введение, за которым следует описание вспомогательного приложения, предназначенного для работы с социальной сетью Twitter, электронной почтой и протоколом OAuth (открытый протокол авторизации, предназначенный для получения доступа к данным с помощью программных интерфейсов приложений, API).

Задача этой главы состоит в том, чтобы представить читателю реальный инструмент, повседневно используемый разработчиками Python для выполнения своей работы. Эта глава позволяет освоить новые навыки и приобрести достаточно широкие знания, чтобы с их помощью создавать более сложные приложения на основе Django. Этими навыками можно будет также воспользоваться для перехода на любую другую веб-платформу Python. Для начала определим общую тему обсуждения.

## 11.2. Веб-платформы

Существенное понимание особенностей разработки для веб предоставляет материал, изложенный в главе 10. Эта работа является весьма трудоемкой, но вместо того, чтобы делать все вручную, можно воспользоваться результатами значительных усилий, затраченными другими разработчиками, и упростить себе жизнь. Среду разработки для веб принято называть *веб-платформой*. Веб-платформы предназначены для оказания помощи программисту при выполнении наиболее распространенных задач и предоставления ресурсов, позволяющих создавать, обновлять, эксплуатировать и масштабировать приложения при минимальном объеме дополнительной работы.

Кроме того, как было описано в предыдущих главах, возможность использования CGI теперь исключена в связи с ограничениями масштабируемости, характерными для этого интерфейса. Поэтому представители сообщества Python обратились к использованию более мощных решений для веб-серверов, таких как Apache, `lighttpd` (читается как `lighty` — лайти) или `nginx`. Для некоторых серверов, таких как `Pylons` и `CherryPy`, предусмотрена собственная “экосистема” в виде веб-платформ. Однако предоставление содержимого по запросу — это лишь один аспект создания веб-приложений. По-прежнему приходится предусматривать применение таких вспомогательных инструментов, как платформа JavaScript, объектно-реляционные преобразователи (`object-relational mapper` — ORM), адаптеры баз данных низкого уровня, системы формирования веб-шаблонов, а также инструментов, которые непосредственно не относятся к разработке, но необходимы в любой работе по программированию: средства



тестирования модулей и (или) платформа непрерывной интеграции. Веб-платформы Python представляют собой либо отдельные (или комплексные) субкомпоненты, либо полнофункциональные системы.

Под термином *полнофункциональная* подразумевается, что система с таким определением позволяет разрабатывать код для всех этапов и уровней развития веб-приложения. Платформы, рассматриваемые как полнофункциональные, поддерживают все необходимые службы, такие как веб-сервер, объектно-реляционный преобразователь для базы данных, средство формирования шаблонов, а также все необходимые функциональные средства для промежуточного программного обеспечения. В составе некоторых веб-платформ предусмотрена даже библиотека JavaScript. Несомненно, Django относится к числу наиболее широко известных сегодня на рынке веб-платформ; многие рассматривают ее как ответ Python на создание Ruby on Rails. Эта платформа включает все указанные выше службы в виде отдельного, целостного решения (за исключением встроенной библиотеки JavaScript; таковая специально не предусмотрена, поскольку предоставляется возможность использовать любую библиотеку по своему выбору). Как будет показано в главе 12, платформа Google App Engine также предоставляет многие из указанных компонентов, но в большей степени направлена на обеспечение масштабируемости и создание быстродействующих веб-приложений (и отличных от веб) обработки запросов и ответов, которые размещены на площадках интернет-гиганта Google.

Безусловно, веб-платформа Django была создана как единое целое одним коллективом специалистов, но не всегда разработка веб-платформы складывается подобным образом. Например, TurboGears, лучшая в своем роде полнофункциональная система, создана разрозненным коллективом разработчиков и служит для сопряжения в одном комплексе таких широко известных отдельных компонентов, как ToscaWidgets (высокоуровневые графические элементы для веб, в которых может использоваться целый ряд платформ JavaScript, таких как ExtJS, jQuery и т.д.), SQLAlchemy (ORM), Pylons (веб-сервер) и Genshi (поддержка шаблонов). Платформы, созданные на основе такого подхода, обеспечивают большую гибкость, поскольку пользователи могут выбирать среди различных систем поддержки шаблонов, библиотек JS, инструментов для формирования простого кода SQL и нескольких веб-серверов. Приходится лишь сделать небольшую уступку, отказавшись от единообразия и полной уверенности в успехе, которые достигаются при использовании исключительно одного инструмента. Тем не менее для многих разработчиков такая ситуация давно стала привычной.

Кроме того, весьма популярна платформа Pyramd, для которой предшественниками были *geroze.bfg* (или просто BFG) и веб-платформа Pylons. Применяемый в Pyramd подход еще более прост: она поддерживает лишь основные функции, такие как отправка URL, поддержка шаблонов, безопасность и ресурсы. Если потребуется что-либо еще, то эти функциональные возможности приходится реализовывать самостоятельно. Платформа Pyramd упрощена до предела, для нее предусмотрены качественные средства тестирования и превосходная документация, а в состав пользователей вошли все те, кто в свое время входил в сообщество Pylons и BFG, поэтому данная платформа является одной из самых предпочтительных среди многочисленных веб-платформ, предусмотренных для Python.

Многие программисты, которые впервые начинают осваивать Python, до этого работали в таких системах, как Rails или даже PHP (развитие платформы PHP начиналось с того, что вначале был создан язык поддержки сценариев, внедряемых в код HTML, и привело к созданию современной крупной целостной системы). Переходя на Python, разработчик получает то преимущество, что он больше не должен

ограничиваться использованием лишь одного языка и одной платформы. Python предоставляет для выбора много платформ, и это подчеркивает цитата, которая стала эпиграфом для данной главы. Привлекательность веб-платформ стала возрастать со времени создания стандарта интерфейса шлюза для веб-сервера (Web Server Gateway Interface — WSGI), который определен в документе PEP 333 по адресу <http://python.org/dev/peps/pep-0333>.

В отношении WSGI можно кратко отметить, что этот стандарт фактически является не определением кода или API, а скорее определением интерфейса, который освобождает разработчика веб-платформы от необходимости создавать для платформы специальный веб-сервер, а это, в свою очередь, позволяет разработчикам приложений избавиться от необходимости применять лишь жестко заданный сервер, тогда как они бы предпочли использовать какой-то другой. Интерфейс WSGI позволяет разработчикам приложений легко переходить с одного сервера, совместимого с WSGI, на другой (или разработать новый сервер), не задумываясь над тем, что в связи с этим может потребоваться изменить код приложения. Дополнительные сведения об интерфейсе WSGI см. в главе 10.

Я не знаю, вправе ли я говорить об этом (и тем более писать), но если высококвалифицированные разработчики на языке Python остаются недовольны тем выбором веб-платформ, который имеется в их распоряжении, то разрабатывают новую. В конечном итоге веб-платформ, поддерживаемых языком Python, больше, чем поддерживаемых ключевых слов, не правда ли? К другим платформам, с которыми читатель так или иначе познакомится уже уже познакомился, относятся web2py, web.py, Tornado, Diesel и Zope. В качестве хорошего источника информации по этой теме можно рекомендовать страницу вики-сайта на веб-сайте Python по адресу <http://wiki.python.org/moin/WebFrameworks>.

Итак, после краткого введения перейдем к описанию основных принципов разработки для веб и приступим к рассмотрению платформы Django.

## 11.3. Введение в Django

В описании платформы Django сказано, что это веб-платформа для взыскательных разработчиков, перед которыми поставлены жесткие сроки. Об этой платформе впервые стало известно в начале 2000-х годов. Она была создана веб-разработчиками для интернет-версии газеты Lawrence Journal-World, но мировому сообществу представлена в 2005 году как платформа программной поддержки журналистики, представители которой работают в жестких временных рамках. Мы сами поставим перед собой жесткие сроки и посмотрим, насколько быстро нам удастся создать с помощью Django очень простой блог. В дальнейшем ту же задачу мы попытаемся решить с помощью Google App Engine. (Роль взыскательного разработчика будет играть сам читатель.) Этот пример будет пройден чрезвычайно быстро, но все равно останется достаточно времени на объяснения, чтобы читатель мог проследить за тем, что происходит. Читатели, которые хотели бы ознакомиться с подробным описанием именно этого примера, могут найти его в главе 2 книги *Python Web Development with Django* (Addison-Wesley, 2009), написанной уважаемыми коллегами, Джефом Форсье (Jeff Forcier), ведущим разработчиком Fabric, Полом Биссексом (Paul Bissex), создателем dpaste, и вашим покорным слугой.



### Работа по подготовке к использованию в версии Python 3

**3.x**

Ко времени написания данной книги веб-платформа Django не поддерживала версию Python 3, поэтому все примеры в этой главе относятся только к версии Python 2.x. Вариант платформы для Python 3 к этому времени (к моменту выхода этой книги) прошел все проверки, поэтому будет выпущен сразу после завершения подготовки документации. После того как это произойдет, я размещу версии кода этой главы для Python 3 на веб-сайте книги. У меня есть все основания рассчитывать на то, что поддержка в версии Python 3 станет еще одним важным шагом в развитии таких крупных платформ, как Django, а также других библиотек поддержки инфраструктуры, таких как адаптеры баз данных, которые будут реализованы на этой платформе нового поколения.

## 11.3.1. Установка

До начала разработки на платформе Django требуется установить необходимые компоненты, к которым, кроме самого программного обеспечения Django, относятся дополнительные программные средства.

### Предварительные требования

До начала установки Django должно быть установлено программное обеспечение Python. Однако, поскольку это не первая глава данной книги по языку Python, можно предположить, что это уже сделано. Кроме того, Python уже установлен в большинстве операционных систем, совместимых с POSIX (Mac OS X, Linux, \*BSD). Как правило, загружать и устанавливать Python приходится только пользователям Microsoft Windows.

Для работы с платформой Django требуется также веб-сервер, а поскольку ведущим из них является Apache, он и используется в большинстве вариантов развертывания. Разработчики Django рекомендуют устанавливать модуль Apache `mod_wsgi` и предоставляют по адресу <http://docs.djangoproject.com/en/dev/topics/install/#install-apache-and-mod-wsgi> основные инструкции, а по адресу <http://docs.djangoproject.com/en/dev/howto/deployment/modwsgi/> — всеобъемлющую документацию. Еще один превосходный комплект документов для более сложных установок (такowymi являются варианты среды, в которых несколько веб-сайтов, или проектов, Django размещаются лишь на одном экземпляре Apache) можно найти по адресу <http://forum.webfaction.com/viewtopic.php?id=3646>. Что касается модуля `mod_python`, то он главным образом входит в состав более старых вариантов установки или дистрибутивов операционной системы, которые распространялись до внедрения в качестве стандартного модуля `mod_wsgi`. Поддержка модуля `mod_python` в настоящее время официально отменена (и этот модуль полностью удален в версии Django 1.5).

В завершение обсуждения вопроса о веб-серверах<sup>1</sup> следует отметить, что для серверов производственного назначения не обязательно использовать исключительно

<sup>1</sup> Потребность в использовании веб-сервера возникает только с началом развертывания приложения, поэтому программисты в ходе самой разработки могут обойтись без него. В состав Django входит сервер для разработки (также рассматриваемый в этой главе), который применяется во время создания и проверки приложения в тот период, пока ведется подготовка к передаче программного продукта в эксплуатацию.

Apache. Как уже было сказано, есть другие варианты веб-серверов, причем многие из них характеризуются меньшей потребностью в памяти и работают быстрее; возможно, именно один из этих серверов окажется наиболее подходящим для вашего конкретного приложения. Дополнительные сведения о некоторых возможных вариантах применения веб-серверов можно найти по адресу <http://code.djangoproject.com/wiki/ServerArrangements>.

Для успешной работы Django обязательно должна применяться база данных. Стандартная версия Django в настоящее время может эксплуатироваться только в сочетании с реляционными СУБД с поддержкой языка SQL. Пользователи Django применяют в основном такие четыре базы данных, как PostgreSQL, MySQL, Oracle и SQLite. Намного проще в установке по сравнению с другими является SQLite. Более того, SQLite — единственная из четырех баз данных, для которой не требуется работающий сервер базы данных, поэтому она к тому же является самой удобной. Безусловно, в связи с отсутствием сервера не следует считать эту базу данных игрушечной; она демонстрирует превосходные характеристики при сравнении со своими более мощными аналогами.

## 2.5

Причины, способствующие упрощению настройки SQLite, таковы. Адаптер базы данных SQLite входит в комплект всех версий Python, начиная с версии 2.5. Следует учитывать, что адаптер является основным средством взаимодействия с базой данных. В некоторых дистрибутивах SQLite уже включена в комплект, в других применяется связь с базой данных SQLite, установленной в системе, а для всех остальных приходится загружать и устанавливать эту базу данных вручную.

Система SQLite — не единственная реляционная СУБД, поддерживаемая платформой Django, поэтому не следует считать себя обязанным работать только с ней, особенно если в компании уже применяется база данных на основе сервера. Дополнительные сведения о платформе Django и установке базы данных см. по адресу <http://docs.djangoproject.com/en/dev/topics/install/#database-installation>.

Кроме того, в последнее время происходит бурный расцвет нереляционных баз данных (которые условно обозначаются как NoSQL). По-видимому, это обусловлено дополнительными возможностями масштабирования, которые обеспечивают такими системами, в условиях постоянно растущей потребности в хранении все большего объема данных. Например, если для поддержки такого объема данных, который накапливается в процессе работы Facebook, Twitter или аналогичных служб, применяется реляционная база данных, то обычно требуется постоянно проводить ручную операции секционирования, которые именуются также сегментированием. Разработчики, желающие создавать приложения на основе использования в качестве собственных хранилищ данных таких баз данных NoSQL, как MongoDB или Google App Engine, могут попытаться воспользоваться версией Django-nonrel, которая позволяет пользователям работать и с реляционными, и с нереляционными базами данных, а не только с одним типом баз данных. (Отметим, что для Google App Engine предусмотрен также аналог с реляционной базой данных, совместимой с MySQL, — Google Cloud SQL.)

Вариант Django-nonrel можно загрузить по адресу <http://www.allbuttonspressed.com/projects/django-nonrel>, после чего получить один из адаптеров <https://github.com/FlaPer87/django-mongodb-engine> (Django с MongoDB) или

<http://www.allbuttonspressed.com/projects/djangoappengine> (Django на основе хранилища данных Google App Engine). Ко времени написания данной книги версия Django-nonrel представляла собой просто один из аналогов Django, поэтому для работы с книгой можно загрузить и установить эту версию вместо стандартного пакета Django. Кроме того, применение Django-nonrel для разработки оправдано, если в дальнейшем предусматривается внедрение приложения на основе этой версии в эксплуатацию. Как указано на веб-сайте <http://www.allbuttonspressed.com/projects/django-nonrel>, в эту версию внесен минимальный объем изменений по сравнению с Django (возможно, меньше 100 строк). Версия Django-nonrel предоставляется в виде архивированного файла, поэтому достаточно просто ее разархивировать, перейти в развернутую папку и выполнить следующую команду:

```
$ sudo python setup.py install
```

Аналогичные инструкции распространяются на использование загруженного архивного файла стандартной версии Django (см. ниже), поэтому при желании можно полностью пропустить следующий подраздел (“Установка Django”) и перейти к началу учебника.

## Установка Django

Предусмотрено несколько способов установки Django в системе, которые перечислены ниже в порядке возрастания трудозатрат и (или) сложности.

- Администратор пакетов Python.
- Администратор пакетов операционной системы.
- Архив со стандартным выпуском.
- Репозиторий исходного кода.

В простейшем процессе загрузки и установки используются преимущества таких инструментов управления пакетами Python, как `easy_install` or `Setuptools` ([http://packages.python.org/distribute/easy\\_install.html](http://packages.python.org/distribute/easy_install.html)) или `pip` (<http://pip.openplans.org>); оба инструмента доступны для всех платформ. Что касается работы пользователей Windows с `Setuptools`, то они должны установить файл `easy_install.exe` в папке `Scripts`, в которой находится дистрибутив Python. Достаточно лишь вызвать на выполнение одну команду, но это следует выполнять в окне команд DOS:

```
C:\WINDOWS\system32>easy_install django
Searching for django
Reading http://pypi.python.org/simple/django/
Reading http://www.djangoproject.com/
Best match: Django 1.2.7
Downloading http://media.djangoproject.com/releases/1.2/Django-1.2.7.tar.gz
Processing Django-1.2.7.tar.gz
...
Adding django 1.2.7 to easy-install.pth file
Installing django-admin.py script to c:\python27\Scripts

Installed c:\python27\lib\site-packages\django-1.2.7-py2.7.egg
Processing dependencies for django
Finished processing dependencies for django
```

Чтобы не приходилось вводить полный путь к программе `easy_install.exe`, рекомендуется добавить в переменную среды `PATH` путь `C:\Python2x\Scripts`<sup>2</sup>, с учетом установленной версии Python 2.x. В системе POSIX программа `easy_install` устанавливается в таком широко применяемом пути, как `/usr/bin` или `/usr/local/bin`, поэтому можно специально не предусматривать добавление нового каталога к переменной `PATH`. Тем не менее может потребоваться использование команды `sudo`, чтобы обеспечить установку в типичном системном каталоге, например `/usr/local`. Применяемая команда может выглядеть примерно так:

```
$ sudo easy_install django
```

или так:

```
$ sudo pip install django
```

Необходимость в использовании `sudo` возникает только в том случае, если установка выполняется в таком местоположении, для которого требуется доступ суперпользователя; в противном случае проведите установку в одном из пользовательских каталогов. Рекомендуется также использовать какую-то “контейнерную” среду, такую как `virtualenv`. Применение среды `virtualenv` дает возможность осуществить несколько установок с различными версиями Python и (или) Django, установить различные базы данных и т.д. Каждая среда работает в своем собственном контейнере, и ее создание, управление, эксплуатацию и удаление можно проводить, не затрагивая другие установки. Дополнительные сведения о среде `virtualenv` см. по адресу <http://pypi.python.org/pypi/virtualenv>.

Еще один способ установки Django состоит в использовании администратора пакетов операционной системы, если таковой имеется в конкретной системе. Вообще говоря, администраторы пакетов предусмотрены только в компьютерах POSIX (Linux и Mac OS X). Должна быть выполнена примерно такая команда:

```
(Linux) $ sudo COMMAND install django
(Mac OS X) $ sudo port install django
```

Что касается Linux, то в качестве параметра `COMMAND` необходимо указать администратор пакетов применяемого дистрибутива, например, `apt-get`, `yum`, `aptitude` и др. Инструкции по установке из дистрибутивов можно найти по адресу <http://docs.djangoproject.com/en/dev/misc/distributions>.

Кроме только что описанных методов, можно просто загрузить и установить исходный архив выпуска с веб-сайта Django. После его разархивирования можно выполнить обычную команду установки:

```
$ sudo python setup.py install
```

Более конкретные инструкции можно найти по адресу <http://docs.djangoproject.com/en/dev/topics/install/#installing-an-official-release>.

Наиболее опытные разработчики предпочитают получать новейшие версии непосредственно из дерева исходного кода Subversion. Соответствующие инструкции можно найти по адресу <http://docs.djangoproject.com/en/dev/topics/install/#installing-the-development-version>.

<sup>2</sup>

Пользователь персонального компьютера с операционной системой Windows может внести изменения в переменную `PATH`, щелкнув правой кнопкой мыши на пиктограмме “My Computer” (Мой компьютер) и выбрав “Properties” (Свойства).

Наконец, ниже приведены полные инструкции по установке.

```
http://docs.djangoproject.com/en/dev/topics/install/
#install-the-django-code
```

Следующий шаг состоит в вызове сервера и проверке того, что все установлено должным образом и работает правильно. Вначале рассмотрим некоторые основные понятия Django: проекты и приложения.

## 11.4. Проекты и приложения

Что такое проекты и приложения Django? Проект Django можно проще всего определить, как совокупность всех файлов, необходимых для создания и эксплуатации веб-сайта как такового. В папке проекта создается набор из одного или нескольких подкаталогов, имеющих специальное назначение; эти подкаталоги именуются приложениями, но это не означает, что в каждой папке проекта находится приложение. Приложения могут относиться исключительно к одному проекту или представлять собой многократно используемые компоненты, которые можно переносить из одного проекта в другой. Приложения — это отдельные субкомпоненты функциональных средств, совокупность которых образует полную среду веб-сайта. Отдельные приложения могут запрашивать и обрабатывать обратную связь от пользователя/читателя, обновлять информацию в реальном времени, управлять данными каналов, агрегировать данные с других сайтов и т.д.

Одни из наиболее широко известных повторно используемых приложений Django можно найти в составе платформы Pinax. Эти приложения включают (но не ограничиваются) средства аутентификации (поддержка OpenID, управление паролями и т.д.), передачи сообщений (проверка электронной почты, уведомления, взаимодействие пользователей, организация групп по интересам, обсуждения в потоках и т.д.), а также в большей степени автономные средства, такие как управление проектами, ведение блогов, разметка содержимого и импорт данных о контактах. Дополнительные сведения о платформе Pinax можно найти по адресу <http://pinaxproject.com>.

Такая концепция проектов и приложений позволяет организовывать поддержку функциональных средств на основе простого включения, а также предоставляет дополнительный бонус, стимулируя применение адаптивного проектирования и повторное использование кода. После ознакомления с этими основными понятиями проектов и приложений приступим к созданию конкретного проекта.

### 11.4.1. Создание проекта в Django

В состав программного обеспечения Django входит утилита `django-admin.py`, которая позволяет проще решать такие задачи, как создание вышеупомянутых каталогов проекта. На платформах POSIX эти каталоги обычно устанавливаются в таких каталогах, как `/usr/local/bin`, `/usr/bin` и т.д., а на компьютерах с операционной системой Windows каталоги размещаются в папке Scripts, которая находится непосредственно в папке для установок Python, например `C:\Python27\Scripts`. Независимо от того, применяется ли компьютер с операционной системой POSIX или Windows, необходимо обеспечить, чтобы сценарий `django-admin.py` находился в пути, доступном с помощью переменной среды `PATH`, для вызова этого сценария из командной строки (еще один вариант состоит в том, что сценарий можно вызвать, указав полный путь к нему).

Что касается компьютеров с операционной системой Windows, то скорее всего потребуется вручную добавить обозначения каталогов `c:\python27` и `c:\python27\scripts` в системную переменную `PATH` для обеспечения бесперебойной работы программного обеспечения (независимо от того, в каком каталоге установлен Python). Для этого необходимо открыть окно Панель управления (Control Panel), а затем щелкнуть на пиктограмме Система (System) или щелкнуть правой кнопкой мыши на пиктограмме Мой компьютер (My Computer) и выбрать пункт Свойства (Properties). В открывшемся окне откройте вкладку Дополнительно (Advanced) и щелкните на кнопке Переменные среды (Environment Variables). Можно выбрать вариант с редактированием записи `PATH` для одного пользователя (верхнее окно со списком) или для всех пользователей (нижнее окно со списком), а затем добавить `;c:\python27;c:\python27\scripts` в конце строки в текстовом поле со значением переменной. На рис. 11.1 показана часть открывающегося при этом экрана.

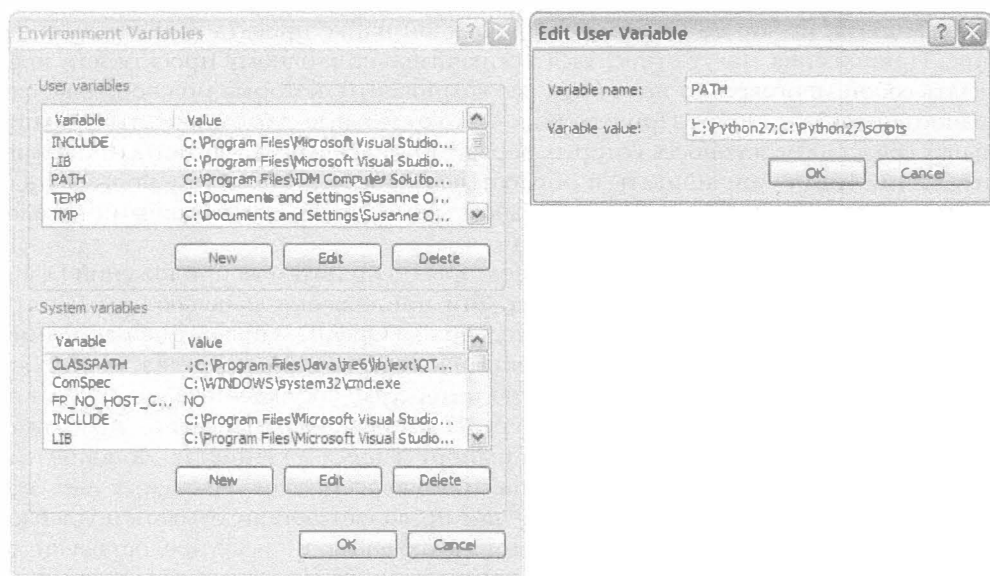


Рис. 11.1. Добавление каталогов Python к переменной `PATH` операционной системы Windows

После задания необходимых данных в переменной `PATH` (на той или иной платформе) должна появиться возможность выполнить команду `python`, чтобы открылся интерактивный интерпретатор, и вызвать команду `Django`, `django-admin.py`, для ознакомления с ее использованием. Чтобы выполнить эту проверку, необходимо открыть командный интерпретатор Unix или окно команд DOS и ввести указанные команды. Убедившись в том, что предыдущее задание выполнено успешно, переходите к дальнейшему изучению материала.

Следующий шаг состоит в переходе в каталог или папку, где намечено размещение собственного кода. Чтобы создать проект в текущем рабочем каталоге, выполните следующую команду (мы используем ничего не значащее имя проекта, такое как `mysite`, но читатель при желании может задать для обозначения проекта любое имя):

```
$ django-admin.py startproject mysite
```



Необходимо отметить, что, работая на персональном компьютере Windows, необходимо в первую очередь открыть окно команд DOS. Разумеется, появится такое приглашение к вводу данных, которое скорее будет напоминать `C:\WINDOWS\system32>`, как в обычном командном интерпретаторе, в отличие от знака доллара (\$) или знака процента (%) в системе POSIX.

Перейдем к рассмотрению содержимого каталога, чтобы ознакомиться с тем, что было создано с помощью указанной команды. На компьютере POSIX содержимое каталога будет выглядеть примерно так:

```
$ cd mysite
$ ls -l
total 32
-rw-r--r-- 1 wesley admin 0 Dec 7 17:13 __init__.py
-rw-r--r-- 1 wesley admin 546 Dec 7 17:13 manage.py
-rw-r--r-- 1 wesley admin 4778 Dec 7 17:13 settings.py
-rw-r--r-- 1 wesley admin 482 Dec 7 17:13 urls.py
```

Если разработка ведется в ОС Windows, то, как показано на рис. 11.2, в окне проводника отображается содержимое, аналогичное тому, которое формируется при предварительном создании, допустим, папки `C:\py\django` с намерением поместить в нее проект.

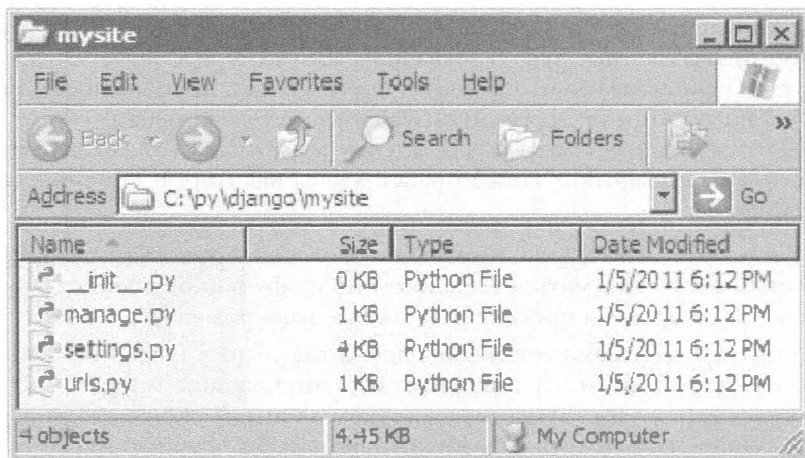


Рис. 11.2. Папка `mysite` на персональном компьютере с операционной системой Windows

На платформе Django в основе проекта лежат четыре файла, `__init__.py`, `manage.py`, `settings.py` и `urls.py` (добавление приложений происходит позже). В табл. 11.1 указано назначение каждого файла.

Таблица 11.1. Файлы проекта Django

Имя файла	Описание/назначение
<code>__init__.py</code>	Служит для Python указанием, что это пакет
<code>urls.py</code>	Глобальная конфигурация URL (URLconf)
<code>settings.py</code>	Конфигурация, зависящая от проекта
<code>manage.py</code>	Интерфейс командной строки для приложений

Заслуживает внимания то, что каждый файл, созданный командой `startproject`, представляет собой файл исходного кода Python, т.е. отсутствуют какие-либо файлы `.ini`, данные XML или даже файлы с традиционным синтаксисом описания конфигурации. Особенностью платформы Django является то, что вся ее работа по возможности организована с применением кода Python. Это не приводит к усложнению платформы, но обеспечивает большую гибкость, а также предоставляет возможность импортировать в файл настроек параметры из других файлов с учетом текущей конфигурации или вычислять необходимые значения, вместо того, чтобы их жестко задавать. Чем шире применяется код Python, тем меньше становится барьеров. Вполне очевидно, что `django-admin.py` также представляет собой сценарий на языке Python. Он служит в качестве интерфейса командной строки, с помощью которого пользователь может управлять своим проектом. Аналогичным образом сценарий `manage.py` служит для управления приложениями. (Для обоих сценариев предусмотрена опция справки, с помощью которой можно получить подробную информацию об их использовании.)

### 11.4.2. Работа с сервером для разработки

На данном этапе мы еще не приступаем к изучению того, как создавать приложение, но должны заранее ознакомиться с некоторыми удобными средствами Django, которые позволяют проще решать эту задачу. Одним из самых удобных средств является встроенный веб-сервер Django. Это сервер, предназначенный для этапа разработки, который эксплуатируется на локальном компьютере. Следует отметить, что не рекомендуется использовать этот сервер для развертывания общедоступных сайтов, поскольку он не является сервером производственного назначения.

В связи с этим возникает вопрос: для чего же тогда нужен отдельный сервер, предназначенный для разработки? Ниже представлены некоторые причины такой организации работы.

1. Этот сервер можно использовать для проверки проекта (и приложения) без необходимости заниматься созданием полнофункциональной рабочей среды лишь для того, чтобы провести некоторые эксперименты с кодом.
2. Данный сервер автоматически обнаруживает, что в исходные файлы Python внесены изменения, и перезагружает соответствующие модули. Это позволяет экономить время и с большим удобством заниматься разработкой по сравнению с системами, которые требуют перезапуска вручную после каждого исправления кода.
3. В сервере для разработки реализованы функции, позволяющие находить и отображать статические медиафайлы для приложения Django Administration (или кратко `admin`), благодаря чему можно сразу же начать работу с этими файлами. (Сведения о приложении `admin` приведены ниже. На данный момент следует лишь отметить, что приложение `admin` не следует пугать со сценарием `django-admin.py`.)

Для ввода в действие сервера разработки достаточно просто выполнить следующую команду, работая с утилитой `manage.py` в рамках проекта:

```
(UNIX) $ python ./manage.py runserver
(PCs) C:\py\django\mysite> python manage.py runserver
```

Если используется система POSIX и права на выполнение сценария назначены таким образом: `$ chmod 755 manage.py`, то можно не вызывать интерпретатор Python явно, например `$ ./manage.py runserver`. То же касается и окна команд DOS, если Python правильно установлен в реестре Windows.

После запуска сервера появляется примерно такой вывод, как в следующем примере (в Windows используется другая комбинация клавиш для выхода из программы):

```
Validating models...
0 errors found.
```

```
Django version 1.2, using settings 'mysite.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Откройте ссылку (<http://127.0.0.1:8000/> или <http://localhost:8000/>) в браузере, и развернется окно платформы Django с надписью “It Worked!”, как показано на рис. 11.3.

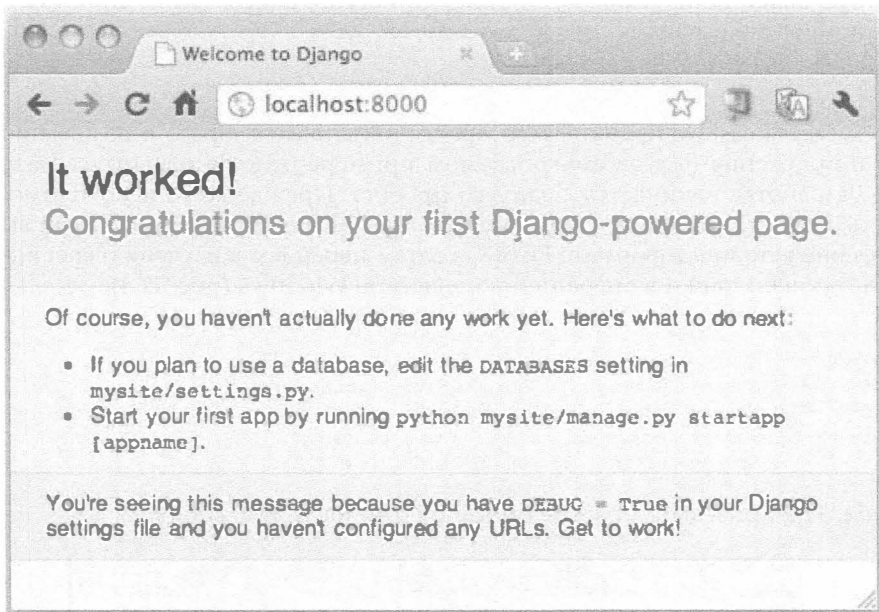


Рис. 11.3. Вступительное окно “It worked!” платформы Django

Следует учитывать, что для эксплуатации сервера может применяться другой порт, но он должен быть указан в командной строке. Например, если должен применяться порт 8080, то вместо этого необходимо указать следующую команду: `$ python ./manage.py runserver 8080`. Со всеми параметрами ключевого слова `runserver` можно ознакомиться по адресу <http://docs.djangoproject.com/en/dev/ref/django-admin/#django-admin-runserver>.

Если окно “It worked!”, которое показано на рис. 11.3, успешно открывается, то установка сервера выполнена должным образом. При этом в ходе терминального сеанса осуществляется регистрация на сервере разработки первого запроса GET:

```
[11/Dec/2010 14:15:51] "GET / HTTP/1.1" 200 2051
```

Эта строка, регистрируемая в журнале, состоит из четырех частей. Крайней слева является отметка времени, затем приведены сам запрос и код ответа HTTP, наконец, указано количество байтов (в каждом конкретном случае вывод может быть немного другим). Страница “It Worked!” на платформе Django предусмотрена с той целью, чтобы можно было легко определить, работает ли сервер разработки, а знать об этом очень важно в процессе создания приложений. Если к этому моменту обнаружено, что сервер не работает, то следует еще раз проверить все описанные выше этапы подготовки. Нельзя останавливаться на полпути! Иногда бывает проще удалить весь проект и начать все с нуля, чем пытаться его отладить на данном этапе.

Если сервер функционирует успешно, можно приступить к созданию своего первого приложения Django.

## 11.5. Первое приложение Hello World (блог)

Итак, прежде всего создается проект, а затем появляется возможность разрабатывать приложения на его основе. Чтобы приступить к созданию приложения для блога, снова воспользуйтесь сценарием `manage.py`:

```
$./manage.py startapp blog
```

Что касается самого проекта, то разработчик вправе выбрать и назначение, и название приложения (а в рассматриваемом примере будет проводиться разработка блога). Разработка начинается с запуска проекта. Прежде всего необходимо предусмотреть каталог для блога в каталоге проекта. Вначале рассмотрим, как выглядит определение каталога в формате POSIX, а затем перейдем к изучению внешнего вида соответствующей папки в операционной системе Windows (рис. 11.4):

```
$ ls -l blog
total 24
-rw-r--r-- 1 wesley admin 0 Dec 8 18:08 __init__.py
-rw-r--r-- 1 wesley admin 175 Dec 10 18:30 models.py
-rw-r--r-- 1 wesley admin 514 Dec 8 18:08 tests.py
-rw-r--r-- 1 wesley admin 26 Dec 8 18:08 views.py
```

В табл. 11.2 приведены краткие описания файлов приложения.

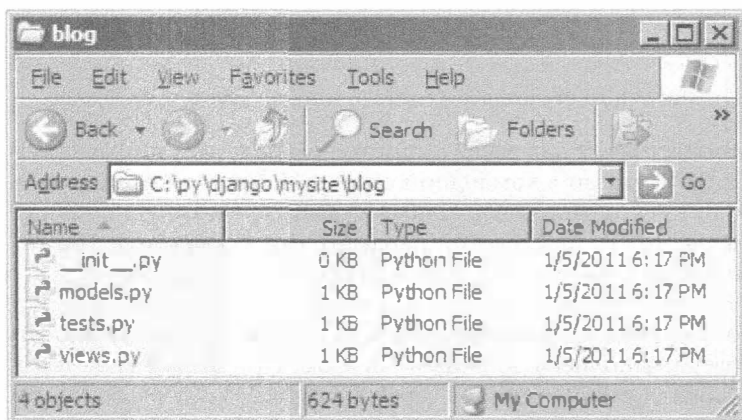


Рис. 11.4. Папка блога в персональном компьютере с операционной системой Windows

**Таблица 11.2.** Файлы приложения Django

Имя файла	Описание/назначение
<code>__init__.py</code>	Служит для Python указанием, что это пакет
<code>urls.py</code>	Конфигурация URL приложения (URLconf); этот файл не создается автоматически, как и файл URLconf для проекта (именно поэтому он не рассматривался выше)
<code>models.py</code>	Модели данных
<code>views.py</code>	Функции внешнего представления (которые можно рассматривать как контроллеры)
<code>tests.py</code>	Модульные тесты

Как и проекты, приложения представляют собой пакеты Python, но в данном случае в файлах `models.py` и `views.py` не представлен (временно) реальный код; эти сценарии определены как местозаполнители для размещения необходимого материала в будущем. Что касается модульных тестов, которые должны находиться в сценарии `tests.py`, то они также еще не подготовлены и будут реализованы в виде кода немного позже. Аналогичным образом, хотя и подготовлена конфигурация URLconf для проекта, чтобы можно было правильно перенаправлять весь трафик, соответствующая конфигурация для локального приложения не создается автоматически. Разработчик должен выполнить эту задачу сам, а затем воспользоваться директивой `include()` из файла URLconf проекта, чтобы запросы перенаправлялись в файл URLconf приложения.

Для передачи платформе Django сведений о том, что данное новое приложение должно войти в состав проекта, необходимо отредактировать файл `settings.py` (который можно также рассматривать как файл параметров). Откройте этот файл в своем предпочтительном редакторе и в его нижней части найдите кортеж `INSTALLED_APPS`. Добавьте имя приложения (блога) в виде элемента этого кортежа (обычно ближе к концу файла), чтобы файл выглядел следующим образом:

```
INSTALLED_APPS = (
 'blog',
)
```

Хотя в этом нет необходимости, можно добавить заключительную запятую, чтобы в случае введения дополнительных данных в этот кортеж для них уже было отведено место. В платформе Django кортеж `INSTALLED_APPS` используется для определения конфигурации различных частей системы, включая приложение для автоматического администрирования и платформу тестирования.

## 11.6. Создание модели для добавления службы базы данных

С этого момента можно приступить к созданию основной части приложения блога на основе Django: файла `models.py`. Именно в этом файле должны быть определены

структуры данных блога. В основе разработки на платформе Django лежит принцип DRY (Don't Repeat Yourself — “Не повторяйся”), поэтому со стороны разработчика достаточно задать информацию о модели, предусмотренной для приложения, после чего платформа Django проводит практически всю необходимую работу. Создадим основную модель, а затем рассмотрим, что может сделать Django на основе этой информации.

Модель данных определяет все типы данных, хранимых в каждой записи базы данных. Платформа Django предоставляет целый ряд полей, с помощью которых можно согласовать исходные данные с переменными в приложении. В рассматриваемом приложении предусмотрено использование полей трех разных типов (см. следующие примеры кода).

Откройте сценарий `models.py` в своем предпочтительном редакторе и добавьте следующий класс модели непосредственно после инструкции импорта, которая уже имеется в файле:

```
models.py
from django.db import models

class BlogPost(models.Model):
 title = models.CharField(max_length=150)
 body = models.TextField()
 timestamp = models.DateTimeField()
```

Это полная модель, представляющая объект почты блога с тремя полями. (Вернее, количество полей равно четырем, поскольку Django по умолчанию создает для каждой модели поле с уникальным, автоматически наращиваемым идентификатором.) Вновь созданный класс `BlogPost` можно рассматривать как подкласс класса `django.db.models.Model`. Это стандартный базовый класс Django для моделей данных, который является основой мощного объектно-реляционного преобразователя Django. Поля определены как регулярные атрибуты класса, причем каждое из них является экземпляром определенного поля класса, таким, что весь экземпляр в целом эквивалентен отдельной записи базы данных.

Для данного приложения выбран класс `CharField`, представляющий объект `title` почты блога, что позволяет задать ограничение для максимальной длины поля. С помощью класса `CharField` удобно представлять короткие, одинарные строки текста. Для более крупных фрагментов, таких как сам текст почты блога, `body`, выбран тип `TextField`. Наконец, для отметки времени `timestamp` применяется тип `DateTimeField`. Поле `DateTimeField` представлено объектом `datetime.datetime` языка Python.

Эти классы, определяющие поля, заданы также в файле `django.db.models`; кроме того, предусмотрена возможность использовать намного больше типов, от `BooleanField` до `XMLField`, а не только те три, которые определены в данном приложении. Для ознакомления с полным списком всех доступных ресурсов см. официальную документацию по адресу <http://docs.djangoproject.com/en/dev/ref/models/fields/#field-types>.

### 11.6.1. Подготовка базы данных

Тем читателям, в распоряжении которых еще нет установленного и работающего сервера базы данных, рекомендуем SQLite в качестве наиболее удобного способа для начала разработки. Это быстродействующая, широко доступная база данных, которая

обеспечивает хранение всех данных в виде одного файла. Для управления доступом достаточно просто задать разрешения для этого файла. Читатели, имеющие установленный сервер базы данных (MySQL, PostgreSQL или Oracle) и желающие использовать его, а не SQLite, должны с помощью средств администрирования для своей базы данных создать новую базу данных для проекта Django. В рассматриваемых примерах база данных именуется `mysite.db`, но при желании можно применить базу данных с другим именем.

## Использование СУБД MySQL

После создания (пустой) базы данных необходимо предоставить платформе Django указания по ее применению. И в этом случае следует использовать файл `settings.py` для проекта. Предусмотрены шесть параметров, которые может потребоваться задать в этом файле (хотя их количество может ограничиваться только двумя): `ENGINE`, `NAME`, `HOST`, `PORT`, `USER` и `PASSWORD`. Имена этих параметров в основном говорят сами за себя. Достаточно задать правильные значения, которые соответствуют серверу базы данных, используемому с Django. Например, параметры для MySQL могут выглядеть примерно так:

```
DATABASES = {
 'default': {
 'ENGINE': 'django.db.backends.mysql',
 'NAME': 'testdb',
 'USER': 'wesley',
 'PASSWORD': 's3Cr3T',
 'HOST': '',
 'PORT': '',
 }
}
```

Следует учитывать, что если используется более старая версия Django, то вместо общего словаря, содержащего все параметры, необходимые настройки должны быть заданы в виде отдельных переменных, определяемых на уровне модуля.

Параметр `PORT` не задан, поскольку необходимость в этом возникает лишь в том случае, если для эксплуатации сервера базы данных применяется нестандартный порт. Например, для сервера MySQL по умолчанию используется порт 3306. Если не предусмотрено внесение изменений в эту настройку, то не требуется указывать и `PORT`. Параметр `HOST` был оставлен пустым для указания на то, что сервер базы данных эксплуатируется на том же компьютере, на котором выполняется само приложение. Следует обязательно выполнить команду `CREATE DATABASE testdb`, где `testdb` представляет собой имя создаваемой базы данных, и убедиться в том, что уже задан пользователь (с паролем), поскольку лишь при этих условиях можно работать с платформой Django. Настройка PostgreSQL в большей степени напоминает настройку MySQL, чем Oracle.

Подробные сведения о подготовке к работе новых баз данных и пользователей, а также об определении необходимых параметров приведены в документации Django по адресу <http://docs.djangoproject.com/en/dev/intro/tutorial01/#database-setup> и <http://docs.djangoproject.com/en/dev/ref/settings/#std:setting-DATABASES>. См. также приложение В книги *Python Web Development with Django*.

## Использование SQLite

Для проверки разрабатываемых приложений широко применяется база данных SQLite. Эта база данных является также приемлемым кандидатом для развертывания в тех сценариях, когда количество одновременно выполняемых операций записи не слишком велико. Информация о хосте, порте, пользователе и пароле не требуется, поскольку в SQLite для хранения данных используется локальная файловая система, а для управления доступом — собственные разрешения файловой системы; предусмотрена также возможность эксплуатировать эту базу данных с размещением данных исключительно в оперативной памяти. Поэтому в следующем коде в составе конфигурации DATABASES в сценарии settings.py для настройки Django в целях применения базы данных SQLite заданы лишь параметры ENGINE и NAME.

```
DATABASES = {
 'default': {
 'ENGINE': 'django.db.backends.sqlite3',
 'NAME': '/tmp/mysite.db', # use full pathname to avoid confusion
 }
}
```

Если SQLite используется с таким веб-сервером производственного назначения, как Apache, то необходимо обеспечить, чтобы учетной записи, которая является владельцем процесса веб-сервера, было предоставлено право на запись и в самом файле базы данных, и в каталоге, содержащем этот файл базы данных. В данном случае работа ведется с сервером разработки, поэтому предоставление разрешений не вызывает затруднений, поскольку пользователь, эксплуатирующий сервер разработки (читатель), владеет также файлами проекта и каталогами.

### 2.5

Кроме того, база данных SQLite относится к числу наиболее часто применяемых в персональных компьютерах с операционной системой Windows, поскольку она входит в состав дистрибутива Python (начиная с версии 2.5). Предположим, что для проекта (и приложения) уже создана папка C:\py\django. Затем следует создать также каталог db и указать имя файла базы данных, которая будет создана позже:

```
DATABASES = {
 'default': {
 'ENGINE': 'django.db.backends.sqlite3',
 'NAME': r'C:\py\django\db\mysite.db', # full pathname
 }
}
```

Разработчики, имеющие опыт работы на языке Python в течение некоторого времени, должны знать, что буква r перед строкой с именем папки обозначает данную строку в Python как бесформатную. Это просто означает, что каждый символ строки воспринимается буквально и преобразование специальных символов не происходит. Иными словами, конструкция \n интерпретируется как обратная косая черта (\), за которой следует буква n, а не как единственный символ обозначения конца строки. Бесформатные строки Python в основном применяются в двух случаях, для обозначения имен путей файлов DOS и для формирования регулярных выражений. Дело в том, что и те и другие часто включают символ обратной косой черты, который служит в языке Python в качестве специального экранирующего символа. Дополнительные сведения о строках приведены в главе "Последовательности" книг *Core Python Programming* и *Core Python Language Fundamentals*.



## 11.6.2. Создание таблиц

Теперь необходимо передать Django указание об использовании информации о соединении, которая должна быть задана для подключения к базе данных и создания таблиц, необходимых для приложения. Как показано в следующем примере кода, для этого используются сценарий `manage.py` и предусмотренная в нем команда `syncdb`:

```
$./manage.py syncdb
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table blog_blogpost
```

После получения команды `syncdb` на платформе Django производится поиск файла `models.py` в каждом отдельном параметре в `INSTALLED_APPS`. Для каждой найденной модели создается таблица базы данных. (Из этого правила также есть исключения, но обычно оно выполняется неукоснительно.) Если используется база данных SQLite, то необходимо учитывать, что файл базы данных `mysite.db` создается именно там, где указано в параметрах.

Для всех других параметров в `INSTALLED_APPS` (в том числе заданных по умолчанию) также предусмотрены модели. Это подтверждается выводом команды `manage.py syncdb`; можно видеть, что в Django создаются одна или несколько таблиц для каждого из указанных приложений. На этом вывод команды `syncdb` не исчерпывается. Предусмотрено также несколько интерактивных запросов, связанных с приложением `django.contrib.auth` (см. следующий пример). Теперь рекомендуется создать учетную запись суперпользователя, поскольку она вскоре потребуется. Ниже показано, как работает этот процесс, по результатам заключительной части команды `syncdb`.

```
You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'wesley'):
E-mail address: ****@****.com
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
No fixtures found.
```

Теперь в системе авторизации имеется один суперпользователь (если все сделано правильно, это должны быть вы сами). Такая учетная запись вскоре потребуется, после перехода к использованию приложения автоматического администрирования Django.

Процесс начальной установки завершается строкой, которая касается средства, обозначенного как постоянный ресурс (fixture). Это сериализованное, введенное ранее содержимое базы данных. Постоянные ресурсы можно использовать для предварительной загрузки необходимых данных в любые вновь созданные приложения. На этом начальная установка базы данных заканчивается. В следующий раз при выполнении команды `syncdb` применительно к тому же проекту (необходимость в этом возникает каждый раз при добавлении приложения или модели) объем вывода будет гораздо меньше, поскольку не потребуется повторно создавать какие-либо из имеющихся таблиц или выводить запрос на создание суперпользователя.

На этом часть настройки приложения, касающаяся создания модели данных, завершается. Приложение уже может применяться для получения данных, введенных пользователем, но способ ввода данных еще не определен. Если во время настройки предусмотрено использование только шаблона контроллера “модель–представление” (model-view controller — MVC) для проекта веб-приложения, то действительно и будет создана только эта модель. Еще не заданы какие-либо дополнительные представления (код HTML для работы с пользователем, код поддержки шаблонов и т.д.) или контроллеры (программные средства приложения).



### Сравнение MVC и MTV

В сообществе Django используется альтернативное представление шаблона MVC. В Django оно именуется как “модель–шаблон–представление” (model-template-view — MTV). Модель данных в Django остается той же, но вместо термина “представление” применяется термин “шаблон”, поскольку на этой платформе для задания внешнего интерфейса, с которым работают пользователи, применяются шаблоны. Наконец, следует отметить, что в Django термин “представление” тоже используется, но обозначает функции работы с представлением, совокупность которых формирует все программные средства контроллера. По существу эти термины аналогичны, но указывают на разную интерпретацию выполняемых ролей. Чтобы больше узнать о том, какая трактовка этой проблематики применяется в Django, ознакомьтесь с ответами на вопросы по адресу <http://docs.djangoproject.com/en/dev/faq/general/#djangoappears-tobe-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names>.

## 11.7. Командный интерпретатор для приложений Python

Программисты на Python знают, насколько удобно работать с интерактивным интерпретатором. Об этом знают и создатели платформы Django, поэтому предусмотрели возможность работы в интерактивном режиме и в ходе повседневной разработки Django. В следующих подразделах будет показано, как использовать командный интерпретатор Python для проверки значений переменных на низком уровне и манипулирования этими значениями, что обычно не так легко достигается в ходе разработки веб-приложений.

### 11.7.1. Использование командного интерпретатора Python в Django

Предусмотрена возможность проверки модели данных даже без шаблона (представления) или представления (контроллера), для чего следует добавить некоторые записи в BlogPost. Если для хранения данных приложения применяется реляционная СУБД (как и в большинстве приложений Django), то каждая запись блога преобразуется в строки таблиц. Если же используется база данных, отличная от SQL, такая как MongoDB или хранилище данных Google App Engine, то вместо строк в базу данных добавляются объекты, документы или сущности.

Для работы с базой данных на платформе Django предусмотрен командный интерпретатор Python, который можно использовать для создания экземпляров моделей и выполнения других операций взаимодействия с приложением. Пользователи Python после ввода команды `shell` сценария `manage.py` увидят знакомые строки запуска интерактивного интерпретатора и приглашения:

```
$ python2.5 ./manage.py shell
Python 2.5.1 (r251:54863, Feb 9 2009, 18:49:36)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

Этот командный интерпретатор Django отличается от стандартного командного интерпретатора Python тем, что первый, в отличие от последнего, поддерживает дополнительные возможности работы со средой проекта Django. Предусмотрена возможность работать с функциями представления и моделями данных, поскольку командный интерпретатор автоматически устанавливает переменные среды, включая `sys.path`, что позволяет получать доступ к модулям и пакетам не только Django, но и проекта. В противном случае такую задачу приходилось бы решать вручную. В дополнение к стандартному командному интерпретатору Python предусмотрен целый ряд дополнительных интерактивных командных интерпретаторов, которые отчасти представлены в главе 1 книг *Core Python Programming* и *Core Python Language Fundamentals*.

Фактически для Django являются более предпочтительными такие развитые командные интерпретаторы, как IPython и bpython, поскольку в них кроме стандартных функций предусмотрены чрезвычайно удобные расширения. После вызова на выполнение команды `shell` платформа Django в первую очередь выполняет поиск расширенного командного интерпретатора и использует первый из найденных; в противном случае, если таковой не обнаруживается, происходит возврат к стандартному интерпретатору.

В предыдущем примере использовался интерпретатор Python 2.5 без дополнительных функций и поэтому произошел вызов именно его. Если же команда `manage.py shell` вызывается в среде с расширенным интерпретатором (допустим, IPython), то к работе приступает именно он:

```
$./manage.py shell
Python 2.7.1 (r271:86882M, Nov 30 2010, 09:39:13)
[GCC 4.0.1 (Apple Inc. build 5494)] on darwin
Type "copyright", "credits" or "license" for more information.
```

```
IPython 0.10.1 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.
```

```
In [1]:
```

Можно также воспользоваться опцией `--plain` для принудительного вызова стандартного интерпретатора:

```
$./manage.py shell --plain
Python 2.7.1 (r271:86882M, Nov 30 2010, 09:39:13)
[GCC 4.0.1 (Apple Inc. build 5494)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

Заслуживает внимание то, что наличие или отсутствие развитого командного интерпретатора зависит от установленной версии Python, как показано в предыдущем примере; просто так оказалось, что на компьютере автора установлена версия Python 2.7, притом что интерпретатор IPython имеется только для этой версии, но не для версии 2.5.

Чтобы установить командный интерпретатор с дополнительными функциями, достаточно воспользоваться средством `easy_install` или `pip`, как было указано ранее при описании различных методов установки Django. Ниже показана последовательность установки IPython на персональном компьютере с операционной системой Windows.

```
C:\WINDOWS\system32>\python27\Scripts\easy_install ipython
Searching for ipython
Reading http://pypi.python.org/simple/ipython/
Reading http://ipython.scipy.org
Reading http://ipython.scipy.org/dist/0.10
Reading http://ipython.scipy.org/dist/0.9.1
...
Installing ipengine-script.py script to c:\python27\Scripts
Installing ipengine.exe script to c:\python27\Scripts
Installed c:\python27\lib\site-packages\ipython-0.10.1-py2.7.egg
Processing dependencies for ipython
Finished processing dependencies for ipython
```

## 11.7.2. Эксперименты с моделью данных

В предыдущем разделе было показано, как запустить командный интерпретатор Python, а теперь перейдем к рассмотрению работы с приложением и его моделью данных. Для этого необходимо выполнить запуск IPython и выдать несколько команд Python или IPython:

```
In [1]: from datetime import datetime
In [2]: from blog.models import BlogPost
In [3]: BlogPost.objects.all() # no objects saved yet!
Out[3]: []
In [4]: bp = BlogPost(title='test cmd-line entry', body='')
....: yo, my 1st blog post...
```

```
....: it's even multilined!''',
....: timestamp=datetime.now())
In [5]: bp
Out[5]: <BlogPost: BlogPost object>
In [6]: bp.save()
In [7]: BlogPost.objects.count()
Out[7]: 1
In [8]: exec _i3 # repeat cmd #3; should have 1 object now
Out[8]: [<BlogPost: BlogPost object>]
In [9]: bp = BlogPost.objects.all()[0]
In [10]: print bp.title
test cmd-line entry
In [11]: print bp.body # yes an extra \n in front, see above

yo, my 1st blog post...
it's even multilined!
In [12]: bp.timestamp.ctime()
Out[12]: 'Sat Dec 11 16:38:37 2010'
```

Первые несколько команд лишь вводят в действие необходимые объекты. В шаге 3 выполняется запрос к базе данных на получение объектов `BlogPost`, которые в настоящее время отсутствуют, поэтому в шаге 4 должен быть добавлен в базу данных первый объект. Для этого создается экземпляр объекта `BlogPost` и передаются его атрибуты, которые были заданы ранее (`title`, `body` и `timestamp`). После создания объекта необходимо записать его в базу данных (шаг 6) с помощью метода `BlogPost.save()`.

Вслед за выполнением этих действий можно проверить, изменилось ли число объектов в базе данных с 0 на 1. Для этого применяется метод `BlogPost.objects.count()` (шаг 7). В шаге 8 используется команда `IPython` для повтора шага 3 (получения списка всех объектов `BlogPost`, хранимых в базе данных). Для этого можно было бы просто еще раз задать команду `BlogPost.objects.all()`, но было решено продемонстрировать более мощное средство командного интерпретатора. В последних шагах осуществляется выборка первого (и единственного) элемента из списка всех объектов `BlogPost` (шаг 9) и выгрузка всех данных. Это позволяет показать, что можно успешно извлечь все данные, которые были сохранены ранее.

Приведенный выше пример демонстрирует лишь часть того, что можно сделать с помощью интерактивного интерпретатора, подключенного к приложению. Дополнительные сведения о средствах командного интерпретатора можно получить по адресу <http://docs.djangoproject.com/en/dev/intro/tutorial01/#playing-with-the-api>. Такие командные интерпретаторы Python представляют собой весьма удобные инструменты для разработчика. Безусловно, в составе Python предусмотрен стандартный инструмент с интерфейсом командной строки, однако аналогичные инструменты не только включены в различные варианты интегрированной среды разработки (*integrated development environment* — IDE), но и дополнены еще более широким набором функциональных средств. В качестве примера можно указать такие интерактивные интерпретаторы, созданные сторонними разработчиками, как `IPython` и `bpython`.

Тем не менее почти все пользователи и многие разработчики предпочитают вместо этого использовать инструменты CRUD (*create, read, update, delete* — создать, прочитать, обновить, удалить) на основе веб-интерфейса. Это же касается почти любого разработанного веб-приложения. Рассмотрим причины, по которым разработчики так стремятся предусмотреть административную веб-консоль для каждого создаваемого ими приложения. Прежде всего создается впечатление, что потребность в создании подобного интерфейса возникает постоянно, поэтому административные приложения на базе Django нашли такое широкое распространение.

## 11.8. Приложение администрирования Django

В литературе встречалось даже такое утверждение, что автоматизированные серверные административные приложения (которые принято кратко называть `admin`) являются одной из драгоценностей в короне Django. Эти приложения позволяют вдохнуть с облегчением любому разработчику, которому надоело постоянно создавать ограниченные по своим возможностям интерфейсы CRUD для веб-приложений. Приложение `admin` стало необходимым для каждого создаваемого веб-сайта. Рассмотрим, с чем это связано. Прежде всего, необходимо наделить веб-приложение способностью вставлять новые записи, а также обновлять их и удалять. Это очевидно, но далеко не исчерпывает функции, которые должны быть возложены на приложение, и после того, как выявляется истинный объем предстоящей работы, начинаются настоящие трудности. Приложение `admin` способствует решению многих проблем, с которыми сталкиваются разработчики, поскольку позволяет проверять подготавливаемый код манипулирования данными задолго до полного завершения работы над пользовательским интерфейсом.

### 11.8.1. Настройка приложения `admin`

Безусловно, приложение `admin` предоставляется бесплатно в составе Django, но все еще требуется явно разрешить его использование с помощью параметров конфигурации, по аналогии с тем, что было сделано в рассматриваемом ранее приложении блога. Откроем сценарий `settings.py` и перейдем ближе к его концу, где дано определение кортежа `INSTALLED_APPS`, как уже было сделано ранее. Выше уже шла речь о том, что должно быть добавлено объявление переменной `'blog'`, но следует также уделить внимание четырем строкам, находящимся непосредственно над этим объявлением:

```
INSTALLED_APPS = (
 . . .
 # Uncomment the next line to enable the admin:
 # 'django.contrib.admin',
 # Uncomment the next line to enable admin documentation:
 # 'django.contrib.admindocs',
 'blog',
)
```

Прежде всего нас интересует первая закомментированная строка, `'django.contrib.admin'`. Удалите в начале строки символ номера (`#`), называемый также восьмиконечником, знаком дизеля, знаком фунта или обозначением комментария, чтобы включить эту строку в код. Вторая строка — необязательная; она представляет генератор документации программы `admin` в составе Django. Приложение `admindocs` автоматически формирует документы для проекта, извлекая строки документации Python (docstrings) и предоставляя доступ к этим строкам для `admin`. Решение о том, разрешать ли использование этого приложения, передается на усмотрение читателя, но в данном примере оно не рассматривается.

После добавления каждого нового приложения к проекту необходимо выполнить команду `syncdb`, чтобы обеспечить создание в базе данных требуемых таблиц. В данном примере можно видеть, что после добавления приложения `admin` к списку `INSTALLED_APPS` и выполнения команды `syncdb` в базе данных создается еще одна таблица:

```
$./manage.py syncdb
Creating tables ...
Creating table django_admin_log
Installing custom SQL ...
Installing indexes ...
No fixtures found.
```

Теперь, после установки приложения, достаточно присвоить этому приложению URL для обеспечения к нему доступа. В файле `urls.py`, автоматически формируемом для проекта, ближе к его началу появляются следующие строки:

```
Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()
```

Кроме того, ближе к концу объявления глобальной переменной `urlpatterns` обнаруживается следующая закомментированная строка с двухэлементным кортежем:

```
Uncomment the next line to enable the admin:
(r'^admin/', include(admin.site.urls)),
```

Раскомментируйте все эти три строки, фактически присутствующие в коде, и сохраните файл. Это равносильно передаче Django указания о том, что при переходе посетителей на веб-сайт с помощью URL `http://localhost:8000/admin` должен загружаться заданный по умолчанию сайт `admin`.

Наконец, в приложениях необходимо предусматривать передачу Django указания о том, какие модели должны предоставляться для редактирования в экранах `admin`. Для этого достаточно зарегистрировать в Django модель `BlogPost`. Подготовьте сценарий `blog/admin.py`, состоящий из следующих строк:

```
admin.py
from django.contrib import admin
from blog import models

admin.site.register(models.BlogPost)
```

В первых двух строках осуществляется импорт приложения `admin` и применяемых моделей данных. За ними следует строка, в которой регистрируется класс `BlogPost` для приложения `admin`. Это позволяет обеспечить, чтобы приложение `admin` управляло в базе данных объектами этого типа (в дополнение к другим, зарегистрированным ранее).

## 11.8.2. Проверка работы с приложением `admin`

После регистрации модели для работы с приложением `admin` приступим к экспериментам с этим приложением. Снова вызовите на выполнение команду `manage.py runserver`, а затем перейдите по той же ссылке, что и раньше (`http://127.0.0.1:8000` или `http://localhost:8000`). Рассмотрим полученные результаты. Если все сделано правильно, то фактически должна быть получена ошибка. А именно: должна появиться ошибка 404, которая отображается так, как показано на рис. 11.5.

С чем связано появление этой ошибки? Причина состоит в том, что для URL еще не задана операция ('/'). Как было описано выше, для данного приложения разрешена только операция `/admin`, поэтому необходимо указать ее непосредственно в URL.

Это означает, что должен быть выполнен переход по адресу `http://127.0.0.1:8000/admin` или `http://localhost:8000/admin`. Еще один вариант состоит в том, что можно добавить подстроку `/admin` к существующему пути в браузере.



Рис. 11.5. Экран регистрации admin

И действительно, внимательное изучение экрана с сообщением об ошибке показывает, что даже Django сообщает о доступности только пути `/admin`, поскольку на этой платформе вначале осуществляется проверка других возможных вариантов и лишь за тем появляется сообщение об ошибке. Заслуживает внимания то, что формирование страницы с сообщением “It Worked!” является частным случаем для той ситуации, в которой еще не задан URL для приложения. (Если бы не был предусмотрен этот частный случай, то была бы также получена ошибка 404.)

После успешного открытия приложения admin появится запрос на регистрацию в виде хорошо оформленного и удобного в использовании экрана (рис. 11.6).

Отображаемые сведения представляют собой множество всех классов, зарегистрированных для работы с приложением admin. Приложение admin дает возможность манипулировать всеми этими классами, данные которых находятся в базе данных, включая Users; это означает, что теперь можно добавлять учетные записи для стандартных (рядовых) пользователей или других суперпользователей (имея возможность при этом работать с удобным веб-интерфейсом, а не с командной строкой или средой командного интерпретатора).

Введите логин и пароль суперпользователя, созданные ранее. После регистрации откроется начальная страница admin, как показано на рис. 11.7.





Рис. 11.6. Экран регистрации admin

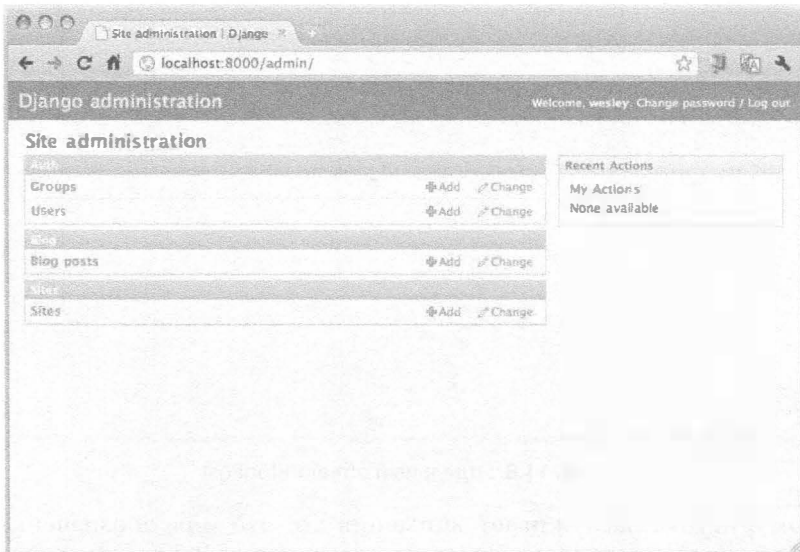


Рис. 11.7. Начальная страница admin



### Класс не обнаруживается

Действительно, иногда пользовательские классы не появляются в списке. Ниже описаны три наиболее распространенные причины, по которым данные пользовательского приложения не обнаруживаются приложением `admin`.

1. Упущение, которое заключается в том, что класс модели не зарегистрирован с помощью метода `admin.site.register()`
2. Ошибки в файле `models.py` приложения
3. Упущение, связанное с тем, что приложение не добавлено к кортежу `INSTALLED_APPS` в файле `settings.py`.

Теперь перейдем к рассмотрению примеров, позволяющих понять истинные возможности `admin` при управлении данными. После щелчка на ссылке “Blog posts” (публикации блога) происходит переход на страницу со списком всех объектов `BlogPost` в базе данных (рис. 11.8). На данный момент в списке присутствуют только те объекты, которые были введены с помощью командного интерпретатора.

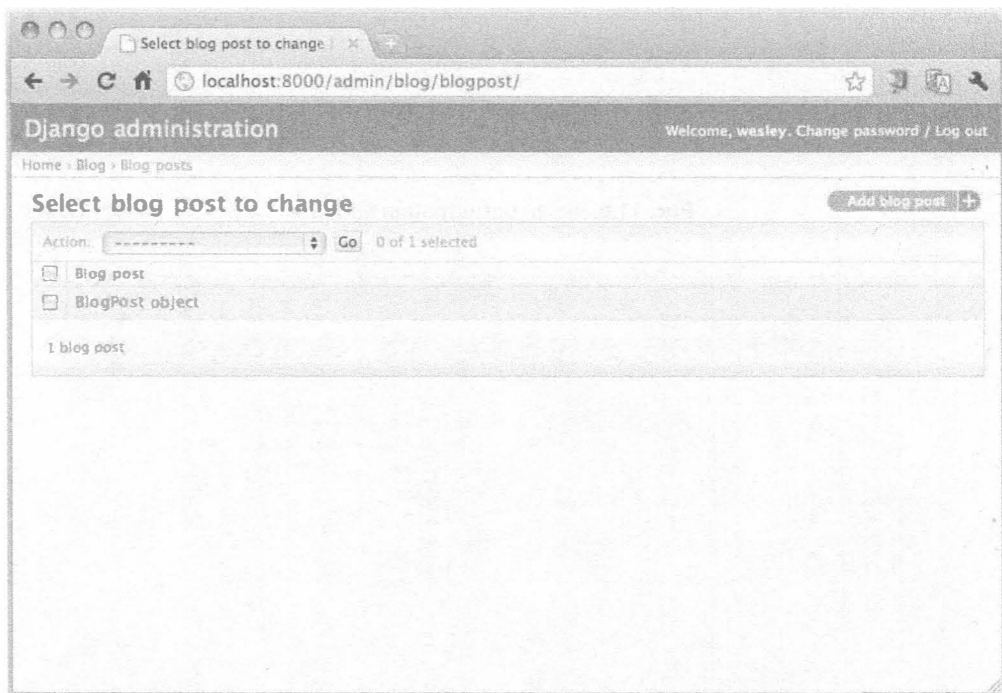


Рис. 11.8. Отдельный объект `BlogPost`

На этом рисунке заслуживает внимания то, что для обозначения объекта `BlogPost` применяется тег весьма общего содержания. Рассмотрим, почему публикации присвоено такое неудобное имя. Платформа Django спроектирована так, чтобы с ее помощью можно было обрабатывать бесконечно разнообразные типы содержимого, поэтому не выдвигаются какие-либо предположения в отношении того, какое поле могло бы лучше всего представлять какой-то конкретный

фрагмент содержимого. В результате принятое решение является прямолинейным и не представляет интереса.

В данном случае нет сомнений в том, что именно публикация лучше всего представляет данные, введенные ранее, к тому же невозможно спутать эту запись с другими объектами BlogPost, поэтому дополнительные сведения об этом объекте не требуются. Продолжите работу и щелкните на указанной ссылке, чтобы перейти в окно редактирования, показанное на рис. 11.9.

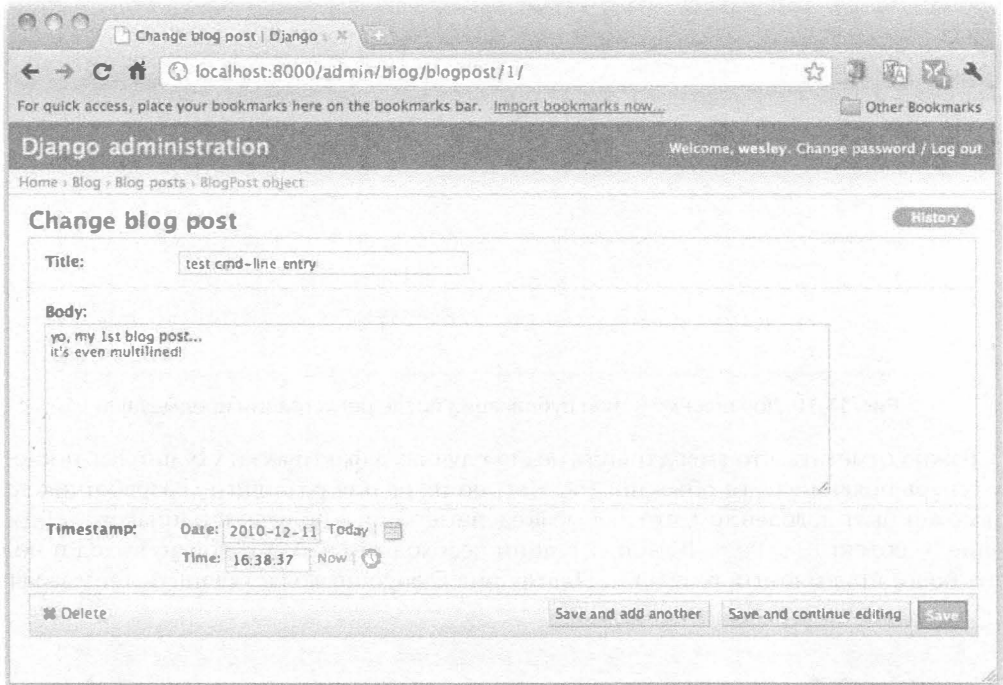


Рис. 11.9. Представление записи командной строки BlogPost в веб-интерфейсе

Внесите любые желаемые изменения (если это потребуется), а затем щелкните на кнопке Сохранить и внесите еще одну запись, чтобы можно было провести эксперименты с добавлением записи с помощью веб-формы (а не с применением командного интерпретатора). Как показано на рис. 11.10, эта форма идентична той, которая применялась только что для редактирования предыдущей публикации.

Рассмотрим новую публикацию BlogPost без содержимого. Назначим ей заголовок и введем привлекательное содержимое, допустим, такое, как показано на рис. 11.11. Для формирования отметки времени можно щелкнуть на вспомогательных ссылках Today (Сегодня) и Now (Сейчас), чтобы ввести текущую дату и время. Можно также щелкнуть на значках календаря и часов, чтобы воспользоваться удобными средствами выбора даты и времени. Закончив работу над своим шедевром, нажмите на кнопке Сохранить.

После сохранения публикации в базе данных откроется окно с подтверждающим сообщением ("The blog post, BlogPost object, was added successfully" — "Успешное добавление публикации, объекта BlogPost"), а также список всех прочих сформированных вами публикаций в блоге (рис. 11.12).

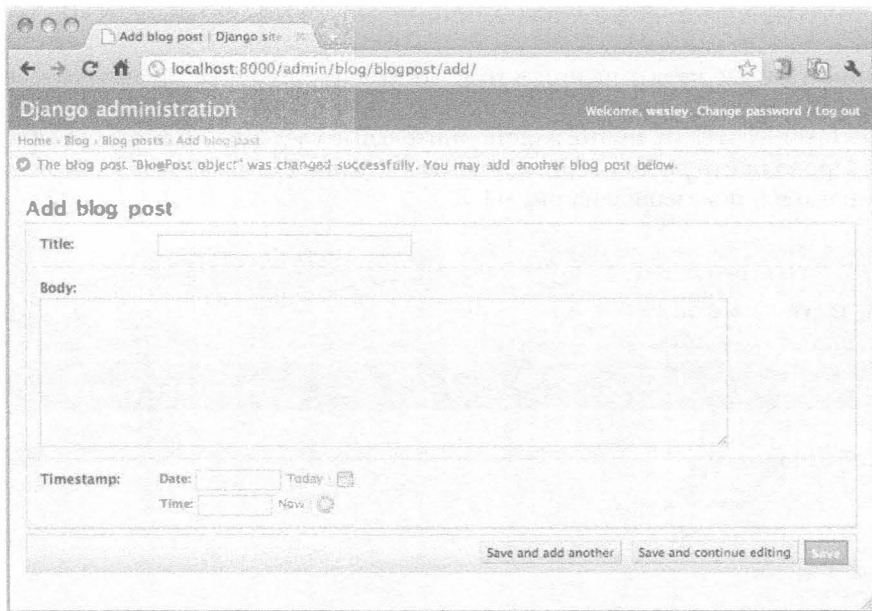


Рис. 11.10. Добавление новой публикации после регистрации предыдущей

Важно отметить, что вывод ничуть не стал лучше, а фактически ухудшился, поскольку теперь появились два объекта `BlogPost`, но их нельзя различить. Разработчик вряд ли может быть доволен тем, что любые введенные записи получают одинаковое обозначение — объект `BlogPost`. В этой ситуации необходимо найти какой-то выход и получить более приемлемый результат. Платформа Django позволяет решить и эту задачу.



Рис. 11.11. Добавление новой публикации непосредственно из приложения admin



Рис. 11.12. Результат сохранения новой публикации BlogPost. Теперь в базе данных хранятся две публикации, но возможность отличить их друг от друга отсутствует

Выше было показано, как ввести в действие инструмент admin с абсолютно минимальной конфигурацией, а именно: как зарегистрировать модель в приложении admin без каких-либо дополнений. Тем не менее достаточно ввести в код еще две строки, и модифицировать вызов команды регистрации и можно значительно улучшить представление списков и сделать его более полезным. Обновите файл `blog/admin.py`, определив в нем новый класс `BlogPostAdmin`, и добавьте его к строке регистрации, чтобы она выглядела следующим образом:

```
admin.py
from django.contrib import admin
from blog import models

class BlogPostAdmin(admin.ModelAdmin):
 list_display = ('title', 'timestamp')

admin.site.register(models.BlogPost, BlogPostAdmin)
```

Следует отметить, что в связи с определением класса `BlogPostAdmin` не применяется префикс в виде атрибута модуля `blog/models.py`; иными словами, регистрация метода `models.BlogPostAdmin` не проводится. После обновления страницы приложения admin для вывода объектов `BlogPost` (рис. 11.13) формируется намного более удобный вывод благодаря использованию новой переменной `list_display`, которая добавлена к классу `BlogPostAdmin`:

Окно, приведенное на рис. 11.13, выглядит намного лучше, поскольку включает содержательную информацию, а не просто указания на пару объектов `BlogPost`. Разработчики, впервые приступающие к использованию платформы Django, часто

удивляются тому, что добавления двух строк и редактирования третьей достаточно для изменения вывода в направлении его значительного улучшения.

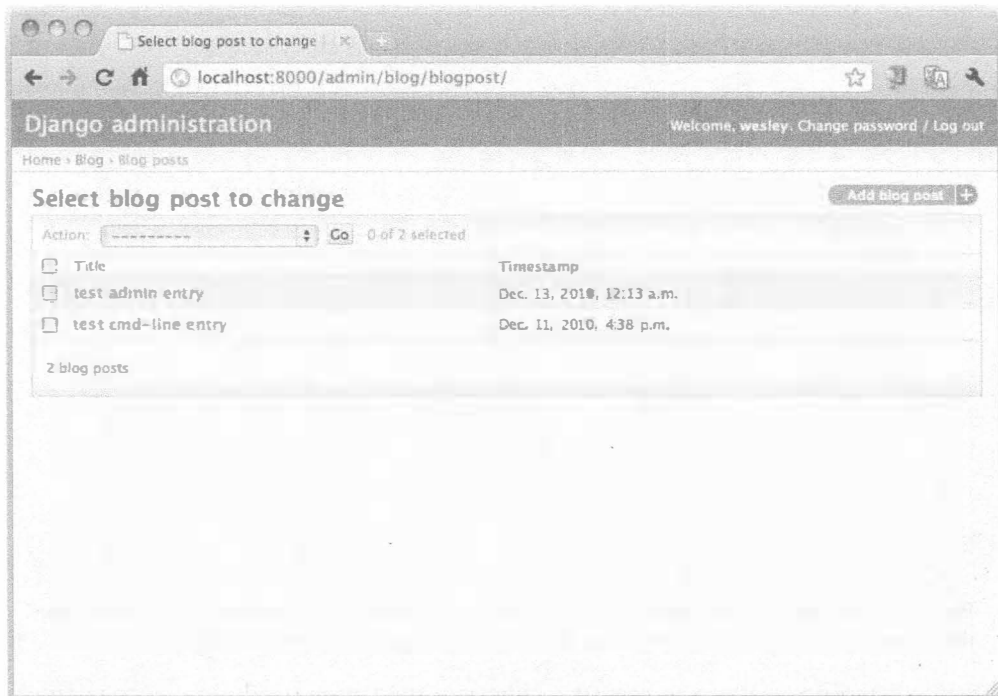


Рис. 11.13. Результаты выполненной доработки

Проведите эксперименты, щелкая на заголовках столбцов Title (Название) и Timestamp (Отметка времени); с помощью этих столбцов можно изменять порядок сортировки элементов. Например, после одного щелчка на заголовке столбца Title происходит сортировка записей в порядке возрастания лексикографических значений заголовков; после второго щелчка сортировка происходит в порядке убывания. Кроме того, попытайтесь провести сортировку по отметкам времени. Любопытно то, что эти возможности уже предусмотрены в приложении admin, и нам осталось лишь воспользоваться ими. От разработчика уже не требуется, чтобы он сам реализовывал все функции, как “в старые добрые времена”.

В приложении admin предусмотрено много других полезных средств, которые можно вводить в действие с помощью всего лишь одной или двух строк кода: поиск, определяемое пользователем упорядочение, фильтры и т.д. Мы едва коснулись возможностей приложения admin, но можно надеяться, что читатель почувствовал вкус к их использованию.

## 11.9. Создание пользовательского интерфейса для блога

Можно смело утверждать, что платформа Django создана исключительно ради полного удовлетворения потребностей разработчика. Каким бы ни было намеченное

приложение, его всегда можно разработать так, чтобы пользователям не приходилось работать непосредственно с командным интерпретатором Django, а также, скорее всего, и с инструментом `admin`. Теперь перейдем к созданию той части приложения, которая предназначена для самих пользователей. С точки зрения Django веб-страница имеет три типичных компонента.

Шаблон, с помощью которого отображается информация, переданная на страницу (с помощью объекта Python, подобного словарию).

Функция представления, или просмотра, в которой реализуются основные алгоритмы обработки запроса. Именно эта функция чаще всего применяется для выборки (и форматирования) отображаемой информации, для чего, как правило, используется база данных.

Шаблон URL, который согласует входящий запрос с соответствующим представлением и в случае необходимости передает также в это представление параметры.

Рассматривая выполняемые операции, можно обнаружить, что при обработке запроса на платформе Django необходимые действия осуществляются снизу вверх: вначале происходит поиск сопоставляемого шаблона URL. Затем вызывается соответствующая функция представления, которая возвращает пользователю данные, подготовленные к просмотру с помощью шаблона.

На этот раз примем немного иную последовательность формирования приложения.

1. В первую очередь формируется основной шаблон, без которого невозможно обеспечить просмотр материала.
2. Разработка шаблона URL, с помощью которого можно предоставить платформе Django доступ к создаваемому приложению, не требует больших трудозатрат.
3. Для этого достаточно подготовить прототип, а затем возвращаться к нему в ходе разработки функции представления.

Основная причина, по которой выбрана такая последовательность, состоит в том, что шаблон приложения и шаблон URL не подвержены значительным изменениям. Данное приложение главным образом предназначено для просмотра данных, поэтому необходимо предусмотреть способ динамического формирования данных для просмотра. Представление данных подготавливается поэтапно, во многом аналогично тому, как в модели разработки, основанной на тестировании (test-driven development — TDD).

### 11.9.1. Создание шаблона

Поскольку язык шаблонов Django настолько прост для восприятия, что можно обойтись без его подробного обсуждения, мы можем сразу перейти к рассмотрению кода примера. Это простой шаблон для просмотра отдельных публикаций блога (с учетом атрибутов объекта `BlogPost`):

```
<h2>{{ post.title }}</h2>
<p>{{ post.timestamp }}</p>
<p>{{ post.body }}</p>
```

Читатель мог заметить, что рассматриваемый код состоит из тегов HTML (хотя шаблоны Django могут применяться для вывода текста любого типа), которые

дополненными специальными тегами в фигурных скобках: `{{ ... }}`. Эти теги принято называть тегами переменных. Они отображают содержимое объекта, указанного в фигурных скобках. В теге переменной можно использовать точечную систему обозначений в стиле Python для получения доступа к атрибутам переменных. В качестве значений могут быть указаны данные в чистом виде или вызываемые модули. В последнем случае происходит автоматический вызов, причем не требуется включать круглые скобки, `()`, для указания на то, что имеет место вызов функции/метода.

Предусмотрены также специальные функции, которые можно использовать в тегах переменных, называемые фильтрами. Это функции, которые можно немедленно применять к переменной, указанной в теге. Для этого достаточно вставить символ канала `(|)` непосредственно после переменной, а затем указать имя фильтра. Например, чтобы преобразовать в прописные первые буквы в заголовке `BlogPost`, достаточно вызвать фильтр `title()`, как в следующем примере:

```
<h2>{{ post.title|title }}</h2>
```

Таким образом, при обнаружении в шаблоне переменной `post.title` (со значением "test admin entry") в результирующем выводе HTML появится тег `<h2>Test Admin Entry</h2>`.

Переменные передаются в шаблон с помощью специального словаря Python, который принято называть контекстом. В предыдущем примере предполагается использование объекта `BlogPost`, именуемого как "post", который передается с помощью контекста. Эти три строки шаблона предназначены для выборки в шаблоне полей заголовка, текста и временной отметки объекта `BlogPost` соответственно. Шаблон станет немного более удобным после внесения в него некоторых усовершенствований. В частности, можно предусмотреть передачу всех публикаций блога с помощью контекста, чтобы можно было обрабатывать их в цикле и отображать:

```
<!-- archive.html -->
{% for post in posts %}
 <h2>{{ post.title }}</h2>
 <p>{{ post.timestamp }}</p>
 <p>{{ post.body }}</p>
 <hr>
{% endfor %}
```

Исходные три строки остаются неизменными; внесенное изменение состоит лишь в том, что указанные действия выполняются в цикле, который охватывает все публикации. Таким образом, была показана еще одна конструкция языка шаблонов Django: блочные теги. Для тегов переменных разграничителями служат пары фигурных скобок, а в блочных тегах в качестве открывающих и закрывающих разграничителей применяются фигурные скобки и знаки процента: `{% ... %}`. С помощью блочных тегов можно включать в шаблоны HTML такие программные конструкции, как циклы и условные выражения.

Сохраните приведенный выше код шаблона HTML в виде простого шаблона в файле `archive.html` и поместите этот файл в каталог `templates` в папке приложения; таким образом, путь к файлу шаблона принимает вид `mysite/blog/templates/archive.html`. Имя самого шаблона может быть выбрано произвольно (файл шаблона можно было бы назвать пусть даже `foo.html`), но имя каталога `templates` является обязательным. По умолчанию при поиске шаблонов на платформе Django предпринимается попытка найти каталог `templates` в каждом из установленных приложений.



Дополнительные сведения о шаблонах и тегах см. на странице с официальными документами по адресу <http://docs.djangoproject.com/en/dev/ref/templates/api/#basics>.

Следующий шаг состоит в подготовке к созданию функции представления, которую пользователи в конечном итоге будут применять для просмотра вывода вновь созданного шаблона. Прежде чем приступить к созданию представления, рассмотрим эту задачу с точки зрения пользователя.

## 11.9.2. Создание шаблона URL

В этом разделе рассматривается вопрос о том, как имена путей в формате URL в браузерах пользователей обрабатываются различными частями приложения. После выдачи пользователем клиентского запроса со своего браузера происходит преобразование с помощью системы DNS имени хоста в IP-адрес. Затем клиент устанавливает соединение с сервером по указанному адресу, применяя порт 80 или другой назначенный порт (на сервере разработки Django по умолчанию используется порт 8000).

### Проект URLconf

После всей подготовительной работы сервер с помощью протокола WSGI вызывает конечную точку Django, через которую запрос передается дальше по назначению. Происходит проверка типа запроса (GET, POST и т.д.) и пути (оставшейся части URL после исключения данных, касающихся протокола, узла и порта), после чего вызывается файл URLconf проекта (`mysite/urls.py`). На данном этапе проверка (с помощью регулярного выражения) должна показать, что путь, с помощью которого происходит выполнение запроса, является допустимым; в противном случае сервер возвращает ошибку 404, как и в описанном ранее случае в разделе “Проверка работы с приложением `admin`”, когда не был задан обработчик для пути `/'`.

Безусловно, необходимый шаблон URL можно было бы создавать непосредственно в сценарии `mysite/urls.py`, но в таком случае связь между проектом и приложением не была бы столь прозрачной. Тем не менее не исключено, что разрабатываемое нами приложение блога найдет применение и в других проектах, поэтому было бы более удобно, если бы для создания необходимого URL применялось само это приложение, а не дополнительный код. Это соответствует принципам повторного использования кода, DRY, согласно которым достаточно один раз отладить код универсального назначения, затем применять его повсеместно и т.д. Для этого прежде всего необходимо разбить проект и приложение должным образом на компоненты. Затем преобразуем URL в виде двух простых шагов и создадим два файла URLconf — для проекта и для приложения.

Первый шаг весьма напоминает ввод в действие приложения `admin`, о чем уже шла речь выше в данной главе. В сценарии `mysite/urls.py` имеется автоматически сформированная и закоментированная строка, предназначенная для использования в качестве примера; она содержит почти все, что нам необходимо. Эта строка находится в верхней части определения переменной `urlpatterns`:

```
urlpatterns = patterns('',
 # Example:
 # (r'^mysite/', include('mysite.foo.urls')),
 ...
```

Раскомментируем строку и внесем необходимые изменения в имя, чтобы оно указывало на файл URLconf приложения:

```
(r'^blog/', include('blog.urls')),
```

Функция `include()` позволяет отложить выполнение необходимых действий до вызова другого файла URLconf (таковым, естественно, является файл URLconf приложения). В приведенном здесь примере обеспечивается перехват запросов, которые начинаются с `blog/`, после чего эти запросы должны быть переданы в сценарий `mysite/blog/urls.py`, к созданию которого мы сейчас приступим. (Дополнительные сведения о функции `include()` приведены ниже.)

С учетом настройки приложения `admin`, выполненной ранее, весь файл URLconf проекта должен выглядеть следующим образом:

```
mysite/urls.py
from django.conf.urls.defaults import *

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
 (r'^blog/', include('blog.urls')),
 (r'^admin/', include(admin.site.urls)),
)
```

Функция `patterns()` принимает в качестве параметров группу двухэлементных кортежей (регулярное выражение URL, назначение). Что такое регулярное выражение, особенно пояснять не требуется, но для чего предусмотрено назначение? Назначение — либо сама функция представления, вызываемая применительно к URL-адресам, которые согласуются с шаблоном, либо — вызов функции `include()` в другом файле URLconf.

Если используется функция `include()`, то удаляется заголовок текущего пути URL и остаток пути передается в функцию `patterns()` следующего файла URLconf. Например, если в браузере клиента будет введен URL `http://localhost:8000/blog/foo/bar`, то произойдет передача в файл URLconf проекта значения `blog/foo/bar`. Это значение согласуется с регулярным выражением `^blog` и позволяет найти функцию `include()` (а не функцию представления), таким образом, в соответствующий обработчик URL в сценарии `mysite/blog/urls.py` передается значение `foo/bar`.

В качестве параметра `include()` применяется `'blog.urls'`. Аналогичный сценарий выполняется при обработке значения `http://localhost:8000/admin/xxx/yyy/zzz`; в сценарии `admin/site/urls.py` передается значение `xxx/yyy/zzz`, в соответствии с вызовом `include(admin.site.urls)`. Внимательный читатель может заметить в этом фрагменте кода нечто необычное. Возможно, что-то показалось нам незначительным и было пропущено? Такие ситуации в программировании иногда сродни оптическим иллюзиям. Рассмотрим более внимательно вызов функции `include()`.

Сумел ли читатель заметить, что ссылка на `blog.urls` приведена в кавычках, а ссылка на `admin.site.urls` — нет? Нет, это не опечатка. Обе функции, `patterns()` и `include()`, принимают в качестве параметров и строки, и объекты. Обычно используются строки, но некоторые разработчики предпочитают передавать в функцию объекты, что позволяет точнее определить назначение вызова. Единственное, о чем следует помнить при передаче объектов, состоит в том, что передаваемые объекты

обязательно должны быть импортированы. В предыдущем примере это требование выполняется в результате импорта `django.contrib.admin`.

Еще один пример подобного использования объекта будет рассматриваться в следующем разделе. Чтобы больше узнать о том, чем отличается вызов строк от вызова объектов, см. страницу документации на эту тему по адресу <http://docs.djangoproject.com/en/dev/topics/http/urls/#passing-callable-objects-instead-of-strings>.

## Приложение URLconf

Если в качестве параметра функции `include()` применяется `blog.urls`, то необходимо определить URL для сопоставления остальных элементов пути в самом пакете приложений блога. Создайте новый файл, `mysite/blog/urls.py`, содержащий следующие строки:

```
urls.py
from django.conf.urls.defaults import *
import blog.views

urlpatterns = patterns('',
 (r'^$', blog.views.archive),
)
```

Этот файл весьма напоминает файл URLconf проекта, который рассматривался ранее. Прежде всего необходимо еще раз отметить, что заглавная часть (`blog/`) URL запроса, с которой сопоставляется корневой файл URLconf, уже удалена, поэтому осталось провести сопоставление с пустой строкой, для чего предназначено регулярное выражение `^$`. Теперь обеспечено повторное использование приложения блога, а это означает, что оно может применяться для любого пути, будь то `blog/`, `news/` или `what/i/had/for/lunch/`. Осталось лишь рассмотреть функцию представления `archive()`, в которую передается запрос.

Для включения новых функций представления в состав приложения достаточно ввести отдельные строки в файл URLconf, а не добавлять, скажем, десять строк здесь, редактировать еще пять строк сложного XML-файла там и т.д. Иными словами, если потребуется добавить новые функции представления `foo()` и `bar()`, то следует лишь изменить значение `urlpatterns`, чтобы оно выглядело следующим образом (хотя мы не рекомендуем вносить эти изменения в рассматриваемом приложении):

```
urlpatterns = patterns('',
 (r'^$', blog.views.archive),
 (r'foo/', blog.views.foo),
 (r'bar/', blog.views.bar),
)
```

Мы практически достигли своей цели, но по мере дальнейшей разработки с помощью платформы Django, когда приходится снова и снова открывать один и тот же файл с кодом приложения, невольно начинаешь замечать, насколько много в нем повторов. Это, безусловно, является нарушением принципа DRY. В частности, многократно повторяется подстрока `blog.views`, с помощью которой происходит доступ к функциям представления. Это может служить указанием на то, что для уменьшения объема кода следует воспользоваться одной из возможностей, предоставляемых функцией `patterns()`, а именно первым параметром, в качестве которого до сих пор применялась только пустая строка.

Этот параметр задает префикс для представлений, поэтому ему можно присвоить значение `blog.views`, удалить повторы и исправить оператор импорта, чтобы при его обработке не возникала ошибка `NameError`-. После этих изменений файл `URLconf` может выглядеть примерно так:

```
from django.conf.urls.defaults import *
from blog.views import *
urlpatterns = patterns('blog.views',
 (r'^$', archive),
 (r'foo/', foo),
 (r'bar/', bar),
)
```

В соответствии с инструкцией `import` все три функции, как и следовало ожидать, будут иметь префикс `blog.views` и в таком виде войдут в файл `mysite/blog/views.py`. Как было описано выше, объекты после импорта входят в состав сценария, как было сделано в предыдущем примере (с объектами `archive`, `foo`, `bar`). Есть ли возможность еще больше сократить объем кода и исключить даже саму инструкцию `import`?

Как было описано в предыдущем подразделе, Django поддерживает не только объекты, но и строки, поэтому оператор импорта действительно не требуется. Его вполне можно удалить и просто обозначить кавычками имена представлений, как в следующем примере:

```
from django.conf.urls.defaults import *

urlpatterns = patterns('blog.views',
 (r'^$', 'archive'),
 (r'foo/', 'foo'),
 (r'bar/', 'bar'),
)
```

Отметим также, что объявленные функции `foo()` и `bar()` на самом деле не существуют в рассматриваемом примере приложения, но реальные проекты вполне могут предусматривать применение в приложении большого количества представлений, которые должны быть заданы в файле `URLconf`. В данном случае мы стремились лишь показать некоторые способы сокращения объема кода приложения. Дополнительные сведения о том, как упростить структуру файлов `URLconf`, можно найти в документации Django по адресу <http://docs.djangoproject.com/en/dev/intro/tutorial03/#simplifying-the-urlconfs>.

Наконец, рассмотрим так называемый контроллер, т.е. функцию представления, которая вызывается после рассмотрения сопоставляемого пути URL.

### 11.9.3. Создание функции представления

В этом разделе в основном рассматриваются функции представления, на которых основаны функциональные возможности приложений данного типа. Процесс разработки может потребовать определенного времени, поэтому вначале представим готовый результат, чтобы можно было понять, к чему мы стремимся, а затем перейдем к подробному изучению процесса, что позволит получить навыки программирования этих приложений.

## Фиктивное представление Hello World

Прежде всего попытаемся непосредственно на этом раннем этапе разработки сразу же заняться отладкой шаблона HTML и файла URLconf, не создавая полное и всеобъемлющее представление. Ниже показано, как это сделать. Создадим фиктивный файл `BlogPost` и немедленно развернем его в виде шаблона. Для этого необходимо подготовить следующий файл `mysite/blog/views.py` из шести инструкций для приложения Hello World:

```
views.py
from datetime import datetime
from django.shortcuts import render_to_response
from blog.models import BlogPost

def archive(request):
 post = BlogPost(title='mocktitle', body='mockbody',
 timestamp=datetime.now())
 return render_to_response('archive.html', {'posts': [post]})
```

Напомним, что представление должно иметь имя `archive()`, поскольку так оно обозначено в URLconf, поэтому не приходится задумываться при выборе имени. В коде создается фиктивная публикация блога и передается в шаблон в качестве списка публикаций с одним элементом. (Не следует применять вызов `post.save()`, поскольку... Впрочем, предлагаем объяснить причину этого читателю.)

К рассмотрению функции `render_to_response()` вернемся немного позже, но уже сейчас можно представить себе, что в этой функции должно происходить получение шаблона (`archive.html`, находящегося в файле `mysite/blog/templates`) и контекстного словаря, их объединение, формирование кода HTML и возврат пользователю.

Вызовите на выполнение сервер разработки (или переведите его в действующий режим с помощью работающего веб-сервера). Устраните все обнаруженные ошибки в файле URLconf и шаблоне. После того как приложение начнет нормально работать, будут получены результаты, аналогичные тем, что показаны на рис. 11.14.

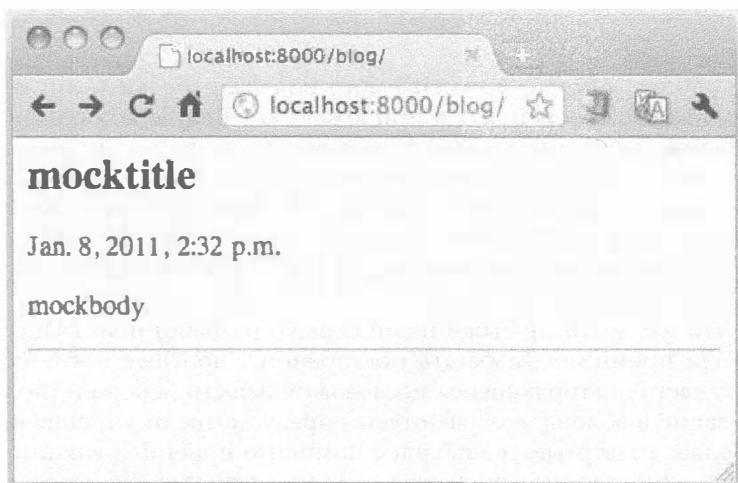


Рис. 11.14. Вывод, полученный при формировании фиктивного представления

Чтобы быстрее всего убедиться в правильности выбранного подхода и проверить основные параметры настройки, в подобных случаях действительно целесообразно создавать фиктивные представления с данными, часть которых получена в результате имитации. Таким образом можно организовать итерационный и адаптивный процессы, а после получения на всех этапах нужных результатов приступить к разработке реального приложения.

## Действительное представление

Перейдем к созданию реального продукта, простой функции представления (фактически двух), предназначенной для выборки всех публикаций блога из базы данных и отображения их для пользователей с помощью подготовленного нами шаблона. Вначале рассмотрим “формализованный” процесс разработки. Под этим подразумевается процесс, который строго следует указанным ниже этапам, от получения данных до возврата ответа HTTP назад клиенту.

Передача запроса в базу данных для получения всех записей блога.

Загрузка файла шаблона.

Создание контекстного словаря для шаблона.

Передача контекста в шаблон.

Развертывание шаблона в виде кода HTML.

Возврат кода HTML с помощью ответа HTTP.

Откройте файл `blog/views.py` и введите следующие строки кода точно так, как показано. В этом коде применяются описанные раньше усовершенствования, а приведенный выше фиктивный файл `views.py` подвергается существенным изменениям:

```
views.py
from django.http import HttpResponse
from django.template import loader, Context
from blog.models import BlogPost

def archive(request):
 posts = BlogPost.objects.all()
 t = loader.get_template("archive.html")
 c = Context({'posts': posts})
 return HttpResponse(t.render(c))
```

Проверьте работу сервера, применяемого для разработки (или действующего веб-сервера), затем снова перейдите к приложению в браузере. Должен раскрыться простой, лишенный ненужных деталей список всех введенных нами почтовых отправок блога (с действительными данными), который снабжен заголовком и отметками времени, а также содержит элементы текста публикаций, разделенные горизонтальной линией (с применением тега `<hr>`), как показано на рис. 11.15 (при условии, что создана только первая и единственная пара публикаций, как в описанном выше примере).

Очевидно, что мы достигли своей цели! Однако разработчики Django неуклонно придерживаются принципа “избегать повторений”, поэтому после обнаружения исключительно часто повторяющейся последовательности действий (получение данных, развертывание шаблона, возврат ответа) предусмотрели упрощенный процесс, который позволяет развертывать шаблон с помощью простой функции представления. Именно здесь снова применяется уже знакомая функция `render_to_response()`.



Рис. 11.15. Представление публикаций пользователя в блоге

Перед этим функция `render_to_response()` была опробована в фиктивном представлении, а на данном этапе попытаемся ее применить в реальном представлении. Добавим ее к оператору импорта из `django.shortcuts`, удалим ставшие ненужными операторы импорта из `loader`, `Context` и `HttpResponse` и заменим последние три строки представления. Полученный в итоге код должен выглядеть примерно так:

```
views.py
from django.shortcuts import render_to_response
from blog.models import BlogPost

def archive(request):
 posts = BlogPost.objects.all()
 return render_to_response('archive.html', {'posts': posts})
```

После повторной загрузки кода в браузере ничего не должно измениться, поскольку мы лишь сократили код, но не внесли никаких реальных изменений в его функции. Дополнительные сведения о функции `render_to_response()` приведены на следующих страницах официальной документации:

- <http://docs.djangoproject.com/en/dev/intro/tutorial03/#a-shortcut-render-to-response>
- <http://docs.djangoproject.com/en/dev/topics/http/shortcuts/#render-to-response>

Описанные выше способы сокращения кода — только первые шаги в этом направлении. Предусмотрены еще некоторые, специальные типы функций представления,

называемые универсальными представлениями, которые будут рассматриваться далее. Эти функции требуют еще меньше доработки, чем `render_to_response()`. Например, при использовании универсальных представлений можно даже не создавать какие-либо функции представления; достаточно лишь ввести в действие заранее подготовленное универсальное представление, предусмотренное в Django, а затем связать его непосредственно с файлом `URLconf`. В этом состоит одно из основных назначений универсальных представлений и основное их преимущество: они вообще не требуют написания какого-либо кода!

## 11.10. Усовершенствование вывода

Получен именно такой результат, который требуется! Выше были описаны три этапа создания работающего приложения, после чего был получен интерфейс, с которым могут работать пользователи (и не пришлось обращаться к роли `Admin` для организации выполнения операций `CRUD` с данными). Что на очереди? В нашем распоряжении имеется простое действующее приложение для поддержки блога. Он отвечает на клиентские запросы, извлекает информацию из базы данных и отображает все публикации для пользователя. Это замечательно, но вне всякого сомнения остается возможность внести некоторые полезные усовершенствования, чтобы функционирование приложения в большей степени напоминало соответствующие реальные аналоги.

Прежде всего, вполне оправдано такое направление доработки, как сортировка публикаций в обратном хронологическом порядке, поскольку это позволяет в первую очередь рассматривать вновь поступившие сообщения. Целесообразно также ограничить объем вывода. Если на странице удастся показать лишь немного больше 10 (или даже 5) сообщений, это говорит о том, что пользователю приходится просматривать слишком большую часть каждого сообщения. Вначале займемся сортировкой в обратной хронологической последовательности.

Чтобы решить эту задачу, достаточно передать Django соответствующее указание. Фактически можно даже выбрать один из нескольких способов передачи такого указания. Например, можно задать для модели упорядочение, применяемое по умолчанию, или ввести соответствующее указание в запрос, применяемый в коде представления. Мы прибегнем к использованию последнего варианта, поскольку он проще в реализации.

### 11.10.1. Изменение запроса

Возвратившись к одному из предыдущих этапов, отметим, что `BlogPost` — применяемый класс модели данных. Атрибут `objects` представляет собой класс `Manager` модели и включает метод `all()`, предназначенный для получения набора запросов `QuerySet`. `QuerySet` можно рассматривать как объекты, которые представляют строки данных, возвращаемые из базы данных. Это описание в определенной степени условно, ведь фактически получение строк непосредственно не происходит, поскольку в объектах `QuerySet` выполняется отложенная итерация.

В действительности доступ к базе данных не происходит до тех пор, пока не начнется вычисление `QuerySet`. Иными словами, всевозможные манипуляции с объектами `QuerySet` осуществляются вообще без обращения к данным. Сведения о том, когда происходит вычисление `QuerySet`, приведены в официальной документации по адресу <http://docs.djangoproject.com/en/dev/ref/models/queries/>.



На этом вводное описание завершается. Все изложение этой темы можно было свести к тому, что следует добавить вызов метода `order_by()` и задать параметр сортировки. В данном случае необходимо, чтобы новейшие сообщения отображались в первую очередь, а это означает, что сортировка должна проводиться в обратном порядке по отметкам времени. Для этого достаточно заменить инструкцию запроса следующей:

```
posts = BlogPost.objects.all().order by('-timestamp')
```

Указанием на то, что сортировка должна проводиться в хронологическом порядке по убыванию, служит знак “минус” (-) перед параметром timestamp. Для возврата к обычному порядку сортировки по возрастанию достаточно удалить знак “минус”.

Для проведения проверки с чтением десяти последних публикаций требуется больше записей по сравнению с теми двумя записями BlogPost, которые находятся в нашей базе данных, поэтому для нас настало время рассмотреть, как с помощью нескольких строк кода ввести в действие командный интерпретатор Django (на сей раз для несложных действий; нам пока не нужна вся мощь IPython или bpython) и автоматически сформировать ряд записей в базе данных:

```
$./manage.py shell --plain
Python 2.7.1 (r271:86882M, Nov 30 2010, 09:39:13)
[GCC 4.0.1 (Apple Inc. build 5494)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from datetime import datetime as dt
>>> from blog.models import BlogPost
>>> for i in range(10):
... bp = BlogPost(title='post #%d' % i,
... body='body of post #%d' % i, timestamp=dt.now())
... bp.save()
...
```

На рис. 11.16 показано, какие изменения произойдут в браузере после обновления содержимого базы данных.

Для проверки только что внесенных изменений, а также нового запроса, который мы собираемся использовать, можно, кроме всего прочего, применить командный интерпретатор:

```
>>> posts = BlogPost.objects.all().order_by('-timestamp')
>>> for p in posts:
... print p.timestamp.ctime(), p.title
...
Fri Dec 17 15:59:37 2010 post #9
Fri Dec 17 15:59:37 2010 post #8
Fri Dec 17 15:59:37 2010 post #7
Fri Dec 17 15:59:37 2010 post #6
Fri Dec 17 15:59:37 2010 post #5
Fri Dec 17 15:59:37 2010 post #4
Fri Dec 17 15:59:37 2010 post #3
Fri Dec 17 15:59:37 2010 post #2
Fri Dec 17 15:59:37 2010 post #1
Fri Dec 17 15:59:37 2010 post #0
Mon Dec 13 00:13:01 2010 test admin entry
Sat Dec 11 16:38:37 2010 test cmd-line entry
```



Рис. 11.16. Результат добавления десяти записей к исходной паре публикаций блога

Это позволяет достичь определенной степени уверенности в том, что после копирования основной части кода в функцию представления будут получены ожидаемые результаты.

Кроме того, вывод можно ограничить только наиболее подходящими 10 элементами с использованием удобного синтаксиса среза (`[ :10 ]`) языка Python, поэтому предусмотрим и такое изменение. Внесите эти изменения и обновите файл `blog/views.py`, чтобы он выглядел примерно таким образом:

```
views.py
from django.shortcuts import render_to_response
from blog.models import BlogPost

def archive(request):
 posts = BlogPost.objects.all().order_by('-timestamp')[:10]
 return render_to_response('archive.html', {'posts': posts})
```

Сохраните указанные исправления и снова перезагрузите страницу в браузере. Должны быть обнаружены следующие два изменения: публикации блогов отображаются в обратном хронологическом порядке и видны только первые десять публикаций. Иными словами, в связи с тем, что общее количество записей равно 12, больше не отображаются две первоначально поступившие записи, как показано на рис. 11.17.

Очевидно, что корректировки, внесенные в запрос, были довольно несложными, но важными, поскольку, например, в данном случае удобнее всего перейти на другое применяемое по умолчанию упорядочение записей в модели (позволяющее выбрать N последних, самых свежих публикаций) как более подходящее для блога.

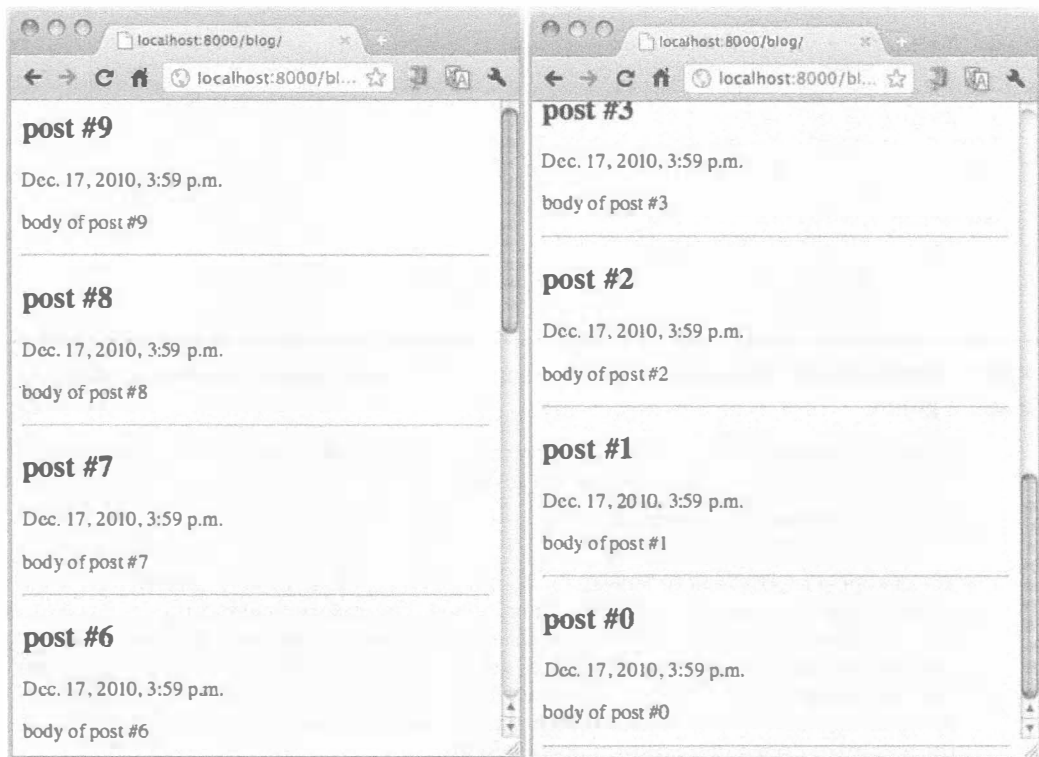


Рис. 11.17. Результаты отображения только десяти публикаций блога, полученных в последнюю очередь

## Задание упорядочения, применяемого по умолчанию в модели

После задания в модели предпочтительного упорядочения в любых других приложениях или проектах на основе Django, получающих доступ к данным в модели, будет использоваться это упорядочение. Чтобы задать применяемое по умолчанию упорядочение для модели, необходимо внести изменение во внутренний класс Meta — задать значение атрибута упорядочения в этом классе:

```
class Meta:
 ordering = ('-timestamp',)
```

Это равносильно перемещению вызова `order_by('-timestamp')` из запроса в модель. Внесите это исправление в оба файла, после чего должен быть получен следующий код:

```
models.py
from django.db import models

class BlogPost(models.Model):
 title = models.CharField(max_length=150)
 body = models.TextField()
 timestamp = models.DateTimeField()

 class Meta:
 ordering = ('-timestamp',)

views.py
from django.shortcuts import render_to_response
from blog.models import BlogPost

def archive(request):
 posts = BlogPost.objects.all()[:10]
 return render_to_response('archive.html', {'posts': posts})
```



### Уголок хакера. Уменьшение объема `archive()` до одной (длинной) строки Python

После успешного освоения конструкции `lambda` программист сможет сократить объем кода в функции `archive()` до одной строки:

```
archive = lambda req: render_to_response('archive.html',
 {'posts': BlogPost.objects.all()[:10]})
```

Такая корректировка способствует также повышению удобства кода для чтения, а это одно из преимуществ применения стиля Python. Тем самым еще раз подчеркиваются выразительные возможности Python и аналогичных ему языков, которые позволяют сокращать количество строк кода и одновременно повышать удобство чтения. Сокращение объема кода не всегда способствует тому, что сценарий становится более удобным для восприятия, поэтому в этой книге время от времени приводятся примечания, где можно ознакомиться с дополнительными пояснениями.

В первоначальную версию сценария были также внесены следующие корректировки: имя переменной `request` было сокращено до `req`, а также исключена переменная `posts` (следует учитывать, что уменьшение количества используемых переменных способствует уменьшению потребности в оперативной памяти, пусть даже не столь значительному). Программистам, не столь знакомым с языком Python, рекомендуется изучить главу “Функции” книг *Core Python Programming* или *Core Python Language Fundamentals*, в которой рассматривается конструкция `lambda`.

После перезагрузки страницы в веб-браузере не должны появиться какие-либо изменения, как и следовало ожидать. К этому времени мы фактически провели определенную работу по улучшению процедуры выборки данных из базы данных, а теперь рассмотрим возможности сведения к минимуму взаимодействия с базой данных.

## 11.11. Работа с данными, введенными пользователем

Прежде всего следует определить, все ли сделано до конца в рассматриваемом приложении. Во-первых, проверим, можно ли добавлять публикации блога с помощью командного интерпретатора или административного приложения. Во-вторых, определим, обеспечивается ли просмотр данных с помощью предусмотренного нами пользовательского интерфейса. И можем ли мы остановиться на достигнутом? Рассмотрим все последовательно.

Безусловно, любой программист не испытывает затруднений при вводе данных путем создания объектов в командном интерпретаторе, не говоря уже о работе с более удобным административным приложением, но нельзя рассчитывать на то, что все пользователи будут иметь представление о командном интерпретаторе Python и тем более о работе с этим интерпретатором. Нежелательно также вкладывать в руки пользователей мощные и специализированные средства административного приложения, которое служит для управления проектом. Итак, без создания пользовательского интерфейса нельзя обойтись.

Согласно описанию, приведенному в главе 10, включая то, что было сказано выше в данной главе, для создания пользовательского интерфейса может применяться следующий трехэтапный процесс.

Подготовка формы HTML, в которой пользователь может вводить данные.

Вставка в файл URLconf записи (URL, view).

Создание представления для обработки данных, введенных пользователем.

При выполнении этих действий будем придерживаться той же последовательности операций, как и при создании первого представления, описанного в данной книге.

### 11.11.1. Шаблон. Добавление формы HTML

Первый шаг остается довольно простым: создание формы для пользователей. Чтобы упростить разработку, на данный момент просто добавим следующий код HTML в верхней части сценария `blog/templates/archive.html` (перед кодом вывода объекта `BlogPost`); после решения рассматриваемой задачи можно (по желанию) вынести данный код в отдельный файл.

```
<!-- archive.html -->
<form action="/blog/create/" method="post">
 Title:
 <input type="text" name="title">

 Body:
 <textarea name="body" rows=3 cols=60></textarea>

 <input type="submit">
</form>
<hr>

{% for post in posts %}
...
```

Применение созданного ранее шаблона во время разработки вполне оправдано, поскольку он позволяет обработать данные, введенные пользователем, и вывести публикации блога с помощью одной и той же страницы. Иными словами, этот шаблон исключает необходимость постоянно щелкать и переходить от отдельной страницы ввода в форму к листингу с отображением данных `BlogPost`, и наоборот.

### 11.11.2. Добавление записи URLconf

Следующий шаг состоит в добавлении записи URLconf. Приведенный ранее код HTML позволяет использовать путь `/blog/create/`, поэтому осталось лишь связать его с функцией представления, предназначенной для сохранения записей в базе данных. Эту функцию нам также предстоит создать. Назовем функцию представления `create_blogpost()` и добавим соответствующий двухэлементный кортеж в переменную `urlpatterns` в файле URLconf приложения, чтобы она выглядела следующим образом:

```
urls.py
from django.conf.urls.defaults import *

urlpatterns = patterns('blog.views',
 (r'^$', 'archive'),
 (r'^create/', 'create_blogpost'),
)
```

Теперь займемся подготовкой кода сценария `create_blogpost()`.

### 11.11.3. Представление. Обработка данных, введенных пользователем

Обработка веб-форм на платформе Django во многом напоминает обработку переменных стандартного интерфейса обмена данными (common gateway interface — CGI), который рассматривался в главе 10. Наша задача состоит в разработке эквивалента соответствующего кода для платформы Django. Чтобы уточнить детали подготовки фрагментов кода для добавления в сценарий `blog/views.py`, можно еще раз просмотреть документацию Django. Прежде всего необходимо обеспечить импорт некоторых дополнительных модулей, как показано ниже.

```
from datetime import datetime
from django.http import HttpResponseRedirect
```

Функция представления должна выглядеть примерно так:

```
def create_blogpost(request):
 if request.method == 'POST':
 BlogPost(
 title=request.POST.get('title'),
 body=request.POST.get('body'),
 timestamp=datetime.now(),
).save()
 return HttpResponseRedirect('/blog/')
```

Как и при использовании функции представления `archive()`, передача запроса осуществляется автоматически. Данные, введенные в форме, передаются по методу POST, поэтому первым делом необходимо проверить поступление данных. Затем создается новая запись `BlogPost` с данными формы, которые обозначены текущим временем в виде временной отметки, после чего с помощью метода `save()` данные сохраняются в базе данных. Вслед за этим происходит обратный переход по пути `/blog` для проверки последней публикации (и вывода очередной пустой формы для подготовки следующей публикации блога).

Еще раз проверим работу сервера, применяемого для разработки, или реального веб-сервера и откроем страницу, подготовленную с помощью приложения. Над листингом данных откроется форма (рис. 11.18), с помощью которой можно выполнить проверку нового средства.

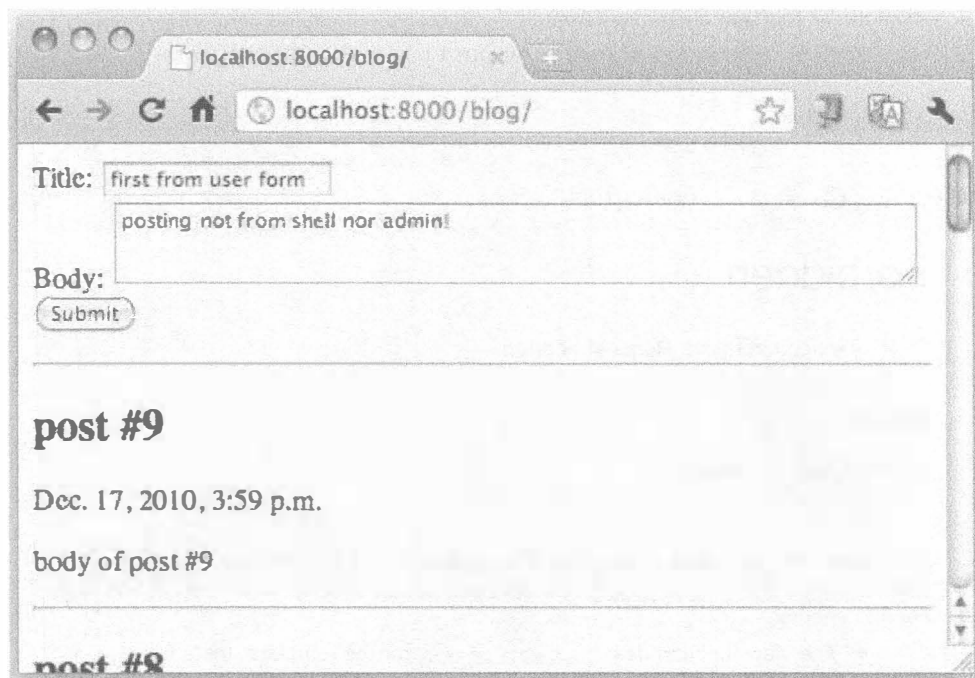


Рис. 11.18. Наша первая пользовательская форма (за которой следуют предыдущие публикации)

#### 11.11.4. Проверка возможности передачи данных с одного сайта на другой

Рассмотрим все последовательно. На том этапе разработки приложения, на котором удалось добиться разворачивания и последующей передачи формы, обнаруживается, что браузер пытается получить доступ к URL `/blog/create/`, но останавливается из-за ошибки, показанной на рис. 11.19.

На платформе Django предусмотрено средство защиты данных, которое устанавливает запрет на обработку данных, передаваемых по методу POST, не позволяющих предотвратить атаки по принципу подделки запросов, передаваемых с одного узла на другой (cross-site request forgery — CSRF). Описание атаки CSRF выходит за рамки книги, но с дополнительными сведениями можно ознакомиться с помощью следующих источников:

- <http://docs.djangoproject.com/en/dev/intro/tutorial04/#write-a-simple-form>
- <http://docs.djangoproject.com/en/dev/ref/contrib/csrf/>

Рассматриваемое нами приложение является несложным, поэтому для решения указанной задачи достаточно добавить в двух местах небольшие фрагменты кода к тем, что уже имеются.

1. Добавление токена CSRF (`{% csrf_token %}`) к формам, отправляемым с применением метода POST на сайт.
2. Отправка экземпляра контекста запроса в токен через шаблон.

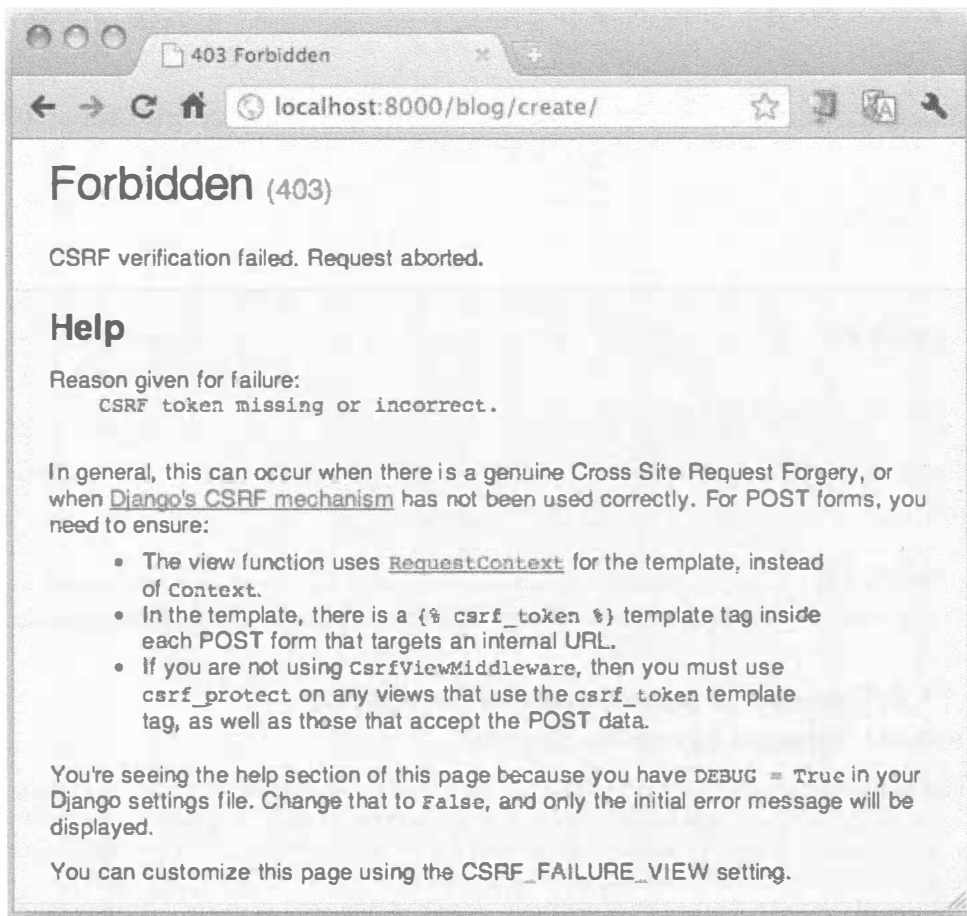


Рис. 11.19. Экран CSRF с сообщением об ошибке

Термин “контекст запроса” расширяется буквально как словарь, который содержит информацию о запросе. Изучение документации по CSRF в приведенных выше источниках показывает, что информация `django.template.Request Context` всегда обрабатывается с учетом применения встроенной защиты CSRF.

Первый этап выполняется путем добавления токена к форме. Отредактируйте строку заголовка `<FORM>` в файле `mysite/blog/templates/archive.html`, добавив токен CSRF в форму, чтобы она выглядела следующим образом:

```
<form action="/blog/create/" method=post>{% csrf_token %}
```



Второй этап требует редактирования сценария `mysite/blog/views.py`. Откорректируйте строку `return` в функции представления `archive()`, добавив экземпляр `RequestContext`, как показано ниже.

```
return render_to_response('archive.html', {'posts': posts},
 RequestContext(request))
```

Следует обязательно предусмотреть импорт `django.template.RequestContext`:

```
from django.template import RequestContext
```

После сохранения указанных изменений появится возможность передавать данные в приложение из формы (без применения административного приложения или командного интерпретатора). Ошибки CSRF больше не будут появляться, и передача записей `BlogPost` станет осуществляться успешно.

## 11.12. Простые и модельные формы

В предыдущем разделе было показано, как работать с данными, введенными пользователем, для чего была проведена доработка формы HTML. Теперь рассмотрим, какие возможности предоставляет платформа Django для упрощения средств получения данных от пользователя (с помощью форм Django), в особенности форм, состоящих именно из тех полей, которые составляют модель данных (формы модели Django).

### 11.12.1. Вводные сведения о формах Django

Если не учитывать одноразовую дополнительную работу, требуемую для создания защиты от атак CSRF, то три описанных ранее шага, необходимых для построения простой формы ввода, безусловно, выглядят слишком трудоемкими и тривиальными. Ведь это же, в конце концов, платформа Django, созданная в строгом соответствии с принципом DRY.

По-видимому, наибольших трудозатрат потребуют те части будущего приложения, при разработке которых необходимо проверить модель данных, применяемую повсеместно в этом приложении. В самой форме заслуживают внимания имя и заголовок:

```
Title: <input type=text name=title>

Body: <textarea name=body rows=3 cols=60></textarea>

```

В основном аналогичные данные применяются и в представлении `create_blogpost()`:

```
BlogPost(
 title=request.POST.get('title'),
 body=request.POST.get('body'),
 timestamp=datetime.now(),
).save()
```

Следует учитывать, что после определения модели данных она должна оставаться единственным местом, где можно видеть заголовок, текст и, допустим, отметку времени (хотя последнее значение представляет собой частный случай, поскольку его не

обязан вводить пользователь). Не будет ли резонным рассчитывать на то, что поля формы будут подготовлены средствами веб-платформы исключительно на основе модели данных? В связи с чем разработчик должен не только создавать модель данных, но и подготавливать поля формы? При решении этой задачи действительно используются программные средства в виде форм Django.

Прежде всего создадим форму Django для своих входных данных:

```
from django import forms

class BlogPostForm(forms.Form):
 title = forms.CharField(max_length=150)
 body = forms.CharField(widget=forms.Textarea)
 timestamp = forms.DateTimeField()
```

На этом наши обязанности не заканчиваются. В разработанной нами форме HTML задан элемент `textarea` языка HTML, состоящий из трех строк и имеющий ширину шестьдесят символов. Вместо разработки кода HTML вручную мы применили программные средства, формирующие этот код автоматически, но должны предусмотреть возможность задавать требования к элементам формы. В данном случае решение состоит в том, что передача атрибутов должна происходить непосредственно в форму:

```
body = forms.CharField(
 widget=forms.Textarea(attrs={'rows':3, 'cols':60})
)
```

## 11.12.2. Вариант с применением форм модели

После небольшого отступления, касающегося задания атрибутов, перейдем к внимательному рассмотрению определения `BlogPostForm`. Хотелось бы знать, насколько тривиальной эта задача была для читателя. Как показано в следующем коде, указанное определение выглядит почти идентичным модели данных:

```
class BlogPost(models.Model):
 title = models.CharField(max_length=150)
 body = models.TextField()
 timestamp = models.DateTimeField()
```

Если читатель подтвердил, что для него задача тривиальна, то был прав: два рассматриваемых компонента похожи, как близнецы. Таким образом, с точки зрения любого уважающего себя автора сценариев Django количество повторов слишком велико. Проведенная нами ранее работа по созданию отдельного объекта `Form` была бы вполне оправдана, если бы задача заключалась в подготовке формы для веб-страницы с нуля, без лежащей в ее основе модели данных.

Если поля формы полностью соответствуют модели данных, то нам не следует заниматься созданием объекта `Form`. Вместо этого гораздо более продуктивные результаты были бы достигнуты с помощью объекта `ModelForm` платформы Django, как показано в следующем коде:

```
class BlogPostForm(forms.ModelForm):
 class Meta:
 model = BlogPost
```

Это много лучше; мы действительно добились требуемой экономии усилий. После перехода от использования `Form` к `ModelForm` появляется возможность задать класс `Meta`, который определяет, на какой модели данных должна быть основана форма. Теперь форма `HTML` после ее создания будет иметь поля для всех атрибутов модели данных.

В рассматриваемом случае мы не можем рассчитывать на то, что пользователь каждый раз будет правильно вводить отметку времени, поэтому должны предусмотреть, чтобы в приложении этот элемент содержимого формировался программным путем для каждой передаваемой записи блога. Это не сложно, поскольку достаточно лишь добавить еще один атрибут с именем `exclude`, который служит для удаления элементов формы из создаваемого кода `HTML`. Добавим инструкцию импорта, а также полный класс `BlogPostForm`, представленный в следующем примере, в нижней части файла `blog/models.py`, согласно разработанному определению `BlogPost`:

```
blog/models.py
from django.db import models
from django import forms

class BlogPost(models.Model):
 ...

class BlogPostForm(forms.ModelForm):
 class Meta:
 model = BlogPost
 exclude = ('timestamp',)
```

### 11.12.3. Использование объекта класса `ModelForm` для создания формы `HTML`

Прежде всего рассмотрим, в чем преимущества этого объекта. Итак, с первого взгляда очевидно, что этот объект позволяет избавиться от необходимости подробно определения полей в форме. Таким образом, можно заменить код в верхней части файла `mysite/blog/templates/archive.html` следующим кодом:

```
<form action="/blog/create/" method=post>{% csrf_token %}
 <table>{{ form }}</table>

 <input type=submit>
</form>
```

Разумеется, кнопку передачи формы исключать нельзя. Кроме того, можно видеть, что по умолчанию форма повторяет структуру таблицы. Проведем тестирование формы. Для этого достаточно открыть командный интерпретатор Django, создать объект `BlogPostForm`, а затем произвести с ним некоторые манипуляции. Выполняемые действия сводятся к простым операциям:

```
>>> from blog.models import BlogPostForm
>>> form = BlogPostForm()
>>> form
<blog.models.BlogPostForm object at 0x12d32d0>
>>> str(form)
'<tr><th><label for="id_title">Title:</label></th><td><input id="id_title" type="text"
name="title" maxlength="150" /></td></tr>\n<tr><th><label for="id_body">Body:</label></
th><td><textarea id="id_body" rows="10" cols="40" name="body"></textarea></td></tr>'
```

Таким образом, весь необходимый код HTML сформирован автоматически, и нам не пришлось ничего разрабатывать. (Еще раз отметим, что в результате применения атрибута `exclude` отметка времени исключена из формы. Ради интереса можно на время законментировать инструкцию с этим атрибутом, чтобы убедиться в том, что в подготовленном коде HTML появилось дополнительное поле с отметкой времени.)

Если нужно будет, чтобы вывод отличался от таблицы HTML, состоящей из строк и ячеек, можно запросить формирование соответствующего вывода с использованием методов `as_*()`: `{{ form.as_p }}` для формы с разграничителями текста `<p>...</p>`, `{{ form.as_ul }}` для маркированного списка с элементами `<li>` и т.д.

Файл `URLconf` остается тем же, что и прежде, поэтому осталось внести лишь одно последнее изменение — обновить функцию представления, чтобы объект `ModelForm` отправлялся в шаблон. Для этого необходимо создать экземпляр объекта и передать его в виде дополнительной пары “ключ — значение” контекстного словаря. Таким образом, заменим последнюю строку функции `archive()` в файле `blog/views.py` следующим образом:

```
return render_to_response('archive.html', {'posts': posts,
 'form': BlogPostForm()}, RequestContext(request))
```

Не забудьте добавить инструкции импорта для моделей данных и формы в верхней части `views.py`:

```
from blog.models import BlogPost, BlogPostForm
```

## 11.12.4. Обработка данных класса `ModelForm`

Изменения, внесенные непосредственно перед этим, были направлены на создание объекта `ModelForm` и подготовку с его помощью кода HTML, который должен быть развернут в браузере пользователя. Рассмотрим, что происходит после отправки пользователем введенной им информации. Очевидно, что все еще обнаруживаются повторы кода в представлении `create_blogpost()`, которое, как известно, также определено в файле `blog/views.py`. По аналогии с тем, как задан класс `Meta` для объекта `BlogPostForm`, указывающий, что поля этого объекта должны быть взяты из объекта `BlogPost`, мы должны отказаться от создания собственного объекта, подобного ему, в методе `create_blogpost()`:

```
def create_blogpost(request):
 if request.method == 'POST':
 BlogPost(
 title=request.POST.get('title'),
 body=request.POST.get('body'),
 timestamp=datetime.now(),
).save()
 return HttpResponseRedirect('/blog/')

```

Задавать такие поля, как заголовок, текст и т.д., нет необходимости, поскольку их определение дано в модели данных. Хотелось бы иметь возможность сократить это представление следующим образом:

```
def create_blogpost(request):
 if request.method == 'POST':
 form = BlogPostForm(request.POST)
```

```
if form.is_valid():
 form.save()
return HttpResponseRedirect('/blog/')
```

К сожалению, этого нельзя сделать из-за наличия поля с отметкой времени. В рассматриваемом ранее варианте создания формы HTML мы были вынуждены предусмотреть исключение и должны применить аналогичную конструкцию здесь. Конструкция `if`, предназначенная для этой цели, выглядит так:

```
if form.is_valid():
 post = form.save(commit=False)
 post.timestamp=datetime.now()
 post.save()
```

Вполне очевидно, что нам пришлось добавить отметку времени к данным, а затем вручную сохранить объект для получения желаемого результата. Следует отметить, что это форма `save()`, а не модель `save()`, которая возвращает экземпляр модели `Blog`, но в связи с тем, что применяется значение `commit=False`, запись данных в базу данных не происходит до вызова `post.save()`. После внесения указанных изменений можно приступить к эксплуатации формы обычным образом, как показано на рис. 11.20.

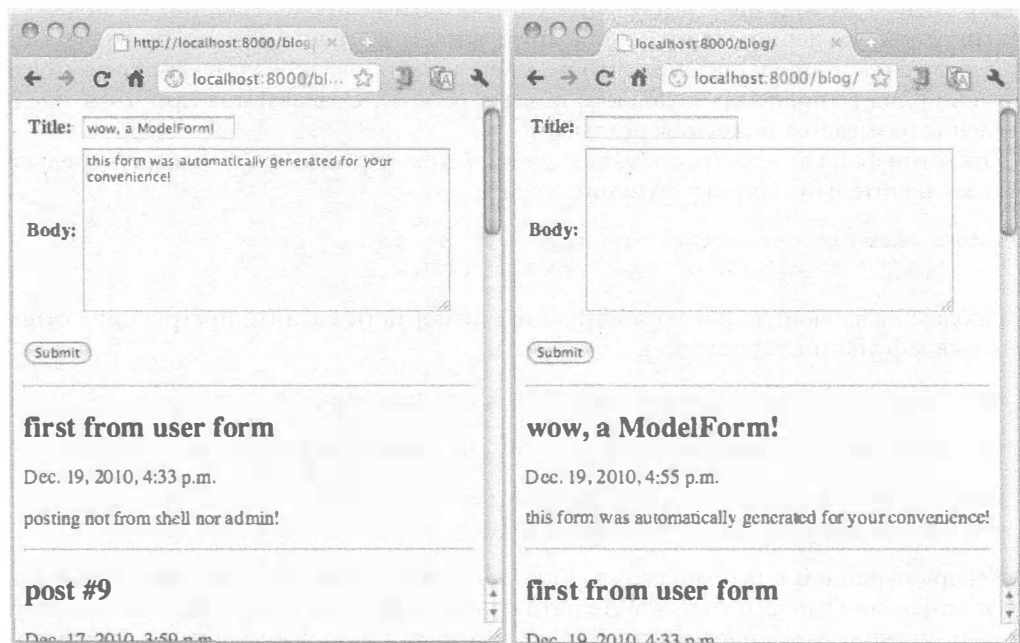


Рис. 11.20. Автоматически подготовленная пользовательская форма

## 11.13. Дополнительные сведения о представлениях

Последней среди наиболее важных тем, которые мы должны обсудить, является следующее описание, обязательное для включения в любую книгу по Django: универсальные представления. До сих пор мы решали задачу создания контроллера или средств обработки данных для приложения с помощью представления,

определяемого пользователем. В связи с тем, что платформа Django создана на основе принципа DRY, разработчику на этой платформе постоянно приходится использовать упрощенные методы, такие как `render_to_response()`.

Универсальные представления, позволяющие подняться на более высокий уровень абстракции, являются не только мощными, но и простыми в использовании, поэтому после ввода их в действие полностью отпадает необходимость в подготовке представлений. Достаточно вызвать общие представления непосредственно из файла `URLconf`, передать несколько фрагментов требуемых данных, после чего применить программу `views.py`, не изменяя и не добавляя в ней какой-либо код. Тем не менее перед ознакомлением с этими средствами необходима дополнительная подготовка. Прежде всего возвратимся к краткому описанию средств предотвращения атак CSRF, но без рассмотрения примеров кода. Вначале рассмотрим несколько вступительных разделов.

### 11.13.1. Полууниверсальные представления

Противодействие атакам CSRF приходится предусматривать в любом приложении, которое получает ответы на отправленные им запросы, поэтому возникает необходимость решать весьма трудоемкую задачу — передачу экземпляра контекста запроса. Эта задача не только трудоемкая, но и сложная, особенно для начинающих. С ее решения можно начать изучение универсальных представлений, но пока без фактического их использования. Для этого внесем корректировку в применяемое пользовательское представление, чтобы в нем использовалось универсальное представление для выполнения основного объема работы. Создаваемое при этом представление называется полууниверсальным.

Откройте файл `mysite/blog/views.py` в текстовом редакторе и замените следующую заключительную строку функции `archive()`:

```
return render_to_response('archive.html', {'posts': posts,
 'form': BlogPostForm(), RequestContext(request))
```

Добавьте следующую новую инструкцию импорта (и удалите инструкцию, относящуюся к функции `render_to_response()`):

```
from django.views.generic.simple import direct_to_template
```

Измените заключительную строку, чтобы она выглядела следующим образом:

```
return direct_to_template(request, 'archive.html',
 {'posts': posts, 'form': BlogPostForm()})
```

Теперь перейдем к рассмотрению того, для чего это все было сделано. Очевидно, что платформа Django позволяет сократить объем работы, выполняемый программистом, поскольку количество строк кода, который он должен написать, уменьшается, но до сих пор мы исключили лишь код, относящийся к экземпляру контекста запроса. Позволяет ли это получить какие-либо иные преимущества? Еще нет. Мы находимся лишь в начале пути. В данном примере `direct_to_template()` фактически не используется как универсальное представление, поэтому в действительности на данный момент лишь преобразовано пользовательское представление в полууниверсальное представление с учетом его назначения.

Следует также отметить, что чисто универсальное представление применяется по принципу его непосредственного вызова из файла `URLconf`, поэтому не требует написания кода, который должен быть включен в файл `view.py`. Универсальные

представления — это представления, многократно используемые повторно, которые довольно просты, но, несмотря на это, нежелательно создавать их или воссоздавать каждый раз, когда потребуются одни и те же реализуемые ими функциональные средства. В качестве примеров таких функциональных средств можно назвать перенаправление пользователей на статические страницы, предоставление универсальных возможностей вывода для объектов и т.д.

## Практическое использование универсального представления

В предыдущем подразделе уже фактически использовалась функция универсального представления, но в действительности это не был пример применения универсального представления в чистом виде. Поэтому теперь рассмотрим, как это действие применяется на практике. Откройте файл `URLconf` своего проекта (`mysite/urls.py`). Напомним, что в подразделе “Проверка работы с приложением `admin`”, приведенном выше в этой главе, было показано, что при переходе по адресу `http://localhost:8000/` возникала ошибка 404.

В качестве объяснения было указано, что платформа Django может обрабатывать только такие пути, для которых имеется согласующееся регулярное выражение. Итак, `'/'` не согласуется ни с  `'/blog/'`, ни с  `/admin/`, поэтому мы вынуждали пользователей посещать только страницы, обозначенные этими ссылками, чтобы получить доступ к приложению. Такая организация работы становится источником ненужных затруднений, поэтому пользователям следует предоставить более удобный способ работать на сайте, позволяя им посетить путь `'/'` верхнего уровня, а затем обеспечить автоматическое перенаправление по пути  `'/blog/'` непосредственно с помощью приложения.

Таким образом, создается идеальная возможность воспользоваться универсальным представлением `redirect_to()` в правильно настроенной среде. Для этого достаточно добавить единственную строку в файл `urlpatterns`, как показано в следующем коде:

```
urlpatterns = patterns('',
 (r'^$', 'django.views.generic.simple.redirect_to',
 {'url': '/blog/'}),
 (r'^blog/', include('blog.urls')),
 (r'^admin/', include(admin.site.urls)),
)
```

Разумеется, эту строку, возможно, придется разбить на две, но все равно они будут составлять одну инструкцию. Кроме того, дополнительные операции импорта здесь не требуются, поскольку применяется строковое значение, а не объект. Теперь, после перехода пользователей по пути `'/'` происходит их перенаправление по пути  `'/blog/'`, а в этом и состоит рассматриваемая задача. В файл `view.py` не потребовалось вносить какие-либо изменения, и вся задача свелась к вызову универсального представления из файла `URLconf` (проекта или приложения). Это один из примеров успешного применения универсального представления! (Возможно, читатель захочет ознакомиться с более сложным примером, и это стремление понятно, поэтому в конце главы приведено упражнение с более сложным универсальным представлением, которое позволяет приобщиться ко всем возможностям этих объектов.)

До сих пор мы рассматривали универсальные представления `direct_to_template()` и `redirect_to()`, но на практике довольно часто приходится сталкиваться и с другими универсальными представлениями. К ним относятся

`object_list()` и `object_detail()`, а также универсальные представления, позволяющие работать с датами и временем, такие как `archive_{day, week, month, year, to day, index}()`. Наконец, имеются универсальные представления CRUD, в частности `{create, update, delete}_object()`.

Кроме того, автор обязан сообщить о такой тенденции, как переход к универсальным представлениям на основе классов. Это новое средство, введенное в версии Django 1.3. Какими бы мощными ни были универсальные представления, их возможности намного увеличиваются после перехода к универсальным представлениям на основе класса. (Это можно сравнить с тем, как возросли возможности исключений после перехода от исключений в виде простых строк к исключениям на основе классов в версии Python 1.5.)

Дополнительные сведения о простых и универсальных представлениях, а также об универсальных представлениях на основе классов см. в официальной документации по адресу <http://docs.djangoproject.com/en/dev/topics/generic-views/> и <http://docs.djangoproject.com/en/dev/topics/class-based-views>.

Последние подразделы данной главы не столь важны, но содержат полезную информацию, которую читателю потребует освежить в памяти в ходе дальнейшего изучения данной книги. Те, кто желает ускорить свое продвижение в данном направлении, могут попытаться сразу же приступить к созданию приложения Django промежуточного уровня или перейти непосредственно к главе 12.

## 11.14. \*Усовершенствования внешнего интерфейса

Теперь рассмотрим, что можно сделать для улучшения организации работы приложения и как добиться единообразного внешнего вида сайта.

Создание файла каскадных таблиц стилей (Cascading Style Sheet — CSS).

### 1. Создание основного шаблона и использование наследования шаблонов.

Тематика таблиц CSS довольно проста, поэтому не будем углубляться в ее рассмотрение в этой главе, но разберем действительно небольшой пример наследования шаблонов:

```
<!-- base.html -->
Generic welcome to your web page [Login - Help - FAQ]
<h1>Blog Central</h1>
{% block content %}
{% endblock %}
© 2011 your company [About - Contact]
</body>
</html>
```

На первый взгляд кажется, что наследование шаблонов не представляет особого интереса, но фактически позволяет добиться хороших результатов. Для этого достаточно разместить такой общий материал заголовка, как логотип корпорации, ссылки для входа и выхода из системы и другие необходимые ссылки в верхней части страницы. В нижней части страницы предусматриваются такие общие сведения, как объявление об авторском праве, другие дополнительные ссылки и т.д. Важнее всего то, что в середине страницы должен быть размещен тег `{% block ... %}`. Тем самым определяется именованная область, находящаяся под управлением субшаблонов.



Чтобы воспользоваться этим новым основным шаблоном, необходимо расширить его и определить блок, перетаскиваемый в основной шаблон. Например, если требуется с помощью данного шаблона создать страницу приложения блога, предназначенную для работы с пользователем, то достаточно добавить соответствующий стандартный текст и включить страницу в приложение. Чтобы не происходила путаница с файлом `archive.html`, присвоим файлу шаблона универсальное имя `index.html`:

```
<!-- index.html -->
{% extends "base.html" %}
{% block content %}
 {% for post in posts %}
 <h2>{{ post.title }}</h2>
 <p>{{ post.timestamp }}</p>
 <p>{{ post.body }}</p>
 <hr>
 {% endfor %}
{% endblock %}
```

Тер `{% extends ... %}` служит для платформы Django указанием, что должен быть найден шаблон с именем `base.html` и в этот шаблон вставлено содержимое всех именованных блоков вместо соответствующих блоков в шаблоне. Если читатель пожелает применить наследование шаблонов, то ему необходимо внести изменение в применяемое представление, чтобы в качестве файла шаблона использовался `index.html` вместо первоначального файла `archive.html`.

## 11.15. \*Проверка компонентов

После этого необходимо провести тестирование. Это настолько важная задача, что разработчики должны выполнять ее без всякого напоминания. Можно забыть о том, что вы голодны, нуждаетесь в отдыхе или хотите спать, но нельзя забывать о тестировании каждого создаваемого приложения. Как и при решении многих поддерживаемых ею задач программирования, платформа Django позволяет проводить тестирование с использованием компонента модульного тестирования, дополняющего стандартные возможности Python, который соответствует применяемой версии Python. Django позволяет также проводить тестирование с помощью проверочных строк документирования (которые сокращенно называют также строками документации). Не удивительно, что проводимые с их помощью тесты называют документальными проверками. Подробные сведения о них приведены на страницах документации Django, поэтому в данной главе они не рассматриваются. Более важными являются модульные тесты.

Задача создания модульных тестов может оказаться несложной. Разработчики платформы Django сделали многое, чтобы помочь программистам решать задачи тестирования. В частности, эта платформа для каждого создаваемого приложения автоматически формирует файл `tests.py`. Замените файл `mysite/blog/tests.py` содержимым, приведенным в примере 11.1.

### Пример 11.1. Сценарий модульного тестирования приложения `blog(tests.py)`

```
1 # tests.py
2 from datetime import datetime
3 from django.test import TestCase
```

```
4 from django.test.client import Client
5 from blog.models import BlogPost
6
7 class BlogPostTest(TestCase):
8 def test_obj_create(self):
9 BlogPost.objects.create(title='raw title',
10 body='raw body', timestamp=datetime.now())
11 self.assertEqual(1, BlogPost.objects.count())
12 self.assertEqual('raw title',
13 BlogPost.objects.get(id=1).title)
14
15 def test_home(self):
16 response = self.client.get('/blog/')
17 self.failUnlessEqual(response.status_code, 200)
18
19 def test_slash(self):
20 response = self.client.get('/')
21 self.assertIn(response.status_code, (301, 302))
22
23 def test_empty_create(self):
24 response = self.client.get('/blog/create/')
25 self.assertIn(response.status_code, (301, 302))
26
27 def test_post_create(self):
28 response = self.client.post('/blog/create/', {
29 'title': 'post title',
30 'body': 'post body',
31 })
32 self.assertIn(response.status_code, (301, 302))
33 self.assertEqual(1, BlogPost.objects.count())
34 self.assertEqual('post title',
35 BlogPost.objects.get(id=1).title)
```

---

## Построчное объяснение

### Строки 1–5

В этих инструкциях осуществляется импорт `datetime` для отправки отметок времени, основного тестового класса `django.test.TestCase`, тестового веб-клиента `django.test.client.Client` и наконец класса `BlogPost`.

### Строки 8–13

Какие-либо требования к именованию методов тестирования не предъявляются, не считая того, что они должны начинаться с `test_`. Метод `test_obj_create()` обеспечивает лишь проверку, позволяющую убедиться в том, что объект создан успешно, и подтверждает правильность заголовка. Метод `assertEqual()` проверяет равенство обоих параметров; в противном случае завершает тест аварийно. В данном случае проверяется количество объектов, а также производится проверка введенных данных. Это очень простой и незамысловатый тест, но он позволяет многое сделать для его усовершенствования. Можно также предусмотреть возможность тестирования `ModelForm`.

## Строки 15–21

Следующая пара методов тестирования предназначена для проверки пользовательского интерфейса; в них, в отличие от первого метода, в котором проверяется лишь создание объектов, применяются вызовы веб-компонентов. Метод `test_home()` вызывает главную страницу приложения, начиная с пути `'/blog/'`, и позволяет убедиться в том, что будет получен код ошибки HTTP 200; метод `test_slash()` выполняет практически то же самое, но дает также возможность проверить, что заданное в файле `URLconf` перенаправление, в котором используется универсальное представление `redirect_to()`, действительно работает. Здесь применяется немного другое подтверждение, поскольку предполагается получение такого кода ответа на перенаправление, как 301 или 302. Фактически в данном случае ожидается получение кода 301, но тестирование не заканчивается аварийно в случае возврата кода 302, который показывает работу метода тестирования `assertIn()`, а также позволяет повторно использовать подтверждение для двух заключительных методов тестирования; оба эти метода должны возвращать код ответа 302. У читателя может возникнуть вопрос, как формируется значение `self.client` в строках 16 и 20. При создании подкласса класса `django.test.TestCase` автоматически происходит получение экземпляра клиента теста Django, которым программист вправе воспользоваться, ссылаясь на него в форме `self.client`.

## Строки 23–35

Два последних метода тестируют представление для `'/blog/create/'`, `create_blogpost()`. Первый метод, `test_empty_create()`, применяется для тестирования в такой ситуации, что ошибочно применяется запрос GET без данных. Код должен пропускать этот запрос и осуществлять перенаправление по пути `'/blog/'`. Второй метод, `test_post_create()`, применяется для моделирования истинного запроса пользователя, при котором происходит отправка действительных данных по методу POST, создается запись и пользователь перенаправляется по пути `'/blog/'`. Предусмотрена проверка всех трех требований: перенаправление 302, добавление новой публикации и проверка допустимости данных.

Итак, приступим к тестированию. Для этого необходимо вызвать на выполнение следующую команду и ознакомиться с полученным выводом:

```
$ manage.py test
Creating test database 'default'...

Ran 288 tests in 7.061s

OK
Destroying test database 'default'...
```

По умолчанию система создает исключительно для тестирования отдельную базу данных в оперативной памяти (называемую `default`). Это предусмотрено для того, чтобы исключить любую возможность повреждения производственных данных. Каждая точка (.) указывает на прохождение одной из проверок. Проверки, окончившиеся неудачей, сокращенно обозначаются буквами E (error — ошибка) и F (failure — сбой). Дополнительные сведения о тестировании с помощью платформы Django см. в документации по адресу <http://docs.djangoproject.com/en/dev/topics/testing>.

### 11.15.1. Описание кода приложения блога

Рассмотрим одновременно все заключительные версии кода данного приложения (наряду с файлами `__init__.py` (пустой файл) и `tests.py` (см. пример 11.1)). Здесь оставлены комментарии, а с веб-сайта этой книги можно также загрузить версию, содержащую только сам код, и дополнительно документированную версию.

Прежде всего рассмотрим файл уровня проекта `URLconf`, `mysite/urls.py`, который представлен в примере 11.2, хотя формально он не входит в состав нашего приложения блога.

#### Пример 11.2. Файл `URLconf` проекта `mysite (urls.py)`

```
1 # urls.py
2 from django.conf.urls.defaults import *
3 from django.contrib import admin
4 admin.autodiscover()
5
6 urlpatterns = patterns('',
7 (r'^$', 'django.views.generic.simple.redirect_to',
8 {'url': '/blog/'}),
9 (r'^blog/', include('blog.urls')),
10 (r'^admin/', include(admin.site.urls)),
11)
```

### Построчное объяснение

#### Строки 1–4

В строках установки осуществляется импорт модулей, необходимых для файла `URLconf` проекта, а также содержится некоторый код, обеспечивающий администрирование. Администрирование требуется не во всех вариантах эксплуатации приложения, поэтому можно удалить вторую и третью строки, если они не используются.

#### Строки 6–11

В разделе `urlpatterns` определены действия и директивы для универсальных представлений, а также для тех или иных приложений в составе проекта. Первый шаблон относится к пути `'/'`, который обеспечивает перенаправление к обработчику для пути `'/blog/'` с использованием универсального представления `redirect_to()`; второй шаблон, который относится к пути `'/blog/'`, позволяет передавать все запросы в файл `URLconf` приложения блога (который следует ниже); последний предназначен для административных запросов.

Следующий файл, `mysite/blog/urls.py`, представляет собой `URLconf` приложения и рассматривается в примере 11.3.

#### Пример 11.3. Файл `URLconf` приложения `blog (urls.py)`

Файл `URLconf` приложения `blog`. На этом участке кода должна производиться обработка URL путем вызова функций представления (или методов `class`).

```
1 # urls.py
2 from django.conf.urls.defaults import *
```

```

3
4 urlpatterns = patterns('blog.views',
5 (r'^$', 'archive'),
6 (r'^create/', 'create_blogpost'),
7)

```

## Построчное объяснение

### Строки 4–7

Ядром файла `urls.py` является определение отображений URL (`urlpatterns`). Работа с пользователями, которые посещают путь  `'/blog/'`, происходит с помощью метода `blog.views.archive()`. Напомним, что в файле `URLconf` проекта предусмотрено удаление пути  `'/blog/'`, поэтому ко времени перехода в данный участок кода путь URL состоит только из  `'/'`. Вызов, направленный по пути  `'/blog/create/'`, должен исходить только из запросов по методу POST к форме и ее данным; этот запрос обрабатывается функцией представления `blog.views.create_blogpost()`.

В примере 11.4 рассматривается модель данных для приложения блога, `mysite/blog/models.py`. Кроме того, в этом коде содержится также класс формы.

#### Пример 11.4. Файл данных приложения и моделей формы `blog(models.py)`

Модели данных размещаются в этом коде, но последнюю группу можно было бы выделить и разместить в своем файле.

```

1 # models.py
2 from django.db import models
3 from django import forms
4
5 class BlogPost(models.Model):
6 title = models.CharField(max_length=150)
7 body = models.TextField()
8 timestamp = models.DateTimeField()
9 class Meta:
10 ordering = ('-timestamp',)
11
12 class BlogPostForm(forms.ModelForm):
13 class Meta:
14 model = BlogPost
15 exclude = ('timestamp',)

```

## Построчное объяснение

### Строки 1–3

Осуществляется импорт классов, необходимых для задания моделей и форм. В это простое приложение включены оба класса. Если бы количество моделей и (или) форм было больше, то имело бы смысл расположить формы в отдельном файле `forms.py`.

## Строки 5–10

Таково определение нашей модели, `BlogPost`. Она включает все атрибуты данных, а также запросы, с помощью которых сортируются в обратном порядке объекты, полученные из базы данных, в соответствии с полем с отметкой времени каждой строки (с помощью внутреннего класса `Meta`).

## Строки 12–15

Здесь создается объект `BlogPostForm`, версия формы модели данных. Атрибут `Meta.model` указывает модель данных, на которой он должен быть основан, а переменная `Meta.exclude` указывает, что поле данных отсутствует в автоматически сформированных формах. Предполагается, что разработчик заполнит это поле (в случае необходимости) перед сохранением экземпляра `BlogPost` в базе данных.

Файл `mysite/blog/admin.py` в примере 11.5 используется, только если предусмотрено применение средств администрирования для приложения. Этот файл содержит классы, которые регистрируются для использования в средствах администрирования, а также все прочие специализированные классы администрирования.

### Пример 11.5. Файл конфигурации средств администрирования для приложения `blog` (`admin.py`)

```
1 # admin.py
2 from django.contrib import admin
3 from blog import models
4
5 class BlogPostAdmin(admin.ModelAdmin):
6 list_display = ('title', 'timestamp')
7
8 admin.site.register(models.BlogPost, BlogPostAdmin)
```

## Построчное объяснение

### Строки 5–8

Атрибут `list_display` класса `BlogPostAdmin` служит исключительно для дополнительных средств администрирования Django и позволяет задать административное указание, согласно которому поля должны отображаться на консоли администрирования так, чтобы пользователи могли видеть различия между разными записями данных. Предусмотрено также много других атрибутов, которые мы не имеем возможности рассмотреть. Однако рекомендуется ознакомиться с документацией по адресу <http://docs.djangoproject.com/en/dev/ref/contrib/admin/#modeladmin-options>. Если эти обозначения не применяются, то отображаются только общие имена объектов для каждой строки, поэтому почти невозможно отличить один экземпляр от другого. Последнее предусмотренное действие (выполняемое с помощью инструкции в строке 8) состоит в том, что модели и модели администрирования регистрируются в административном приложении.

В примере 11.6 представлена основная часть приложения, для которой выделен файл `mysite/blog/views.py`. Именно здесь находятся все применяемые представления; этот код эквивалентен коду контроллера для большинства веб-приложений.

Любопытно отметить, что неизменное следование принципу DRY на платформе Django и широкое применение мощных универсальных представлений становятся причиной появления пустого файла представлений. (Тем не менее некоторые разработчики считают, что из-за этого слишком большая часть кода становится скрытой, поэтому исходный код сложнее читать и понимать.) Остается лишь надеяться на то, что все специализированные или полууниверсальные представления, приведенные в этом файле, содержат лишь небольшой объем кода, являются легкими для чтения, обеспечивают максимальное повторное использование кода и т.д. Иными словами, этот код в наибольшей степени соответствует стилю Python. Само собой разумеется, что свой вклад вносит наличие качественных средств тестирования и хорошей документации.

### Пример 11.6. Файл представлений `blog (views.py)`

Все программные средства рассматриваемого приложения заключены в файле `views.py`, а вызов его компонентов осуществляется с помощью файла `URLconf`.

```
1 # views.py
2 from datetime import datetime
3 from django.http import HttpResponseRedirect
4 from django.views.generic.simple import direct_to_template
5 from blog.models import BlogPost, BlogPostForm
6
7 def archive(request):
8 posts = BlogPost.objects.all()[:10]
9 return direct_to_template(request, 'archive.html',
10 {'posts': posts, 'form': BlogPostForm()})
11
12 def create_blogpost(request):
13 if request.method == 'POST':
14 form = BlogPostForm(request.POST)
15 if form.is_valid():
16 post = form.save(commit=False)
17 post.timestamp=datetime.now()
18 post.save()
19 return HttpResponseRedirect('/blog/')
```

## Построчное объяснение

### Строки 1–5

Здесь приведено много операторов импорта, поэтому есть смысл привести еще одну рекомендацию: располагать инструкции импорта с учетом того, в каких местах приложения они применяются. Это означает, что в первую очередь необходимо обеспечивать доступ ко всем стандартным библиотечным модулям (таким как `datetime`) и пакетам. Вполне вероятно также, что будут обнаруживаться зависимости от модулей и пакетов платформы (`django.*`), поэтому относящиеся к ним инструкции импорта составляют следующий набор инструкций. Наконец, в приложении осуществляется импорт собственных модулей, в последнюю очередь (`blog.models`). Если инструкции импорта располагаются в таком порядке, то удастся предотвратить возникновение наиболее очевидных проблем с зависимостями.

## Строки 7–11

Функция `blog.views.archive()` — это основное представление в нашем приложении. Эта функция обеспечивает выборку из базы данных десяти объектов `BlogPost`, поступивших в последнюю очередь, обработку полученных данных и создание формы ввода для пользователя. Затем данные и форма передаются как контекст для обеспечения вызова шаблона `archive.html`. Обеспечивающая сокращение объема кода функция `render_to_response()` была заменена универсальным представлением `direct_to_template()` (позволяющим в этом процессе превратить `archive()` в по-лууниверсальное представление).

Первоначально функция `render_to_response()` обеспечивала не только получение имени и контекста шаблона, но и передачу объекта `RequestContext`, необходимого для проверки CSRF, и результирующего ответа назад клиенту. При проведении преобразования с использованием `direct_to_template()` не требовалось передавать экземпляр контекста запроса, поскольку все необходимые данные были отправлены на обработку в универсальное представление. Таким образом, разработчику приходилось заниматься лишь функциями основного приложения. Это своего рода дополнительное упрощение по отношению к (первоначальному) упрощению.

## Строки 12–19

Функция `blog.views.create_blogpost()` тесно связана с действием формы в файле `template/archive.html`, поскольку файл `URLconf` обеспечивает перенаправление всех данных, передаваемых по методу `POST`, в это представление. После подтверждения того, что запрос действительно передан по методу `POST`, создается объект `BlogPostForm` для извлечения полей формы, заполненных пользователем. Вслед за успешной проверкой с помощью инструкции, приведенной в строке 16, вызывается метод `form.save()` для возврата созданного экземпляра `BlogPost`.

Как было указано ранее, флаг `commit=False` служит указанием для метода `save()` на то, что этот экземпляр еще не нужно сохранять в базе данных (так как необходимо перед этим заполнить отметку времени). Это означает, что требуется явно вызвать метод `post.save()` экземпляра для его фактического сохранения. Если функция `is_valid()` возвращает `False`, этап сохранения данных пропускается. То же происходит, если запрос передавался по методу `GET`; эта ситуация возникает, если пользователь вводит необходимый URL непосредственно в адресной строке.

Наконец, рассмотрим файл шаблона `myblog/apps/templates/archive.html`, который представлен в примере 11.7.

### Пример 11.7. Файл шаблона главной страницы приложения `blog` (`archive.html`)

Файл шаблона включает код HTML, а также содержит инструкции, необходимые для управления выводом данных программным путем.

```
1 <!-- archive.html -->
2 <form action="/blog/create/" method=post>{% csrf_token %}
3 <table>{{ form }}</table>

4 <input type=submit>
5 </form>
6 <hr>
7
8 {% for post in posts %}
9 <h2>{{ post.title }}</h2>
```



```
10 <p>{{ post.timestamp }}</p>
11 <p>{{ post.body }}</p>
12 <hr>
13 {% endfor %}
```

## Построчное объяснение

### Строки 1–6

Первая половина шаблона представляет форму ввода данных пользователем. После передачи данных формы сервер вызывает на выполнение функцию представления `create_blogpost()`, которая описана выше, для создания новой записи `BlogPost` в базе данных. Данные переменной формы в строке 2 поступают из экземпляра `BlogPostForm`, который представляет собой форму, основанную на заданной модели данных (в табличном формате). Как было указано ранее, можно также выбрать другие варианты организации вывода. Выше было также указано, что значение `csrf_token` в строке 1 используется для защиты от атак CSRF. В связи с этим возникает также необходимость предоставить `RequestContext` в функции представления `archive()`, чтобы этот контекст мог использоваться в шаблоне.

### Строки 8–13

В последней половине шаблона просто берется набор (самое большее) из десяти (последних по времени) объектов `BlogPost` и осуществляется их обработка в цикле с передачей сведений об отдельных публикациях пользователя. При этом формируются горизонтальные линии (в том числе перед началом цикла) для визуального выделения данных.

## 11.15.2. Общие сведения о приложении блога

Разумеется, можно продолжать совершенствовать это приложение блога до бесконечности, добавляя все новые и новые функции (и многие разработчики не упускают такой возможности в своих программах), но мы остановимся на этом, поскольку и в таком виде приложение позволяет показать всю мощь платформы Django. (Предложения по дальнейшему усовершенствованию программы см. в упражнениях, приведенных в конце главы.) В ходе построения даже этого эскизного приложения блога удалось продемонстрировать целый ряд изящных средств Django, позволяющих решать многие задачи с наименьшими затратами. Рассматриваемые предложения включают следующее.

Это встроенный сервер для разработки, который позволяет программисту избавиться от необходимости использовать многие дополнительные инструменты и автоматически перезагружает код после проведения отдельных этапов его редактирования.

Это свойственный исключительно языку Python подход к созданию модели данных, который исключает необходимость создавать или сопровождать код SQL или файлы описания в формате XML.

Это приложение автоматического администрирования, предоставляющее полнофункциональные средства редактирования содержимого, с которыми могут успешно работать даже начинающие пользователи.

Это система шаблонов, которая может использоваться для формирования кода HTML, CSS, JavaScript или данных в любом текстовом формате вывода.

Это фильтры на основе шаблонов, с помощью которых можно изменять презентацию данных (таких как даты), не привлекая для этого бизнес-логику приложения.

Это система файлов URLconf, которая обеспечивает большую гибкость при проектировании URL, позволяя сохранять относящиеся к приложению части URL в самом приложении.

Это объекты `ModelForm`, которые предоставляют простой способ создания данных формы на основе модели данных при минимальных усилиях со стороны разработчика.

По завершении разработки рекомендуется разместить приложение на реальном сервере, подключенном к Интернету, и прекратить использовать сервер разработки. Замена адреса `localhost/127.0.0.1` реальным адресом позволяет практически подтвердить, что приложение будет успешно работать в производственной среде.

Читатели, которым понравился приведенный пример, могут найти его расширенную версию наряду с четырьмя другими аналогичными учебными приложениями по другим направлениям в книге *Python Web Development with Django*. После ознакомления с основами перейдем к более крупному и амбициозному практическому проекту: приложению Django, которое обрабатывает электронную почту, взаимодействует с Twitter, выполняет функции OAuth и служит пунктом запуска для многих других приложений.

## 11.16. \*Промежуточное приложение Django: TweetApprover

Приведенные выше сведения о платформе Django и опыт работы с этой платформой позволяют создавать реальные приложения, которые действительно могут принести пользу на практике. В этой второй части описания платформы Django в этой главе будет показано, как выполнять следующие задачи.

1. Создание крупных многокомпонентных веб-приложений (проектов) на платформе Django.
2. Применение сторонних библиотек.
3. Использование системы разрешений Django.
4. Передача электронной почты с помощью Django.

Это приложение соответствует примеру использования программных средств, который встречается все чаще на практике: в компании имеется учетная запись Twitter, и представители компании желают, чтобы постоянные сотрудники передавали с ее помощью объявления о продажах, новых товарах и т.д. В таком приложении нельзя обойтись без определенной бизнес-логики; кроме того, руководителю должна быть предоставлена возможность утверждать все твиты перед их отправкой.

Твит после утверждения рецензентом автоматически выводится в учетную запись Twitter компании; если же рецензент отклоняет твит, то данное сообщение передается назад автору с примечанием, указывающим причины и (или) содержащим рекомендации по исправлению сообщения, если необходимо или желательно повторить его отправку. Этот поток операций показан на рис. 11.21.

Для написания этого приложения с нуля потребовались бы значительные усилия. Необходимо построить модель данных, написать код подключения к базе данных для чтения и записи данных, отобразить информационные объекты на классы Python, написать код обработки веб-запросов, оформить данные в виде кода HTML перед их возвратом пользователю и т.д. Все эти задачи можно легко решить с помощью платформы Django. Безусловно, на платформе Django отсутствуют встроенные функциональные средства обмена данными с Twitter, но это задание может быть осуществлено с помощью имеющихся библиотек Python.

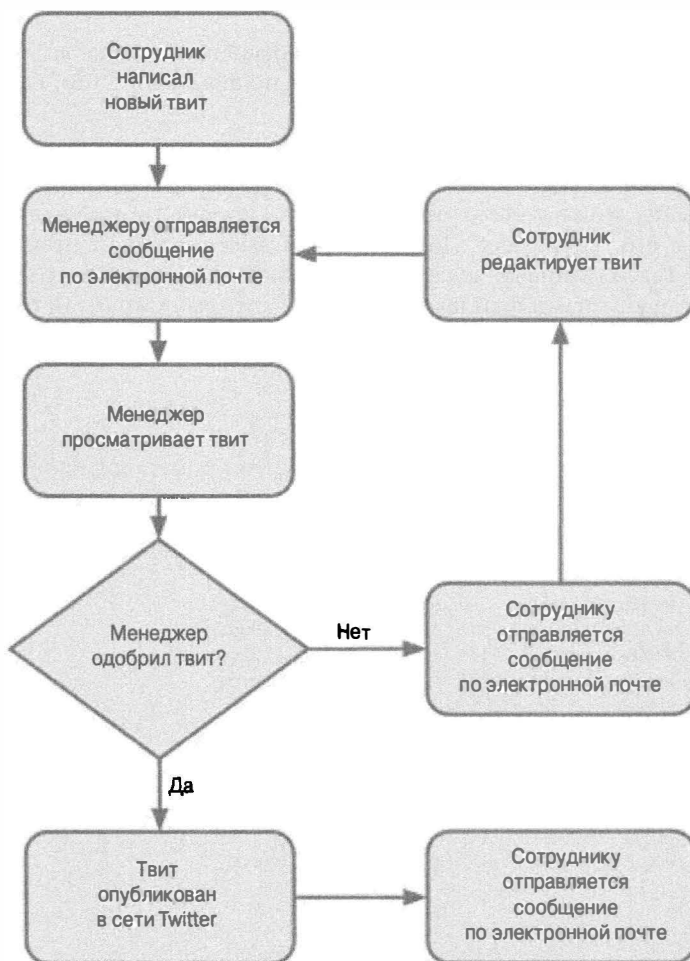


Рис. 11.21. Целевой поток операций TweetApprover

### 11.16.1. Создание структуры файла проекта

Проектирование нового приложения Django следует начинать с разработки структуры приложения. Платформа Django позволяет разделить проект на отдельные приложения. Проект, созданный в приведенном выше примере блога, состоял лишь

из одного приложения (blog), но, как было указано в начале главы, можно не ограничиваться одним приложением. Каждый раз, приступая к написанию нетривиального приложения, следует проверять возможность осуществления такой рекомендации, что проще справляться с несколькими небольшими приложениями, чем с крупным, единым, моноклитным приложением.

Приложение TweetApprover имеет два направления функционирования: с одной стороны, оно должно обслуживать обычных сотрудников (которые отправляют твиты), а с другой — руководителей (которые утверждают твиты). Создадим отдельные приложения Django для каждого направления в проекте TweetApprover; назовем одно приложение *poster*, а другое *approver*.

Прежде всего подготовим проект Django. В командной строке вызовите на выполнение команду `django-admin.py startproject`, по аналогии с тем, как было сделано раньше в проекте *mysite*.

```
$ django-admin.py startproject myproject
```

Для того чтобы можно было отличить этот проект от предыдущего проекта, *mysite*, назовем его “*myproject*”. Возможности выбора имени проекта почти неограниченны. :) Так или иначе, задание этого имени приводит к созданию каталога *myproject* со стандартными файлами заготовок, которые уже были описаны выше.

В командной строке перейдите в папку *myproject*, в которой можно создать два приложения, *poster* и *approver*:

```
$ manage.py startapp poster approver
```

В результате в каталоге *myproject* будут созданы подкаталоги *poster* и *approver*, содержащие стандартные файлы заготовок, которые относятся к приложениям. Первоначальная файловая структура должна выглядеть следующим образом:

```
$ ls -l *
-rw-r--r-- 1 wesley admin 0 Jan 11 10:13 __init__.py
-rwxr-xr-x 1 wesley admin 546 Jan 11 10:13 manage.py
-rw-r--r-- 1 wesley admin 4790 Jan 11 10:13 settings.py
-rw-r--r-- 1 wesley admin 494 Jan 11 10:13 urls.py
```

*approver*:

```
total 24
-rw-r--r-- 1 wesley admin 0 Jan 11 10:14 __init__.py
-rw-r--r-- 1 wesley admin 57 Jan 11 10:14 models.py
-rw-r--r-- 1 wesley admin 514 Jan 11 10:14 tests.py
-rw-r--r-- 1 wesley admin 26 Jan 11 10:14 views.py
```

*poster*:

```
total 24
-rw-r--r-- 1 wesley admin 0 Jan 11 10:14 __init__.py
-rw-r--r-- 1 wesley admin 57 Jan 11 10:14 models.py
-rw-r--r-- 1 wesley admin 514 Jan 11 10:14 tests.py
-rw-r--r-- 1 wesley admin 26 Jan 11 10:14 views.py
```

## Файл настроек

После создания нового проекта Django работа обычно начинается с открытия файла *settings.py* и его редактирования для конкретной установки. Для проекта

TweetApprover необходимо добавить несколько параметров, которые не заданы в файле по умолчанию. Прежде всего добавим новый параметр, который указывает, кто должен получать уведомление о том, что отправлены новые твиты, требующие утверждения.

```
TWEET_APPROVER_EMAIL = 'someone@mydomain.com'
```

Следует учитывать, что это не стандартный параметр Django, а значение, требуемое только для нашего приложения. Поскольку файл настроек — это стандартный файл Python, мы вправе добавлять в него собственные параметры. Однако проще не задавать такую дополнительную информацию отдельно для каждого из двух приложений, а предусмотреть отдельное хранилище для ее размещения на уровне проекта. Не забывайте, что следует заменить значение, приведенное выше в качестве примера, действительным адресом электронной почты руководителя, которому поручено рецензировать твиты.

Аналогичным образом необходимо передать Django указания, как следует отправлять электронную почту. Эти параметры считываются Django, но они не включены в файл настроек по умолчанию, поэтому необходимо их добавить отдельно.

```
EMAIL_HOST = 'smtp.mydomain.com'
EMAIL_HOST_USER = 'username'
EMAIL_HOST_PASSWORD = 'password'
DEFAULT_FROM_EMAIL = 'username@mydomain.com'
SERVER_EMAIL = 'username@mydomain.com'
```

Замените значения, приведенные выше в качестве примеров, значениями, соответствующими применяемому почтовому серверу. Если вы не имеете доступа к почтовому серверу, то можете пропустить эти пять параметров настройки электронной почты и закомментировать в проекте TweetApprover код, с помощью которого производится отправка электронной почты. Мы напомним об этом читателю после перехода к соответствующей части приложения. Сведения обо всех параметрах Django см. по адресу <http://docs.djangoproject.com/en/dev/ref/settings>.

В проекте TweetApprover для публикации твитов применяется общий API-интерфейс Twitter. При этом приложению необходимо передать учетные данные OAuth. (Дополнительные сведения о средствах OAuth приведены на следующей вкладке.) Учетные данные OAuth напоминают обычные логины и пароли, если не считать того, что одна пара учетных данных требуется для приложения (называемого потребителем в OAuth), а еще одна пара — для пользователя.

Для обеспечения работы вызовов API в Twitter должны быть отправлены все четыре фрагмента данных. Как и параметр TWEET\_APPROVER\_EMAIL в первом примере этого подраздела, эти параметры не являются стандартными параметрами Django и относятся только к приложению TweetApprover.

```
TWITTER_CONSUMER_KEY = '...'
TWITTER_CONSUMER_SECRET = '...'
TWITTER_OAUTH_TOKEN = '...'
TWITTER_OAUTH_TOKEN_SECRET = '...'
```

К счастью, система Twitter позволяет легко передавать эти четыре значения. Перейдите по адресу <http://dev.twitter.com>, входу, а затем щелкните на элементе Your Apps (Ваше приложение). После этого щелкните на элементе Register New App (Зарегистрировать новое приложение), если у вас еще нет приложения Twitter, или

выберите приложение, если таковое имеется. Для создания нового приложения заполните форму примерно так, как показано на рис. 11.22. В поле Application Website (Веб-сайт приложения) можно поместить любые данные. Обратите внимание на то, что на рисунках в этой главе используется имя TweetApprover. Очевидно, что это имя уже зарегистрировано, поэтому читателю придется создать приложение с другим именем.

The screenshot shows the 'New App' form on the Twitter Developer Portal. The form is titled 'TweetApprover Settings'. It contains the following fields and options:

- Application Name:** TweetApprover
- Description:** Workflow for approving company tweets.
- Application Website:** https://example.com
- Organization:** Twitter
- Application Type:** Client (selected), Browser
- Default Access type:** Read & Write (selected), Read-only
- Application Icon:** Twitter bird logo

Рис. 11.22. Регистрация нового приложения на сайте Twitter

Заполните форму и щелкните на элементе Save Application (Сохранить приложение), затем щелкните на кнопке Application Details (Сведения о приложении). На странице с этими сведениями найдите элемент OAuth 1.0a Settings (Параметры OAuth 1.0a). Из этого раздела скопируйте значения Consumer Key (Ключ потребителя) и Consumer Secret (Секрет потребителя) в переменные TWITTER\_CONSUMER\_KEY и TWITTER\_CONSUMER\_SECRET файла настроек соответственно.

Наконец, необходимо задать значения TWITTER\_OAUTH\_TOKEN и TWITTER\_OAUTH\_TOKEN\_SECRET. Щелкните на кнопке My Access Token (Мой маркер доступа), после чего откроется страница, подобная показанной на рис. 11.23, где находятся эти значения.



## Протокол OAuth, а также сопоставление авторизации и аутентификации

OAuth — это открытый протокол авторизации, который предоставляет безопасный и надежный способ обеспечения для приложений доступа к данным от имени пользователя через API-интерфейс. Он не только дает возможность получать доступ к приложениям, не раскрывая имя пользователя и пароль, но и позволяет легко отменять доступ. Протокол OAuth используется в Интернете во все большем количестве API-интерфейсов, например в Twitter. Дополнительные сведения о том, как работает OAuth, можно получить по следующим адресам:

<http://hueniverse.com/oauth>

<http://oauth.net>

<http://en.wikipedia.org/wiki/Oauth>

Необходимо учитывать, что OAuth относится к категории протоколов авторизации; эти протоколы не следует путать с протоколами проверки подлинности, такими как OpenID. Протоколы проверки подлинности предназначены не для обеспечения доступа к данным, а для идентификации, например, путем проверки имени пользователя и пароля. В качестве примера приложения, в котором применяются протоколы обоих типов, можно указать приложение, требующее, чтобы пользователь прошел проверку подлинности с помощью Twitter и авторизовал приложение (предоставил ему право) для вывода обновления статуса в свой поток Twitter.

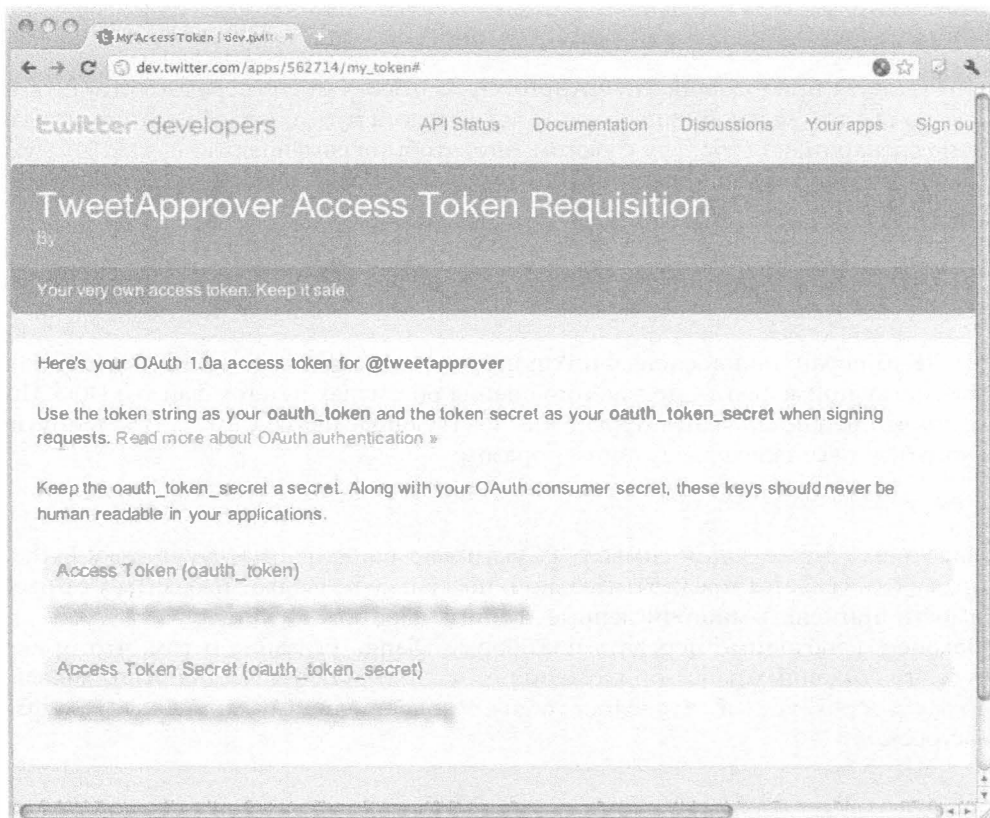


Рис. 11.23. Выделение токена OAuth и его секрета из социальной сети Twitter

Как обычно, необходимо откорректировать значение переменной `DATABASES`, чтобы она указывала на базу данных, в которой приложение `TweetApprover` хранит свои данные. В рассматриваемом простом приложении блога применяется база данных SQLite, но, как уже было сказано, читатель может использовать любую поддерживаемую базу данных. Чтобы продолжать использование SQLite, достаточно скопировать соответствующие настройки из рассматриваемого приложения блога. Следует обязательно выполнить команду `manage.py syncdb`, как и в предыдущих примерах.

Кроме того, как уже было сказано, обычно следует включить в работу приложение `admin` для Django, чтобы обеспечить простой доступ к данным CRUD. Перед этим в процессе работы с приложением блога программа `admin` главным образом выполнялась на сервере разработки, на котором обеспечивается автоматическая выборка изображений и таблиц стилей для страниц `admin`. Если применяется выделенный веб-сервер, такой как Apache, необходимо обеспечить, чтобы переменная `ADMIN_MEDIA_PREFIX` указывала на веб-каталог, в котором находятся файлы этих объектов. Подробные сведения об этом можно найти по адресу <http://docs.djangoproject.com/en/dev/howto/deployment/modwsgi/#serving-the-admin-files>.

Кроме того, можно указать для платформы Django, где находятся шаблоны HTML для веб-страниц, если для их хранения не применяется обычное расположение, такое как подкаталог `templates` в корневом каталоге приложения. Например, если желательно предусмотреть единое унифицированное расположение для этих файлов, как было сделано в данном приложении, то следует явно указать это в файле `settings.py`.

Что касается приложения `TweetApprover`, то имеет смысл сосредоточить необходимые файлы в отдельной папке `templates` каталога `myproject`. Для этого отредактируйте сценарий `settings.py` с учетом того, чтобы переменная `TEMPLATE_DIRS` указывала на этот физический каталог. На компьютере POSIX необходимая инструкция должна выглядеть следующим образом:

```
TEMPLATES_DIRS = (
 '/home/username/myproject/templates',
)
```

На ПК с операционной системой Windows путь к каталогу выглядит немного иначе, поскольку применяются другие соглашения об именах путей к файлам DOS. При условии, что нужно добавить проект к существующей папке `C:\py\django`, путь к каталогу будет выглядеть следующим образом:

```
r'c:\py\django\myproject\templates',
```

Напомним, что ведущий символ `"r"` указывает на бесформатную строку Python, которая здесь является предпочтительной, поскольку позволяет избавиться от необходимости применять многочисленные двойные обратные слэши.

Наконец, необходимо передать платформе Django указание о том, что применяются два созданных ранее приложения (`poster` и `approver`). Для этого добавим `'myproject.approver'` и `'myproject.poster'` к переменной `INSTALLED_APPS` в файле `настроек`.



## 11.16.2. Установка библиотеки Twython

Приложение TweetApprove позволяет публиковать твиты, предназначенные для ознакомления с ними других пользователей, с помощью общего API-интерфейса Twitter. К счастью, предусмотрено несколько библиотек высокого качества, с помощью которых действительно упрощается вызов этого API-интерфейса. Сведения о поддержке наиболее широко применяемых библиотек в Twitter приведены по адресу <http://dev.twitter.com/pages/libraries#python>. В следующей главе, посвященной веб-службам, рассматриваются библиотеки Twython и Tweepy.

### 2.6

Для этого приложения будет использоваться библиотека Twython для упрощения связи между приложением и Twitter. Эта задача осуществляется с помощью `easy_install`. (Можно также применить для установки инструмент `pip`.) Средство `easy_install` установит библиотеку `twython`, а также инструменты, необходимые для ее работы, `oauth2`, `httplib2` и `simplejson`. К сожалению, применяемые соглашения об именовании таковы, что последняя библиотека устанавливается под именем `json`, хотя в Python 2.6 и последующих версиях эта библиотека поставляется как `simplejson`. Так или иначе, `easy_install` устанавливает все три указанные библиотеки, от которых зависит `twython`, как показывает следующий вывод:

```
$ sudo easy_install twython
Password: *****
Searching for twython
...
Processing twython-1.3.4.tar.gz
Running twython-1.3.4/setup.py -q bdist_egg --dist-dir /tmp/easy_install-QrkR6M/
twython-1.3.4/egg-dist-tmp-PpJhMK
...
Adding twython 1.3.4 to easy-install.pth file
...
Processing dependencies for twython
Searching for oauth2
...
Processing oauth2-1.2.0.tar.gz
Running oauth2-1.2.0/setup.py -q bdist_egg --dist-dir /tmp/easy_install-br8On8/
oauth2-1.2.0/egg-dist-tmp-cx3yEm
Adding oauth2 1.2.0 to easy-install.pth file
...
Searching for simplejson
...
Processing simplejson-2.1.2.tar.gz
Running simplejson-2.1.2/setup.py -q bdist_egg --dist-dir /tmp/easy_install-
ZiTOri/simplejson-2.1.2/egg-dist-tmp-FWOza6
Adding simplejson 2.1.2 to easy-install.pth file
...
Searching for httplib2
...
Processing httplib2-0.6.0.zip
Running httplib2-0.6.0/setup.py -q bdist_egg --dist-dir /tmp/easy_install-rafDwd/
httplib2-0.6.0/egg-dist-tmp-zqPmMT
Adding httplib2 0.6.0 to easy-install.pth file
...
Finished processing dependencies for twython
```



### Устранение неполадок в установке

Если применяется версия Python 2.6, то процесс установки может пройти не так гладко, как здесь описано. Ниже приведено несколько примеров того, в чем могут возникнуть затруднения.

1. Автор столкнулся с затруднительной ситуацией, устанавливая библиотеку `simplejson` с применением версии Python 2.5 на компьютере Mac. В этом случае программа `easy_install` оказалась неспособной правильно выполнить все операции установки, вывела сообщения об ошибке и завершила работу, а система осталась в подвешенном состоянии. В этом случае мне пришлось прибегнуть к следующему старомодному способу:
  - Найти и загрузить `tar`-файл (в рассматриваемом примере, `simplejson`)
  - Раскрыть `tar`-файл, разархивировать дистрибутив и перейти в каталог верхнего уровня
  - Вызвать на выполнение команду `python setup.py install`
2. Один читатель обнаружил проблему, компилируя необязательный компонент, предназначенный для ускорения работы `simplejson`. В таком случае, учитывая то, что речь идет о расширении Python, в Windows необходимо предоставить компилятору все средства, предназначенные для построения расширения Python, включая `python.h` и пр. В системе Linux достаточно просто установить пакет `python-dev`.
3. Несомненно, читатель может столкнуться и с другими затруднениями, но можно надеяться на то, что приведенные выше рекомендации помогут ему в подобных случаях. Небольшие нестыковки возникают повсюду, поэтому не теряйте уверенности в себе, если вам придется с этим столкнуться. Помощь можно получить даже там, где вы на это не рассчитывали!

После успешного завершения установки всех необходимых компонентов должно быть принято решение о том, какие URL будут использоваться в приложении `TweetApprover` и как они будут связаны к различными действиями пользователя.

### 11.16.3. Структура URL

Прежде всего следует выработать стратегию единообразного формирования URL. Для этого обозначим имена всех функций приложения `poster` с применением URL, которые начинаются с подстроки `/post`, а для приложения `approver` предусмотреть URL, начинающиеся с подстроки `/approve`. Это означает, что после копирования компонентов `TweetApprover` в домен `example.com` те URL, которые относятся к приложению `poster`, будут начинаться с подстроки `http://example.com/post`, а в URL приложения `approver` будут иметь префикс `http://example.com/approve`.

Теперь перейдем к более подробному рассмотрению страниц в компонентах `/post`, которые используются для создания новых твитов. Одна из страниц должна предназначаться для отправки совершенно нового твита; эта страница должна развернуться перед глазами пользователя после поступления в программу URL с суффиксом `/post` (за которым не следует какое-либо продолжение). После отправки твита пользователем должна открыться страница с подтверждением успешной отправки; предусмотрим для этой страницы суффикс `/post/thankyou`. Наконец, нам потребуются URL, позволяющие пользователю открыть существующий твит

для редактирования; установим соглашение, что для этих URL будет применяться обозначение `/post/edit/X`, где `X` — идентификатор твита, подлежащего редактированию.

Страницы для программного диспетчера должны иметь URL, которые содержат обозначение `/approve`; необходимо предусмотреть, чтобы перед пользователем после получения доступа к этим URL открывался список подлежащих публикации и опубликованных твитов. Кроме того, требуются такие страницы, которые позволяют просматривать отдельно каждый конкретный твит и оставлять относящийся к нему комментарий; предусмотрим для этих страниц суффикс `/approve/review/X`, где `X` — идентификатор твита.

Наконец, необходимо решить, какая страница должна отображаться, если пользователь переходит к URL без каких-либо суффиксов (такому как `example.com/`). Большинство пользователей TweetApprover будут составлять сотрудники, создающие новые твиты, поэтому целесообразно, чтобы пустой URL указывал на ту же страницу, что и URL с суффиксом `/post`.

Выше было показано, как на платформе Django используются файлы конфигурации для обеспечения соответствия между URL и конкретными компонентами кода приложения. На уровне проекта, в каталоге `myproject`, находится файл `urls.py`, который служит для платформы Django указанием по перенаправлению запросов к тем или иным приложениям в проекте. В примере 11.8 представлен файл, который реализует описанную выше структуру URL:

#### Пример 11.8. Файл `URLconf` проекта (`myproject/urls.py`)

Как и в предыдущем примере, этот файл `URLconf` проекта также применяется в приложении или располагается на административном сайте.

```
1 # urls.py
2 from django.conf.urls.defaults import *
3 from django.contrib import admin
4 admin.autodiscover()
5
6 urlpatterns = patterns('',
7 (r'^post/', include('myproject.poster.urls')),
8 (r'^$', include('myproject.poster.urls')),
9 (r'^approve/', include('myproject.approver.urls')),
10 (r'^admin/', include(admin.site.urls)),
11 (r'^login', 'django.contrib.auth.views.login',
12 {'template_name': 'login.html'}),
13 (r'^logout', 'django.contrib.auth.views.logout'),
14)
```

## Построчное объяснение

### Строки 1–4

Первые несколько строк представляют собой стандартный набор инструкций, которые, по-видимому, встречаются во всех файлах `URLconf`: соответствующие инструкции импорта, а также инструкции подключения административных средств для разработки. (Если разработка уже завершена и (или) отпала необходимость в использовании административных средств, то эти инструкции можно исключить.)

## Строки 6–14

Большой интерес в этом файле представляет переменная `urlpatterns`. В строке 7 представлено указание для платформы Django, что при поступлении любого URL, который начинается с `post/` (после имени домена), необходимо проводить проверку URL по конфигурации `myproject.poster.urls`. Эта конфигурация определена в файле `myproject/poster/urls.py`. Следующая строка (строка 8) указывает, что все пустые URL (после имени домена) также должны обрабатываться в соответствии с конфигурацией приложения `poster`. Строка 9 служит для Django указанием, что URL, начинающиеся с `approve/`, должны быть направлены к приложению `approver`.

Наконец, файл включает директивы для URL, которые приводят к приложению `admin` (строка 10), а также к страницам входа и выхода из системы (строки 11 и 12). Значительная часть этих функциональных средств входит в состав Django, поэтому писать для них код не требуется. Вопросы аутентификации еще не рассматривались, но и эта задача решается просто, путем включения еще нескольких двухэлементных кортежей в файл `URLconf`. Django предоставляет собственную систему аутентификации, но разработчик может создать таковую самостоятельно. В главе 12 будет показано, что подсистема Google App Engine поддерживает два варианта аутентификации: учетные записи Google или федеративный вход в систему с использованием идентификатора OpenID.

Подводя итоги, можно отметить, что URL, составленные с применением всех возможных опций, выглядят так, как показано в табл. 11.3.

**Таблица 11.3.** Различные варианты URL, в которых учитываются особенности данного проекта, а также реализуются соответствующие действия

URL	Действие
<code>/post</code>	Предложение к созданию нового твита или публикации в блоге
<code>/post/edit/X</code>	Редактирование публикации X
<code>/post/thankyou</code>	Выражение благодарности пользователю после представления публикации
<code>/</code>	То же, что и <code>/post</code>
<code>/approve</code>	Формирование списка всех приостановленных и опубликованных твитов
<code>/approve/review/X</code>	Просмотр твита X
<code>/admin</code>	Переход к узлу <code>admin</code> для конкретного проекта
<code>/login</code>	Регистрация пользователя в системе
<code>/logout</code>	Отмена регистрации пользователя

Как показано в табл. 11.3, основное назначение файла `URLconf` проекта состоит в том, чтобы перенаправлять запросы к соответствующим приложениям и их обработчикам, поэтому перейдем к рассмотрению файлов `urls.py`, относящихся к уровню приложений. Начнем с приложения `poster`.

Как показывает приведенный выше файл URLconf проекта, те URL, которые содержат подстроки `/post/` (или `/`), должны перенаправляться к файлу URLconf приложения `poster`, `myproject/poster/urls.py`. Этот файл, рассматриваемый в примере 11.9, предназначен для отображения оставшейся части URL на действительный код, который должен выполняться в рамках приложения `poster`.

#### Пример 11.9. Файл URLconf (`urls.py`) приложения `poster`

Файл URLconf для приложения `poster` обрабатывает действия автора публикаций.

```
1 from django.conf.urls.defaults import *
2
3 urlpatterns = patterns('myproject.poster.views',
4 (r'^$', 'post_tweet'),
5 (r'^thankyou', 'thank_you'),
6 (r'^edit/(?P<tweet_id>\d+)', 'post_tweet'),
7)
```

Регулярные выражения, предусмотренные в этом файле, предназначены для обработки только той части URL, которая следует за подстрокой `/post/`. На основании значения первого параметра для функции `patterns()` можно определить, что все функции представления представлены в файле `myproject/poster/views.py`. Что касается первого шаблона URL, то, если он пуст (если значением исходного запроса было `/post/` или `/`), вызывается представление `post_tweet()`. Если эта часть равна `thankyou`, вызывается представление `thank_you()`. Наконец, если эта часть URL равна `edit/X`, где `X` — число, вызывается представление `post_tweet()` и значение `X` передается как параметр `tweet_id` для метода. Очевидно, что в этом коде есть много интересного. Для дополнительного изучения применяемого здесь синтаксиса регулярных выражений, в котором полученные результаты сопоставления присваиваются переменным, обозначенным именами, а не целыми числами (как предусмотрено по умолчанию), еще раз вернитесь к главе 1.

Как уже было сказано, данный проект был разбит на два приложения, поэтому объем файлов URLconf и файлов сценариев с функциями просмотра сведен к минимуму. Это также позволяет упростить изучение приложений и их повторное использование. Выше была описана установка приложения `poster`, а теперь перейдем к аналогичному описанию для приложения `approver`.

По такому же принципу, как при изучении файла URLconf приложения `poster`, рассмотрим файл `myproject/approver/urls.py`, который показан в примере 11.10. Этот файл используется платформой Django при обработке запроса для URL, который начинается с подстроки `/approve/`. Если путь не продолжается за подстрокой `/approve/`, вызывается представление `list_tweets()`, а если путь в URL согласуется с `/approve/review/X`, вызывается представление `review_tweet(tweet_id=X)`.

#### Пример 11.10. Файл URLconf (`urls.py`) приложения `approver`

Файл URLconf для приложения `approver` применяется для обработки действий лица, утверждающего твиты

```
1 from django.conf.urls.defaults import *
2
3 urlpatterns = patterns('myproject.approver.views',
```

```

4 (r'^$', 'list_tweets'),
5 (r'^review/(?P<tweet_id>\d+)$', 'review_tweet'),
6)

```

Этот файл `URLconf` короче, поскольку приложение `approver` поддерживает меньшее количество действий. На данный момент точно известно, куда должны быть направлены пользователи с учетом пути, входящего URL. Теперь перейдем к рассмотрению дополнительных сведений о модели данных, используемой в этом проекте.

#### 11.16.4. Модель данных

В задачу `TweetApprover` входит сохранение твитов в базе данных. Руководители, просматривающие твиты, должны иметь возможность их аннотировать, чтобы можно было сопровождать твиты несколькими комментариями. И твиты, и комментарии состоят из целого ряда полей данных, как показано на рис. 11.24.

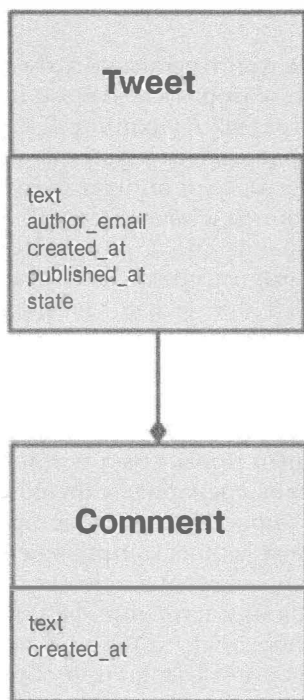


Рис. 11.24. Модель данных для `TweetApprover`

Поле состояния предназначено для сохранения сведений о том, на каком этапе жизненного цикла находится каждый твит. На рис. 11.25 показано, что количество различных состояний равно трем, и платформа Django позволяет добиться того, чтобы ни один твит не оказался в каком-либо другом состоянии.

Как уже было сказано, платформа Django действительно упрощает задачу создания необходимых таблиц в базе данных, а также чтение и запись объектов `Comment` и `Tweet`. В данном случае модель данных может быть определена в сценарии `myproject/poster/models.py` или `myproject/approver/models.py`. Как показано в

примере 11.11, мы произвольно выбрали первый из этих сценариев. Однако это не означает, что приложение approver не сможет получить доступ к модели данных.

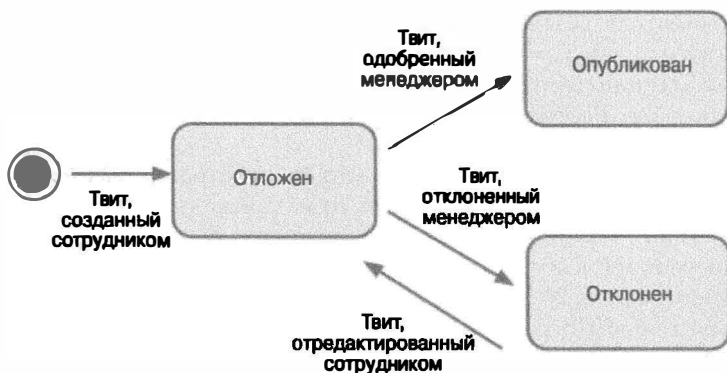


Рис. 11.25. Модель состояния для твитов в задаче TweetApprover

#### Пример 11.11. Файл моделей данных (models.py) для приложения poster

Файл модели данных для приложения poster содержит классы для публикаций (Tweet), а также для отзывов (Comment).

```

1 from django.db import models
2
3 class Tweet(models.Model):
4 text = models.CharField(max_length=140)
5 author_email = models.CharField(max_length=200)
6 created_at = models.DateTimeField(auto_now_add=True)
7 published_at = models.DateTimeField(null=True)
8 STATE_CHOICES = (
9 ('pending', 'pending'),
10 ('published', 'published'),
11 ('rejected', 'rejected'),
12)
13 state = models.CharField(max_length=15, choices=STATE_CHOICES)
14
15 def __unicode__(self):
16 return self.text
17
18 class Meta:
19 permissions = (
20 ("can_approve_or_reject_tweet",
21 "Can approve or reject tweets"),
22)
23
24 class Comment(models.Model):
25 tweet = models.ForeignKey(Tweet)
26 text = models.CharField(max_length=300)
27 created_at = models.DateTimeField(auto_now_add=True)
28
29 def __unicode__(self):
30 return self.text

```

Первой моделью данных является класс `Tweet`. Этот класс представляет сообщение, которое обычно именуется публикацией, или твитом. Это сообщение авторы пытаются отправить в службу Twitter, но прежде его должен утвердить администратор или руководитель. Должна быть предусмотрена возможность для администраторов или руководителей комментировать объекты `Tweet`, поэтому имеются объекты `Comment`, позволяющие сопровождать объекты `Tweet` комментариями в количестве от нуля или больше. Рассмотрим дополнительные сведения об указанных классах и их атрибутах.

Длина полей `text` и `author_email` класса `Tweet` ограничивается 140 и 200 символами соответственно. Что касается твитов, то их длина ограничивается максимально допустимой длиной сообщения SMS или текстового сообщения на мобильных телефонах, а если речь идет об электронной почте, то чаще всего применяются адреса, которые намного короче 200 символов.

Для работы с полем `created_at` мы используем удобное средство `auto_now_add` платформы Django. Это означает, что при создании каждого нового твита и сохранении его в базе данных поле `created_at` заполняется значениями текущей даты и времени, если пользователь не задает эти значения явно. Еще одно поле с типом данных `DateTimeField` (`published_at`) может иметь `NULL`-значение. Оно используется для твитов, которые еще не опубликованы в сети Twitter.

Далее приведены перечисление состояний и определение поля `state`. Платформа Django вызывает только эти состояния и связывает с ними переменную состояния, поэтому гарантируется, что объекты `Tweet` не будут иметь другие состояния, а только одно из трех допустимых состояний. Определение метода `__unicode__()` служит для Django указанием, что атрибут `text` каждого объекта `Tweet` должен быть отображен на веб-сайте администрирования. Напомним, что ранее в этой главе были указаны причины, по которым объекты `BlogPost` оказались не столь полезными. Прежде всего, ни один из них не является объектом `Tweet`. В частности, в большинстве случаев объекты такого типа обозначаются одной и той же надписью.

Ранее в этой главе был представлен внутренний класс `Meta`, но следует помнить, что его назначение состоит в передаче Django указаний о дополнительных специальных требованиях к информационному объекту. В данном случае он используется в качестве указания для Django об использовании нового флага разрешения. По умолчанию Django создает флаги разрешения для добавления, изменения и удаления всех сущностей в модели данных. В приложении может быть выполнена проверка того, имеет ли пользователь, зарегистрированный в настоящее время, разрешение на добавление объекта `Tweet`; разрешения зарегистрированным пользователям администратор узла может предоставить с помощью программы `admin` платформы Django.

Очевидно, что эта функция успешно реализована, но для приложения `TweetApprover` требуется еще один специальный флаг разрешения для публикации твита в сети Twitter. Такую задачу нельзя свести к добавлению, изменению или удалению объектов `Tweet`. Django создает соответствующие флаги в базе данных, добавляя этот флаг к классу `Meta`. Ниже будет показано, как обеспечить чтение этого флага для проверки того, что право утверждать или отвергать твиты предоставлено только руководителям.

Класс `Comment` имеет второстепенное значение, но все равно заслуживает обсуждения. В этом классе имеется поле `ForeignKey`, которое указывает на класс `Tweet`. Это служит для Django указанием, что должна быть создана связь "один ко многим" между объектами `Tweet` и `Comment` в базе данных. Как и объекты `Tweet`, записи `Comment`



также имеют поля `text` и `created_at`, которые предназначены для той же цели, что и их аналоги в объектах `Tweet`.

После завершения подготовки файла модели можно выполнить команду `syncdb` для создания таблиц в базе данных и формирования учетной записи суперпользователя:

```
$./manage.py syncdb
```

Наконец, как показано в примере 11.12, необходимо добавить файл `myproject/poster/admin.py` для передачи Django указания, что должно быть разрешено редактирование объектов `Tweet` и `Comment` с помощью программы `admin`.

#### Пример 11.12. Регистрация моделей с помощью программы `admin` (`admin.py`)

Действия пользователя, который подготавливает публикации, реализуются с применением файла `URLconf` для приложения `poster`

```
1 from django.contrib import admin
2 from models import *
3
4 admin.site.register(Tweet)
5 admin.site.register(Comment)
```

Таким образом, для платформы Django подготовлены все компоненты, необходимые для автоматического формирования веб-сайта администрирования для данного приложения. Провести проверку того, как работает веб-сайт администрирования, можно прямо сейчас, до написания представлений для отправителей твитов и тех, кто утверждает эти твиты, `poster` и `approver`. Для этого необходимо на время закомментировать строки 6–8 в примере 11.13 (`myproject/urls.py`), где находятся ссылки на эти представления. Затем следует получить доступ к веб-сайту администрирования (рис. 11.26) с помощью URL `/admin`. Необходимо учитывать, что указанные строки должны быть снова раскомментированы после создания представлений `poster/views.py` и `approver/views.py`.

#### Пример 11.13. (Временный) файл `URLconf` проекта (`myproject/urls.py`)

После того как ссылки на еще не написанные представления были исключены, можно проверить работу веб-сайта администрирования Django.

```
1 from django.conf.urls.defaults import *
2 from django.contrib import admin
3 admin.autodiscover()
4
5 urlpatterns = patterns('',
6 #(r'^post/', include('myproject.poster.urls')),
7 #(r'^$', include('myproject.poster.urls')),
8 #(r'^approve/', include('myproject.approver.urls')),
9 (r'^admin/', include(admin.site.urls)),
10 (r'^login', 'django.contrib.auth.views.login',
11 {'template_name': 'login.html'}),
12 (r'^logout', 'django.contrib.auth.views.logout'),
13)
```

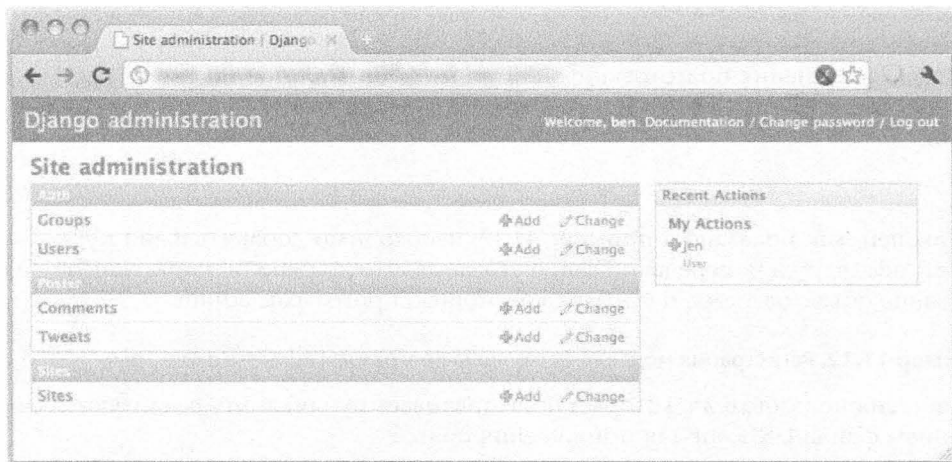


Рис. 11.26. Встроенный узел администрирования Django

Как показано на рис. 11.27, после создания нового пользователя открывается возможность задать специальный флаг разрешения для элемента интерфейса “Can approve or reject tweets” (Может утверждать или отвергать твиты). Создайте пользователя и предоставьте ему указанное разрешение; это потребуется при проверке приложения TweetApprover в дальнейшем. После создания нового пользователя появится возможность отредактировать профиль пользователя и задать специальные разрешения. (До создания нового пользователя возможность задать эти разрешения отсутствует.)

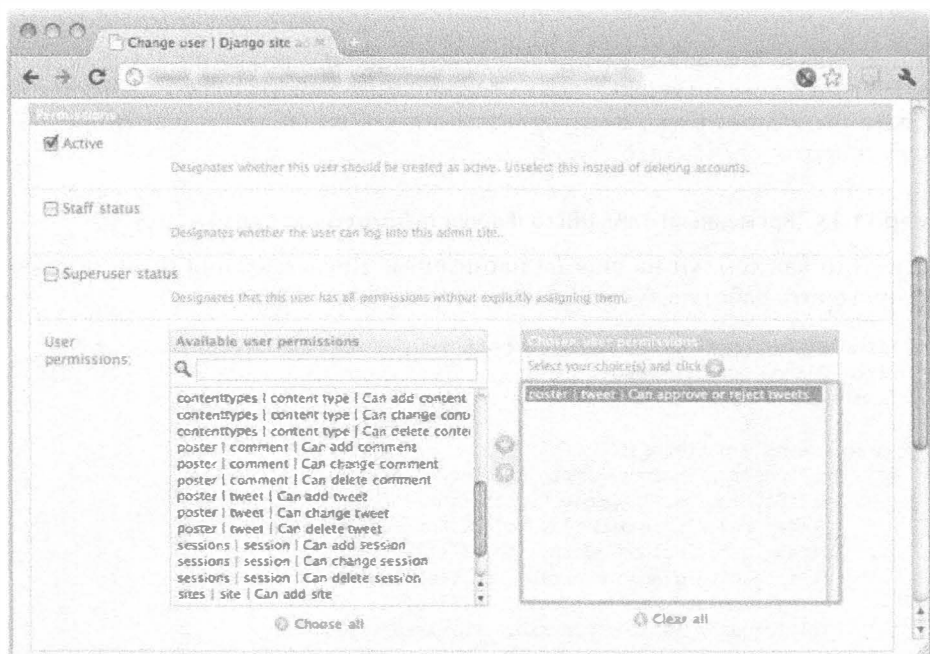


Рис. 11.27. Назначение специальных разрешений для нового пользователя



## Минимизация объема кода

Приведенное выше описание было в основном посвящено вопросам настройки конфигурации, а программированию фактически уделялось мало внимания. Одним из преимуществ Django является то, что после правильной настройки конфигурации отпадает необходимость в написание большого объема кода. С точки зрения программиста немного странно сталкиваться с таким подходом, когда основной объем работы по формированию кода выполняется не им, а программой. Необходимо учитывать, что платформа Django была создана в такой компании, где подавляющей частью пользователей являются журналисты, а не разработчики веб-приложений. С помощью этой платформы решена очень важная задача — расширение возможностей авторов корреспонденций и других представителей средств массовой коммуникации, которые после освоения навыков работы с компьютером дополнительно приобретают возможность разрабатывать веб-приложения, не затрачивая при этом столь значительных усилий, которые вынуждали бы их уделять меньше времени своей основной работе. Иными словами, основой подхода, реализованного создателями платформы Django, является обеспечение максимального удобства работы для пользователей (не имеющих квалификацию разработчиков).

### 11.16.5. Передача новых твитов для рецензирования

При создании приложения poster платформа Django формирует почти пустой файл `views.py` в каталоге этого приложения. Именно в этом файле должны быть определены методы, на которые имеются ссылки в файлах конфигурации URL. Образец файла `myproject/poster/views.py` в законченном виде приведен в примере 11.14.

#### Пример 11.14. Функции представления для приложения poster (`views.py`)

Здесь определены основные программные средства для приложения poster.

```

1 # poster/views.py
2 from django import forms
3 from django.forms import ModelForm
4 from django.core.mail import send_mail
5 from django.db.models import Count
6 from django.http import HttpResponseRedirect
7 from django.shortcuts import get_object_or_404
8 from django.views.generic.simple import direct_to_template
9 from myproject import settings
10 from models import Tweet
11
12 class TweetForm(forms.ModelForm):
13 class Meta:
14 model = Tweet
15 fields = ('text', 'author_email')
16 widgets = {
17 'text': forms.Textarea(attrs={'cols': 50, 'rows': 3}),
18 }
19
20 def post_tweet(request, tweet_id=None):
21 tweet = None
22 if tweet_id:
23 tweet = get_object_or_404(Tweet, id=tweet_id)
24 if request.method == 'POST':

```

```
25 form = TweetForm(request.POST, instance=tweet)
26 if form.is_valid():
27 new_tweet = form.save(commit=False)
28 new_tweet.state = 'pending'
29 new_tweet.save()
30 send_review_email()
31 return HttpResponseRedirect('/post/thankyou')
32 else:
33 form = TweetForm(instance=tweet)
34 return direct_to_template(request, 'post_tweet.html',
35 {'form': TweetForm(instance=tweet)})
36
37 def send_review_email():
38 subject = 'Action required: review tweet'
39 body = ('A new tweet has been submitted for approval. '
40 'Please review it as soon as possible.')
41 send_mail(subject, body, settings.DEFAULT_FROM_EMAIL,
42 [settings.TWEET_APPROVER_EMAIL])
43
44 def thank_you(request):
45 tweets_in_queue = Tweet.objects.filter(
46 state='pending').aggregate(Count('id')).values()[0]
47 return direct_to_template(request, 'thank_you.html',
48 {'tweets_in_queue': tweets_in_queue})
```

## Построчное объяснение

### Строки 1–10

Это обычные инструкции импорта, которые позволяют ввести в действие необходимые функциональные средства Django.

### Строки 12–18

Вслед за всеми инструкциями импорта находится определение `TweetForm`, основанное на сущности `Tweet`. Класс `TweetForm` определен как содержащий только поля `text` и `author_email`, поскольку для пользователя остальные его компоненты не представляют интереса. Кроме того, указано, что текстовое поле должно отображаться с помощью графического элемента `textarea` (многострочное текстовое поле) языка HTML, а не просто отдельного текстового поля, имеющего большую длину. Это определение формы будет использоваться в методе `post_tweet()`.

### Строки 20–36

Метод `post_tweet()` вызывается при обеспечении доступа к URL `/post` или `/post/edit/X`. Это поведение было определено в подготовленных ранее файлах конфигурации URL. Данный метод выполняет одно из четырех действий, как показано на рис. 11.28.

Пользователь начинает работу с одного из верхних полей, затем переходит к нижележащему полю, щелкая в форме на кнопке передачи формы. Приведенный здесь пример использования и такое расположение инструкций `if` являются типичными для методов представления Django, предназначенных для работы с формами. После выполнения в этом методе всех основных действий происходит вызов шаблона `post_tweet.html` и передача его экземпляру `TweetForm`. Кроме того, следует отметить,

что электронная почта передается рецензенту путем вызова метода `send_review_email()`. Удалите эту строку, если у вас нет доступа к почтовому серверу, и не вводите дополнительные сведения о почтовом сервере в файл настроек.

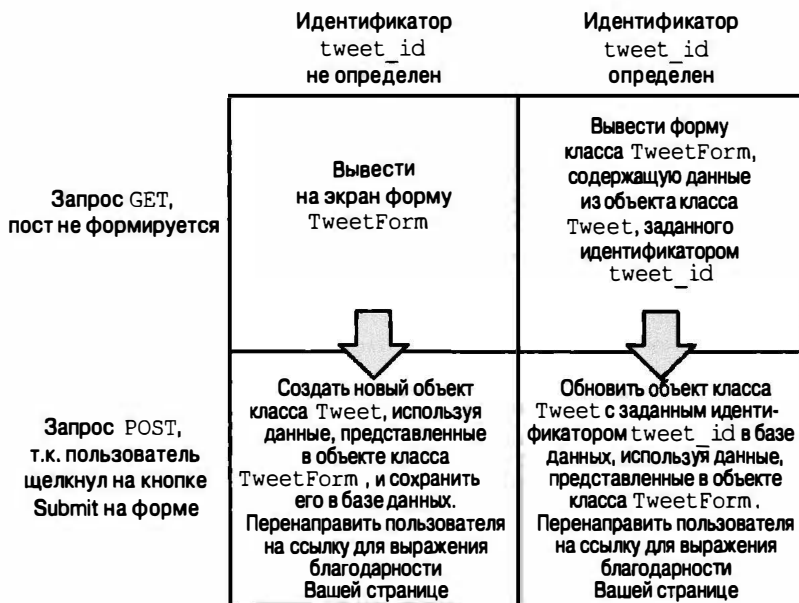


Рис. 11.28. Поведение метода `post_tweet()`

Характерной особенностью этого блока кода является также то, что в нем определена новая функция, еще не рассматриваемая в этой книге, обозначенная как `get_object_or_404()`. Некоторые читатели могут прийти к выводу, что код этой функции практически не имеет отношения к рассматриваемому приложению, но в действительности функция предоставляет вспомогательные средства, в которых часто нуждаются разработчики. В функции используются класс модели данных и первичный ключ для осуществления попытки выборки объекта с указанным идентификатором. Если поиск объекта выполняется успешно, объект присваивается переменной твита. В противном случае активизируется ошибка HTTP 404 (не найдено). По замыслу автора, такое поведение функции должно поставить под контроль пользователей, пытающихся вопреки правилам манипулировать параметрами URL вручную. В таком случае перед пользователем в браузере отображается сообщение об ошибке, независимо от того, произошло ли вмешательство в работу приложения с хорошими или дурными намерениями.

## Строки 38–42

Метод `send_review_email()` представляет собой просто вспомогательное средство, которое используется для отправки электронной почты руководителю после передачи нового твита на рецензирование или обновления существующего твита. При этом используется метод `send_mail()` платформы Django, который отправляет электронную почту с помощью сервера, с указанием учетных данных, которые представлены в файлах с параметрами.

## Строки 44–48

Метод `thank_you()`, показанный в примере 11.16, вызывается при перенаправлении пользователя по адресу `/post/thankyou/` вслед за передачей формы `TweetForm`. В этом методе используются встроенные функциональные средства доступа к данным платформы Django для запроса в базе данных количества объектов `Tweet`, находящихся в настоящее время в состоянии `pending`. Читатели, имеющие подготовку в области реляционных баз данных, вне всякого сомнения, заметят, что ORM Django выдает команду SQL, которая может выглядеть примерно так: `SELECT COUNT(id) FROM Tweet WHERE state= "pending"`. Превосходная особенность компонента ORM состоит в том, что он позволяет формировать цепочки вызова методов с поддержкой объектов (как в приведенном выше примере) даже тем пользователям, которые не знают языка SQL; ORM, как по волшебству, выдает необходимый код SQL от имени разработчика.

После получения данных об общем количестве публикаций со статусом `pending` (ожидающих обработки) приложение вызывает шаблон `thank_you.html` и с его помощью отображает пользователю полученную сумму. Как показано на рис. 11.30, в этом шаблоне применяются разные форматы сообщений, если количество неподтвержденных твитов больше единицы и равно единице. В примерах 11.15 и 11.16 показаны файлы шаблонов, используемые в приложении `poster`.

### Пример 11.15. Шаблон с формой для передачи данных (`post_tweet.html`)

Форма для передачи данных в приложении `poster` внешне выглядит пустой, поскольку для вывода данных дополнительно применяется модель `TweetForm`.

```

1 <html>
2 <body>
3 <form action="" method="post">{% csrf_token %}
4 <table>{{ form }}</table>
5 <input type="submit" value="Submit" />
6 </form>
7 </body>
8 </html>
```

### Пример 11.16. Шаблон `thank_you()` после отправки формы (`thank_you.html`)

В форме с подтверждением отправки (`thank you`) для приложения `poster` предусмотрены средства, позволяющие сообщить пользователю о ходе обработки отправленных им твитов.

```

1 <html>
2 <body>
3 Thank you for your tweet submission. An email has been sent
4 to the assigned approver.
5 <hr>
6 {% if tweets_in_queue > 1 %}
7 There are currently {{ tweets_in_queue }} tweets waiting
8 for approval.
9 {% else %}
10 Your tweet is the only one waiting for approval.
11 {% endif %}
12 </body>
13 </html>
```

Шаблон `post_tweet.html` является несложным: он обеспечивает лишь отображение формы в виде таблицы HTML и добавляет в нижней части кнопку передачи формы. Сравните шаблон в примере 11.15 с формой, которая использовалась выше в рассматриваемом приложении блога; он почти полностью подходит для повторного применения. Напомним, что всегда рекомендуется повторное использование кода, но эта рекомендация практически не распространяется на код HTML.

На рис. 11.29 показан вывод шаблона, представляющий собой входную форму для пользователей, желающих создать публикацию или твит. Перейдем к рассмотрению шаблона, предназначенного для формирования страницы с подтверждением отправки, которая разворачивается перед пользователями последней. Эта страница показана на рис. 11.30.

A screenshot of a web browser window with the address bar showing 'tweet\_approve.momander.w'. The page contains a form with a 'Text:' label, a text input field containing 'Check out our everyday low prices!', an 'Author email:' label, a text input field containing 'ben@gmail.com', and a 'Submit' button.

Рис. 11.29. Форма отправки новых твитов, которая имеется в каталоге `/post`

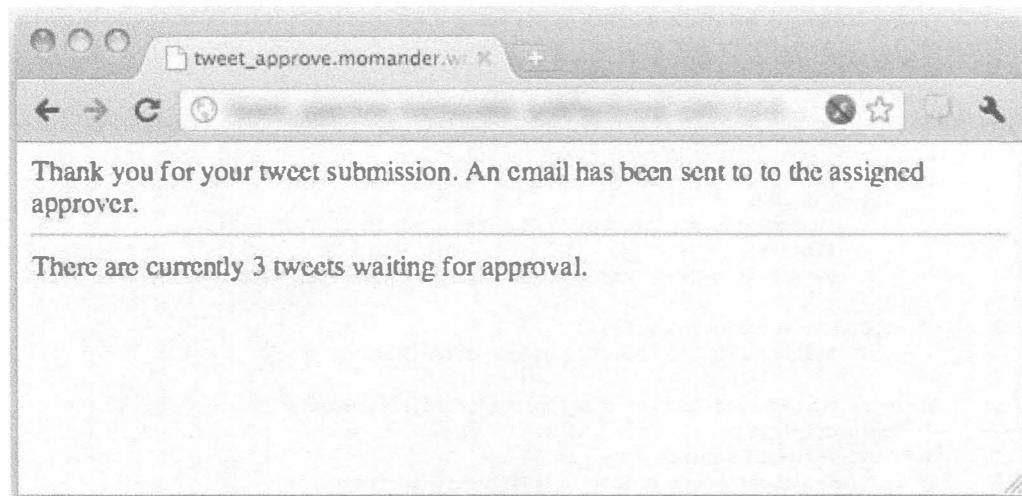
A screenshot of a web browser window with the address bar showing 'tweet\_approve.momander.w'. The page displays a confirmation message: 'Thank you for your tweet submission. An email has been sent to to the assigned approver.' followed by a horizontal line and the text 'There are currently 3 tweets waiting for approval.'

Рис. 11.30. Страница с подтверждением передачи, которая отображается после отправки нового твита

## 11.16.6. Проверка твитов

Выше было подробно описано приложение `poster`, а теперь перейдем к рассмотрению приложения `approver`. В файле `myproject/approver/urls.py` происходит вызов методов `list_tweets()` и `review_tweet()`, представленных в файле `myproject/approver/views.py`. Этот файл полностью приведен в примере 11.17.

### Пример 11.17. Функции представления для приложения `approver` (`views.py`)

К основным функциям приложения `approver` относятся развертывание формы, отображение публикаций для проверки и помощь в подготовке решений.

```

1 # approver/views.py
2 from datetime import datetime
3 from django import forms
4 from django.core.mail import send_mail
5 from django.core.urlresolvers import reverse
6 from django.contrib.auth.decorators import permission_required
7 from django.http import HttpResponseRedirect
8 from django.shortcuts import get_object_or_404
9 from django.views.generic.simple import direct_to_template
10 from twython import Twython
11 from myproject import settings
12 from myproject.poster.views import *
13 from myproject.poster.models import Tweet, Comment
14
15 @permission_required('poster.can_approve_or_reject_tweet',
16 login_url='/login')
17 def list_tweets(request):
18 pending_tweets = Tweet.objects.filter(state=
19 'pending').order_by('created_at')
20 published_tweets = Tweet.objects.filter(state=
21 'published').order_by('-published_at')
22 return direct_to_template(request, 'list_tweets.html',
23 {'pending_tweets': pending_tweets,
24 'published_tweets': published_tweets})
25
26 class ReviewForm(forms.Form):
27 new_comment = forms.CharField(max_length=300,
28 widget=forms.Textarea(attrs={'cols': 50, 'rows': 6}),
29 required=False)
30 APPROVAL_CHOICES = (
31 ('approve', 'Approve this tweet and post it to Twitter'),
32 ('reject',
33 'Reject this tweet and send it back to the author with your comment'),
34)
35 approval = forms.ChoiceField(
36 choices=APPROVAL_CHOICES, widget=forms.RadioSelect)
37
38 @permission_required('poster.can_approve_or_reject_tweet',
39 login_url='/login')
40 def review_tweet(request, tweet_id):
41 reviewed_tweet = get_object_or_404(Tweet, id=tweet_id)
42 if request.method == 'POST':
43 form = ReviewForm(request.POST)
44 if form.is_valid():

```



```
45 new_comment = form.cleaned_data['new_comment']
46 if form.cleaned_data['approval'] == 'approve':
47 publish_tweet(reviewed_tweet)
48 send_approval_email(reviewed_tweet, new_comment)
49 reviewed_tweet.published_at = datetime.now()
50 reviewed_tweet.state = 'published'
51 else:
52 link = request.build_absolute_uri(
53 reverse(post_tweet, args=[reviewed_tweet.id]))
54 send_rejection_email(reviewed_tweet, new_comment,
55 link)
56 reviewed_tweet.state = 'rejected'
57 reviewed_tweet.delete()
58 if new_comment:
59 c = Comment(tweet=reviewed_tweet, text=new_comment)
60 c.save()
61 return HttpResponseRedirect('/approve/')
62 else:
63 form = ReviewForm()
64 return direct_to_template(request, 'review_tweet.html', {
65 'form': form, 'tweet': reviewed_tweet,
66 'comments': reviewed_tweet.comment_set.all()})
67
68 def send_approval_email(tweet, new_comment):
69 body = ['Your tweet (%r) was approved & published on Twitter.' \
70 % tweet.text]
71 if new_comment:
72 body.append(
73 'The reviewer gave this feedback: %r.' % new_comment)
74 send_mail('Tweet published', '%s\r\n' % ' '.join(
75 body), settings.DEFAULT_FROM_EMAIL, [tweet.author_email])
76
77 def send_rejection_email(tweet, new_comment, link):
78 body = ['Your tweet (%r) was rejected.' % tweet.text]
79 if new_comment:
80 body.append(
81 'The reviewer gave this feedback: %r.' % new_comment)
82 body.append('To edit your proposed tweet, go to %s.' % link)
83 send_mail('Tweet rejected', '%s\r\n' % ' '.join(
84 body), settings.DEFAULT_FROM_EMAIL, [tweet.author_email])
85
86 def publish_tweet(tweet):
87 twitter = Twython(
88 twitter_token=settings.TWITTER_CONSUMER_KEY,
89 twitter_secret=settings.TWITTER_CONSUMER_SECRET,
90 oauth_token=settings.TWITTER_OAUTH_TOKEN,
91 oauth_token_secret=settings.TWITTER_OAUTH_TOKEN_SECRET,
92)
93 twitter.updateStatus(status=tweet.text.encode("utf-8"))
```

---

## Построчное объяснение

### Строки 1–24

Вслед за инструкциями импорта находится определение первого метода — `list_tweet()`. Этот метод обеспечивает формирование для пользователя списка

ожидающих обработки и опубликованных твитов. Непосредственно над заголовком этого метода находится декоратор `@permission_required`. Он служит для Django указанием, что доступ к этому методу должен предоставляться только зарегистрированным пользователям с разрешением `poster.can_approve_or_reject_tweet`. Это определяемое пользователем разрешение, которое объявлено в файле `myproject/poster/models.py`. Незарегистрированные пользователи или пользователи, прошедшие регистрацию, но не имеющие требуемых разрешений, отправляются в каталог `/login`. (Общие сведения о декораторах приведены в главе "Функции" книг *Core Python Programming* и *Core Python Language Fundamentals*.)

Если пользователь имеет надлежащее разрешение, метод вызывается на выполнение. В нем используются функциональные средства доступа к данным платформы Django для формирования списка всех твитов, ожидающих утверждения, и списка всех опубликованных твитов. Затем метод передает два указанных списка в шаблон `list_tweets.html`, с помощью которого эти результаты подготавливаются для отображения. Дополнительные сведения об этом файле шаблона приведены ниже.

## Строки 26–36

Затем необходимо отметить определение `ReviewForm`, приведенное в сценарии `myproject/approver/views.py`. Платформа Django предоставляет два способа задания формы. В сценарии `myproject/poster/views.py` форма `TweetForm` была определена на основе компонента `Tweet`. А в рассматриваемом примере вместо этого форма определена как коллекция полей, в основе которой не лежит какой-либо информационный объект. Форма предназначена для использования руководителями, которые утверждают или отклоняют твиты, направленные им для просмотра. При этом отсутствует какой-либо информационный объект, который представлял бы решение, принятое после просмотра. В форме используется коллекция, обеспечивающая выбор, которая определяет варианты, согласно которым ответственный за рецензирование утверждает или отклоняет твиты. Эти варианты представлены в виде списка переключателей.

## Строки 38–66

Далее следует определение метода `review_tweet()` (работа которого иллюстрируется на рис. 11.31). Этот метод напоминает метод обработки формы, представленный в форме `myproject/poster/views.py`, но в нем предполагается, что значение `tweet_id` всегда определено. Пример использования, в котором мог быть представлен на рассмотрение несуществующий твит, отсутствует.

Работа кода основана на том, что в нем происходит чтение реальных данных, отправленных пользователем в форме. Платформа Django позволяет удовлетворить это требование с использованием массива `form.cleaned_data[]`, который содержит значения, передаваемые пользователем с помощью формы, после преобразования в типы данных Python.

Заслуживает внимания то, как вызывается метод `build_absolute_uri()` применительно к объекту `request` в функции просмотра `review_tweet()`. Вызов этого метода осуществляется для получения ссылки на форму редактирования твита. Эта ссылка передается автору твита в письме по электронной почте с сообщением, что его твит отклонен. Таким образом, автор получает возможность рассмотреть отзыв руководителя и внести в твит исправления. Метод `build_absolute_uri()` возвращает URL, соответствующий конкретному методу, в данном случае, `post_tweet()`.

В соответствии с принятым соглашением этот URL имеет форму `/poster/edit/X`, где `X` — идентификатор твита. Рассмотрим, почему нельзя просто использовать строку, содержащую этот URL.

Дело в том, что после принятия решения о формировании URL в форме `/poster/change/X` необходимо обратиться ко всем сценариям, в которых применяется жестко заданный шаблон URL `/poster/edit/X`, и внести изменения, связанные с переходом к новому URL. Это противоречит принципу DRY, на который опирается платформа Django. Дополнительные сведения о том, что такое DRY и какие еще существуют принципы проектирования Django, см. в разделе <http://docs.djangoproject.com/en/dev/misc/design-philosophies>.



Рис. 11.31. Обработка кода в методе `review_tweet()`

Описанная выше ситуация отличается от той, в которой мог бы применяться жестко заданный URL, не включающий каких-либо переменных компонентов, как в разделе `/post/thankyou`, имеющем следующие отличительные особенности: во-первых, количество URL ограничено; во-вторых, маловероятно, что в URL придется вносить изменения; в-третьих, нет необходимости связывать с URL функции представления. В данной ситуации для упрощения работы по переходу к динамически формируемым URL вместо жестко закодированных можно применить еще один инструмент, `django.core.urlresolvers.reverse()`. Рассмотрим, как он работает. Итак, обычно следует начинать с URL и отыскивать функцию представления, с помощью которой запрос планируется на выполнение. В данном случае известно, какая функция представления должна применяться, а по ней необходимо сформировать URL. Имя данного инструмента отражает указанную последовательность действий. Функция представления передается со всеми параметрами функции `reverse()`, после чего происходит возврат URL. Еще один пример использования функции `reverse()` можно найти в учебнике по Django по адресу <https://docs.djangoproject.com/en/dev/intro/tutorial04/#write-a-simple-form>.

## Строки 68–84

Автору твита с помощью одного из двух вспомогательных методов, `send_approval_email()` и `send_rejection_email()`, передается электронная почта, для чего используется функция `send_mail()` Django. Еще раз отметим, что вызовы этих методов необходимо удалить из функции `review_tweet()`, если данный пример выполняется без доступа к почтовому серверу.

## Строки 86–93

Метод `publish_tweet()` также является вспомогательным. Он вызывает метод `updateStatus()`, который определен в пакете `Twython`, для публикации нового твита в `Twitter`. Заслуживает внимания то, что в нем используются четыре набора учетных данных `Twitter`, добавленных ранее в файл `settings.py`. Кроме того, следует отметить, что предусматривается кодирование твита с помощью символьной кодировки `UTF-8`, поскольку именно в этом состоит рекомендуемый способ организации работы `Twitter`.

Теперь перейдем к рассмотрению файлов шаблонов. Вначале рассмотрим страницу с информацией о состоянии, а затем страницу регистрации, поскольку первая, безусловно, намного интереснее, чем последняя. В примере 11.18 показан шаблон, используемый для страницы с информацией о состоянии. Выходная страница, как таковая, с точки зрения пользователя имеет два основных раздела: множество публикаций, ожидающих утверждения, с одной стороны, а также твиты, которые были утверждены и опубликованы.

### Пример 11.18. Шаблон, используемый для отображения состояния публикации (`list_tweets.html`)

Шаблон для страницы с информацией о состоянии приложения `poster` состоит из двух основных разделов: публикации, ожидающие утверждения, и опубликованные публикации.

```

1 <html>
2 <head>
3 <title>
4 Pending and published tweets
5 </title>
6 <style type=text/css>
7 tr.evenrow {
8 background: #FFFFFF;
9 }
10 tr.oddrow {
11 background: #DDDDDD;
12 }
13 </style>
14 </head>
15 <table>
16 <tr>
17 <td colspan=2 align=center>
18 Pending tweets
19 </td>
20 </tr>
21 <tr>
22 <td>
```

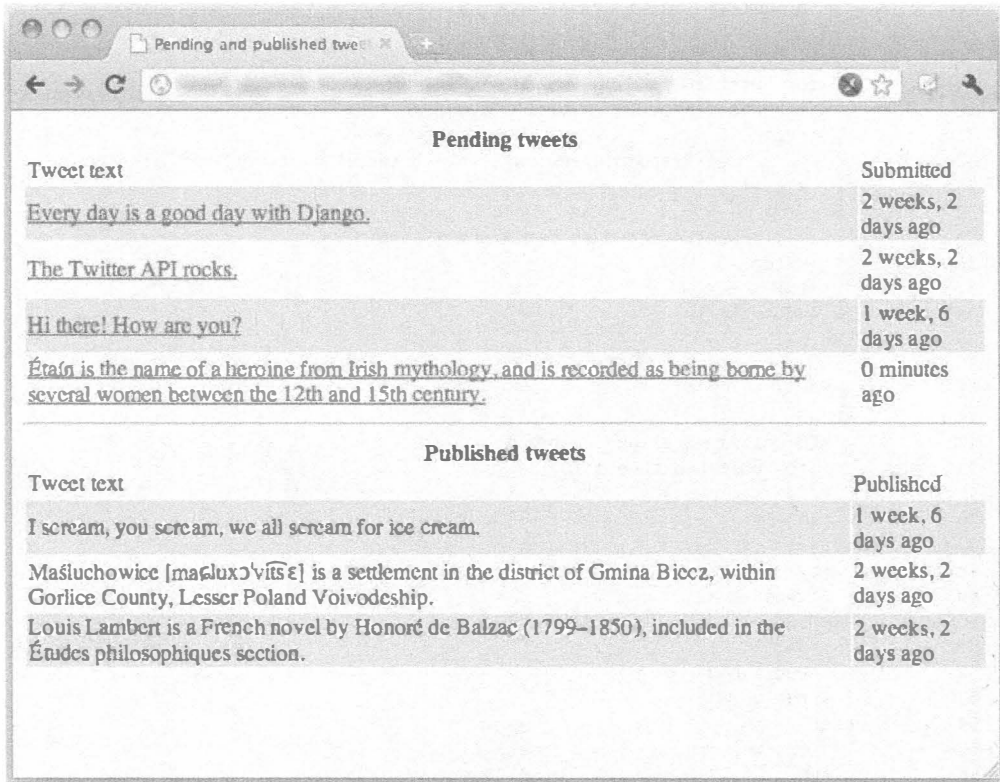
```

23 Tweet text
24 </td>
25 <td>
26 Submitted
27 </td>
28 </tr>
29 {% for tweet in pending_tweets %}
30 <tr class="{% cycle 'oddrow' 'evenrow' %}">
31 <td>
32 {{ tweet.text }}
33 </td>
34 <td>
35 {{ tweet.created_at|timesince }} ago
36 </td>
37 </tr>
38 {% endfor %}
39 </table>
40 <hr>
41 <table>
42 <tr>
43 <td colspan=2 align=center>
44 Published tweets
45 </td>
46 </tr>
47 <tr>
48 <td>
49 Tweet text
50 </td>
51 <td>
52 Published
53 </td>
54 </tr>
55 {% for tweet in published_tweets %}
56 <tr class="{% cycle 'oddrow' 'evenrow' %}">
57 <td>
58 {{ tweet.text }}
59 </td>
60 <td>
61 {{ tweet.published_at|timesince }} ago
62 </td>
63 </tr>
64 {% endfor %}
65 </table>
66 </html>

```

Любопытной особенностью этого шаблона является то, что это первый из рассматриваемых шаблонов, который содержит цикл. В нем происходит перебор коллекции `pending_tweets`, а затем `published_tweets`. После этого для каждого твита формируется строка таблицы с использованием конструкции `cycle`, которая предусматривает формирование серого фона в каждой четной строке, как показано на рис. 11.32. Кроме того, для каждого твита, который не был утвержден, подготавливается ссылка на страницу `/approve/review/X`, где `X` — идентификатор твита. Наконец, в шаблоне используется фильтр `timesince` Django для формирования поля с указанием времени, прошедшего после создания твита. Это гораздо удобнее по сравнению с вариантом, который показывал бы просто текущую дату и время. Тем самым

список становится более удобным для восприятия и позволяет легче ориентироваться во времени пользователям, разбросанным по разным часовым поясам.



**Рис. 11.32.** Список подлежащих обработке и опубликованных твитов

После того как руководитель, принимающий решение о том, следует ли утвердить для отправки или отвергнуть тот или иной твит, выбирает одну из публикаций, эта публикация открывается перед ним отдельно, как показано на рис. 11.33.

Шаблон `review_tweet.html`, с помощью которого разворачивается страница для рассмотрения твита, представлен в примере 11.19.

### Пример 11.19. Файл myproject/templates/review\_tweet.html

### Шаблон страницы проверки твита для приложения poster.

[illegible]

```

11 </td>
12 </tr>
13 <tr>
14 <td>
15 Author:
16 </td>
17 <td>
18 {{ tweet.author_email }}
19 </td>
20 </tr>
21 {{ form.as_table }}
22 </table>
23 <input type="submit" value="Submit" />
24 </form>
25 <hr>
26 History
27 <hr>
28 {% for comment in comments %}
29 <i>{{ comment.created_at|timesince }} ago:</i>
30 {{ comment.text }}
31 <hr>
32 {% endfor %}
33 </body>
34 </html>

```

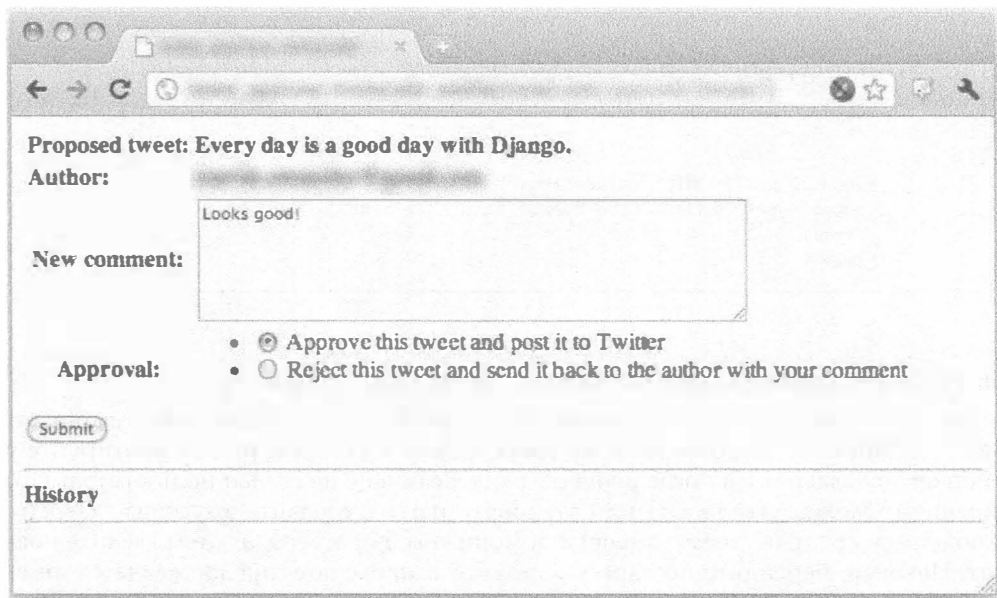


Рис. 11.33. Утверждение твита, подлежащего обработке

Как уже было сказано, если пользователь не зарегистрировался или не имеет соответствующего разрешения, то в его браузер передается URL `login/`. Рассмотрим, что при этом происходит. В сценарии `myproject/urls.py` содержатся указания для платформы Django, согласно которым должен быть выполнен код метода `django.contrib.auth.views.login`, который предусмотрен в Django и предназначен для

работы с формами регистрации. Таким образом, при разработке приложения достаточно лишь подготовить шаблон `login.html`. В примере 11.20 представлен один из простых вариантов таких шаблонов, который используется в данном приложении. Для получения дополнительных сведений о системе проверки подлинности с помощью Django см. документацию по адресу <https://docs.djangoproject.com/en/dev/topics/auth/>.

#### Пример 11.20. Файл `myproject/templates/login.html`

Система проверки подлинности Django используется и в шаблоне страницы регистрации приложения `poster`.

```
1 <html>
2 {% if form.errors %}
3 Your username and password didn't match. Please try again.
4 {% endif %}
5
6 <form method="post"
7 action="{% url django.contrib.auth.views.login %}">
8 {% csrf_token %}
9 <table>
10 <tr>
11 <td>{{ form.username.label_tag }}</td>
12 <td>{{ form.username }}</td>
13 </tr>
14 <tr>
15 <td>{{ form.password.label_tag }}</td>
16 <td>{{ form.password }}</td>
17 </tr>
18 </table>
19
20 <input type="submit" value="login" />
21 <input type="hidden" name="next" value="{{ next }}" />
22 </form>
23 </html>
```

## В качестве образца можно взять TweetApprover

На этом подготовка всех необходимых компонентов приложения завершается. Снова перейдите к подготовленному ранее файлу `URLconf` и раскомментируйте все действия, добавленные в последнюю очередь. Если еще не создан пользователь с разрешением “Может утверждать или отклонять твиты”, сделайте это сейчас. Перейдите по адресу `/post` (в своем домене) с помощью веб-браузера, а затем введите новый твит. Наконец, перейдите по адресу `/approve` и проверьте, предоставляется ли возможность отклонить или утвердить твит. После утверждения твита перейдите на веб-сайт Twitter и проверьте, опубликован ли этот твит.

Полный исходный код этого проекта можно загрузить с веб-сайта данной книги по адресу <http://corepython.com>.



## 11.17. Ресурсы

В табл. 11.4 представлены различные ресурсы, которыми можно воспользоваться для справки, и проекты, описанные в данной главе.

**Таблица 11.4.** Дополнительные ресурсы по веб-платформе

Платформа	Адрес
Django	<a href="http://djangoproject.com">http://djangoproject.com</a>
Pyramid & Pylons	<a href="http://pylonsproject.org">http://pylonsproject.org</a>
TurboGears	<a href="http://turbogears.org">http://turbogears.org</a>
Pinax	<a href="http://pinaxproject.com">http://pinaxproject.com</a>
Веб-платформы Python	<a href="http://wiki.python.org/moin/WebFrameworks">http://wiki.python.org/moin/WebFrameworks</a>
Django-nonrel	<a href="http://www.allbuttonspressed.com">http://www.allbuttonspressed.com</a>
virtualenv	<a href="http://pypi.python.org/pypi/virtualenv">http://pypi.python.org/pypi/virtualenv</a>
Twitter Developers	<a href="http://dev.twitter.com">http://dev.twitter.com</a>
OAuth	<a href="http://oauth.net">http://oauth.net</a>

## 11.18. Заключение

В описанном приложении мы применили лишь малую часть средств Django. Язык Python открывает буквально бескрайние возможности разработки для Интернета. Весьма велико также количество учебных материалов, поэтому рекомендуем вначале прочитать превосходную документацию по Django, особенно учебник, представленный по адресу <http://docs.djangoproject.com/en/dev/intro/tutorial01>. Кроме того, можно начать с изучения приложений, предназначенных для многократного использования, которые входят в состав проекта Pinax.

К тому же немало пользы можно извлечь из углубленного описания данной платформы, приведенного в книге *Python Web Development with Django*. Не меньший интерес представляет изучение других мощных веб-платформ Python, таких как Pyramid, TurboGears, web2py, или в определенной степени упрощенных платформ, таких как Bottle, Flask и Tipyf. Еще одним направлением, с которого можно начать изучение этой тематики, являются облачные вычисления. В данной книге им посвящена глава 12.

## 11.19. Упражнения

### Веб-платформы

- 11.1. Краткий терминологический экскурс. Что означает CGI и WSGI?
- 11.2. Краткий терминологический экскурс. В чем заключается основной недостаток отдельно взятого интерфейса CGI и почему он больше не используется для создания веб-служб производственного назначения?
- 11.3. Краткий терминологический экскурс. Какие проблемы решает интерфейс WSGI?
- 11.4. Веб-платформы. В чем состоит назначение веб-платформы?
- 11.5. Веб-платформы. В разработках для Интернета на основе веб-платформ обычно используются подход на основе шаблона контроллера “модель — представление” (model-view controller — MVC). Опишите каждый из компонентов, применяемых в этом подходе.
- 11.6. Веб-платформы. Назовите несколько веб-платформ Python с полным стеком. Создайте простое приложение “Hello World” с использованием каждой из них. Опишите различия между этими платформами с точки зрения разработки и эксплуатации.
- 11.7. Веб-платформы. Изучите несколько имеющихся систем поддержки шаблонов на основе Python. Подготовьте текстовую или электронную таблицу, которая позволяет сравнить эти системы и найти между ними различия. Обязательно рассмотрите синтаксические особенности применения директив, которые охватывают (по меньшей мере) следующее: а) переменные отображения данных; б) вызов функций или методов; в) внедрение собственного кода Python; г) организация циклов; д) условные выражения if-elseif-else; е) наследование шаблонов.

### Django

- 11.8. Основные сведения. Где и когда была создана платформа Django? В чем состоят основные цели создания этой платформы?
- 11.9. Терминология. В чем заключается различие между проектом Django и приложением Django?
- 11.10. Терминология. На платформе Django вместо принципа MVC используется принцип “модель — шаблон — представление” (model-template-view — MTV). Проведите сравнение и определите различия между MTV и MVC.
- 11.11. Конфигурация. Какие объекты применяются разработчиками Django для создания настроек баз данных?
- 11.12. Конфигурация. Django может эксплуатироваться на основе следующего:
  - а) реляционные базы данных;
  - б) нереляционные базы данных;
  - в) на основе того и другого;
  - г) не применяется ни то ни другое.

**11.13. Конфигурация.** Перейдите по адресу <http://djangoproject.com>, затем загрузите и установите веб-платформу Django (и SQLite, если не используется ПК с операционной системой Windows, поскольку SQLite предоставляется по умолчанию, начиная с версии Python 2.5 для Windows).

- а)** Выполните сценарий `django-admin.py startproject helloworld` для запуска проекта, затем выполните команды `cd helloworld; python ./manage.py startapp hello`, чтобы запустить приложение.
- б)** Отредактируйте сценарий `helloworld/hello/views.py` и включите следующий код:

```
from django.http import HttpResponse
def index(request):
 return HttpResponse('Hello world!')
```

- в)** В сценарии `helloworld/settings.py` добавьте `'hello'` в переменную `INSTALLED_APPS` (в любой позиции этого кортежа).

- г)** В сценарии `helloworld/urls.py` замените закомментированную строку

```
(r'helloworld/', include('helloworld.foo.urls')),
```

следующей (раскомментированной) строкой:

```
(r'^$', 'hello.views.index'),
```

Вызовите на выполнение `python ./manage.py runserver` и перейдите по адресу <http://localhost:8000>, чтобы убедиться в том, что код работает и в браузере отображаются слова “Hello world!”. Внесите изменения, чтобы вывести вместо “Hello world!” какой-то другой текст.

**11.14. Конфигурация.** Что такое файл `URLconf` и где он обычно находится?

**11.15. Учебник.** Найдите учебник по Django и проработайте все его четыре части, начиная с адреса <http://docs.djangoproject.com/en/dev/intro/tutorial01>. **Предупреждение.** Не следует просто копировать приведенный в учебнике код. Предполагается, что читатель будет вносить свои изменения в приложение, чтобы оно отличалось от предложенного варианта, а также, по возможности, добавлять новые функциональные средства.

**11.16. Инструменты.** Что такое приложение `admin` платформы Django? Как разрешить его использование? В чем состоит назначение приложения `admin`?

**11.17. Инструменты.** Есть ли способ проверить код приложения, не вызывая `admin` или даже веб-сервер?

**11.18. Терминология.** Что означает CSRF и почему в Django предусмотрены механизмы безопасности для предотвращения таких попыток?

**11.19. Модели.** Назовите пять лучших типов моделей, которые вы намереваетесь использовать, и перечислите типы данных, обычно применяемых с этими моделями.

**11.20. Шаблоны.** Как определяется понятие тега в шаблонах Django? Кроме того, в чем состоит различие между тегом блока и тегом переменной? Как проще всего различить теги этих двух типов?

- 11.21. Шаблоны. Опишите, как реализуется наследование шаблонов при использовании Django.
- 11.22. Шаблоны. Дайте определение понятия фильтра в шаблонах Django.
- 11.23. Представления. Что такое универсальные представления? Для чего они могут потребоваться? Есть ли такие ситуации, в которых универсальные представления не являются предпочтительными?
- 11.24. Формы. Опишите формы в Django, как они организованы и как действуют в коде (от модели данных до шаблона HTML).
- 11.25. Формы. Опишите формы моделей и покажите их преимущества.

## Приложение блога Django

- 11.26. Шаблоны. В шаблоне `archive.html` приложения `BlogPost` происходит обработка в цикле каждого почтового отправления и отображение его для пользователя. Добавьте проверку частного случая, состоящего в том, что еще не добавлены публикации и в связи с этим должно быть отображено специальное сообщение.
- 11.27. Модели. В рассматриваемом приложении все еще выполняется слишком много лишней работы при обработке отметок времени. Предусмотрена возможность дать Django такое указание, чтобы отметка времени добавлялась автоматически после создания объекта `BlogPost`. Узнайте, что такое автоматическое добавление отметок времени, и внесите необходимые изменения для того, чтобы это происходило, затем удалите код явного задания отметок времени из `blog.views.create_blogpost()` и `blog.tests.BlogPostTest.test_obj_create()`. Если ли также необходимость внести аналогичные изменения в сценарий `blog.tests.BlogPostTest.test_post_create()`? **Подсказка.** В некоторых примерах данной главы можно видеть, как эти действия выполняются в коде Google App Engine.
- 11.28. Универсальные представления. Исключите из использования как устаревшую функцию представления `archive()` и применяемую в ней функцию `render_to_response()`, а также обновите приложение, чтобы в нем использовалось универсальное представление. Для этого достаточно полностью удалить функцию `archive()` из сценария `blog/views.py`, а также переместить шаблон `blog/templates/archive.html` в шаблон `blog/templates/blogpost/blogpost_list.html`. Подробно изучите, как работает универсальный шаблон `list_detail.object_list()`, и предусмотрите его вызов непосредственно из файла `URLconf` рассматриваемого приложения. Необходимо создать словарь с применением `queryset`, а также `extra_context` для передачи автоматически сформированного объекта `BlogPostForm()` вместе со всеми записями блога в шаблон с помощью универсального представления.
- 11.29. Шаблоны. Вводное описание шаблонов фильтра Django приведено выше (и представлен пример с использованием функции `upper()`). Воспользуйтесь шаблоном `archive.html` (или `blogpost_list.html`) для рассматриваемого приложения `BlogPost` и добавьте еще одну строку для отображения

общего количества публикаций блога в базе данных с использованием фильтра перед показом десяти записей, полученных в последнюю очередь.

- 11.30. Формы.** После перехода к автоматическому созданию объекта формы с использованием `ModelForm` мы потеряли возможность указывать атрибуты строк и столбцов в элементе отображения текста `textarea` (количество строк равно 3, количество столбцов — 60), как было в случае применения только `Form` и определения графического элемента `HTML` для `forms.CharField`. Вместо этого произошел переход к использованию по умолчанию 10 строк и 40 столбцов, как показано на рис. 11.20. Как задать 3 строки и 60 столбцов? **Подсказка.** См. документы по адресу <http://docs.djangoproject.com/en/dev/topics/forms/modelforms/#overriding-the-default-field-types-or-widgets>
- 11.31. Шаблоны.** Создайте базовый шаблон для рассматриваемого приложения блога и измените все существующие шаблоны для применения наследования шаблонов.
- 11.32. Шаблоны.** Прочитайте в документации Django об использовании статических файлов (`HTML`, `CSS`, `JS` и т.д.) и улучшите внешний вид рассматриваемого приложения блога. Если в начале этой работы вы сталкиваетесь с затруднениями, опробуйте эти небольшие настройки, после чего сможете перейти к более конструктивным изменениям:
- ```
<style type="text/css">
body { color: #efd; background: #453; padding: 0 5em; margin: 0 }
h1 { padding: 2em 1em; background: #675 }
h2 { color: #bf8; border-top: 1px dotted #fff; margin-top: 2em }
p { margin: 1em 0 }
</style>
```
- 11.33. CRUD.** Предоставьте пользователям возможность редактировать и удалять публикации. Можно предусмотреть введение дополнительного поля с отметкой времени редактирования, если желательно, чтобы существующая отметка времени по-прежнему представляла время создания. В противном случае измените существующую отметку времени, чтобы она указывала, когда произошло редактирование или удаление почты.
- 11.34. Курсоры и разбиение на страницы.** Неплохо, когда отображаются десять последних публикаций, но еще лучше предоставить пользователям возможность также просматривать постранично полученную ранее почту. Введите в состав приложения курсоры и разбиение на страницы.
- 11.35. Кеширование.** В блоге Google App Engine предусмотрено использование `Memcache` для кеширования объектов, что позволяет избежать необходимости неоднократно обращаться к хранилищу данных для получения ответов на аналогичные запросы. Нужно ли вводить эту функцию в рассматриваемое приложение Django? Почему его следует (или не следует) использовать?
- 11.36. Пользователи.** Поддерживает несколько блогов на конкретном узле. Каждый отдельный пользователь должен получить в свое распоряжение набор страниц блога.

- 11.37. Связь. Добавьте в приложение еще одну функцию — передачу администратору веб-сайта и владельцу блога сообщения по электронной почте после создания каждой новой записи блога.
- 11.38. Бизнес-логика. В дополнение к предыдущему упражнению, предусматривающему отправку сообщения по электронной почте, возьмите любую страницу из приложения Twitter и обеспечьте получение утверждения от администратора записи блога до ее фактической отправки в блог.

Приложение Twitter Django

- 11.39. Шаблоны. Для исключения жестко заданных URL, применяемых вне файлов конфигурации URL, использовался метод `build_absolute_uri()`. Однако в шаблонах HTML еще остаются некоторые жестко закодированные пути URL. Где они находятся? **Подсказка.** Прочитайте документацию по адресу <http://docs.djangoproject.com/en/dev/ref/templates/builtins/#std:templatetag-url>.
- 11.40. Шаблоны. Придайте приложению TweetApprover более привлекательный вид. Для этого добавьте файл CSS и ссылайтесь на него из шаблонов HTML.
- 11.41. Пользователи. В настоящее время любой пользователь может отправить новые твиты, не входя в систему. Внесите в приложение изменения, чтобы пользователи не могли отправлять новые твиты, не входя в систему и не имея разрешения “добавлять твиты” в своей учетной записи пользователя.
- 11.42. Пользователи. После того как вы добьетесь, чтобы пользователи регистрировались, прежде чем предлагать новые твиты, обеспечьте предварительное заполнение поля электронной почты “Автор” с указанием адреса электронной почты зарегистрированного пользователя, если таковой имеется в профиле пользователя. **Подсказка.** Прочитайте документацию по адресу <http://docs.djangoproject.com/en/1.2/topics/auth>.
- 11.43. Кеширование. Предусмотрите кеширование списка твитов, отображаемого, когда пользователь переходит в каталог `/approve`. После того как пользователь утвердит или отклонит твит, происходит его переход снова на эту страницу; обеспечьте, чтобы к завершению этого перехода отображалась свежая, не кешированная версия страницы.
- 11.44. Ведение журналов и формирование отчетов. Создайте контрольный журнал для твитов с добавлением новых полей с комментариями, `Comments`, в почту после каждого изменения состояния. Например, если твит отклонен, добавляйте к нему комментарий `Comment` с указанием, что он отклонен в такое-то время. После каждой корректировки текста добавляйте еще один комментарий `Comment`. После каждой публикации также добавляйте комментарий `Comment` с указанием, когда произошла публикация и кто ее утвердил.
- 11.45. CRUD. Предусмотрите на странице рецензирования твита еще один вариант, чтобы руководитель мог не только принимать или отклонять отправленный твит, но и удалять его. Можно удалить объект из базы данных, вызвав применительно к нему метод `delete()`, примерно так: `reviewed_tweet.delete()`

- 11.46.** Связь. После того как сотрудник предлагает новый твит, руководителю отправляется письмо по электронной почте. В этом письме указано лишь то, что имеется новый твит, подлежащий утверждению. Обеспечьте, чтобы это электронное письмо было более удобным, добавив к нему текст нового твита и ссылку, на которой руководитель может щелкнуть, чтобы перейти непосредственно на веб-страницу для утверждения или отклонения твита. Можно сравнить этот вариант с тем, как организована отправка электронной почты в сценарии `myproject/approver/views.py`.

Облачные вычисления: Google App Engine

В этой главе...

- Введение
- Что такое облачные вычисления
- “Песочница” и набор SDK App Engine
- Выбор платформы для системы App Engine
- Поддержка версии Python 2.7
- Сравнение с платформой Django
- Преобразование приложения Hello World в простой блог
- Добавление службы Memcache
- Статические файлы
- Добавление службы Users
- Оболочка удаленного API
- Быстрый обзор (с использованием кода на языке Python)
- Мгновенная отправка сообщений с помощью службы XMPP

- Обработка изображений
- Очереди задач (незапланированные задачи)
- Профилирование с помощью Appstats
- Служба URLfetch
- Быстрый обзор (без использования кода Python)
- Обеспечение замкнутости поставщика
- Ресурсы

В настоящее время наша отрасль вступила в полосу важных изобретений, одним из которых является так называемое “облако”. Между тем никто толком не знает, что это такое и что именно обозначает этот термин
Стив Балмер, октябрь 2010 г.

12.1. Введение

Следующей системой разработки, которую нам предстоит изучить, является Google App Engine. Несмотря на то что App Engine не может служить полномасштабной платформой наподобие Django (хотя, как будет показано ниже в этой главе, у вас есть возможность пользоваться Django в системе App Engine), App Engine представляет собой платформу разработки, которая первоначально была ориентирована на веб-приложения (в ее состав входит ее собственная микроплатформа, webapp, или пришедшая ей на смену новая микроплатформа webapp2), но может использоваться — и реально используется — также для создания приложений и служб общего назначения.

Говоря о приложениях общего назначения, мы не имеем в виду, будто любое приложение может быть создано для платформы App Engine или перенесено на нее; скорее мы имеем в виду сетевые приложения, для обращения к которым требуется лишь конечная точка HTTP. Это относится к веб-приложениям, но не ограничивается лишь ими. Одним популярным вариантом использования, не имеющим отношения к веб, является серверная служба для мобильных клиентов, обеспечивающих непосредственное взаимодействие с пользователем. App Engine принадлежит к категории систем для облачных вычислений, обеспечивающих платформу для разработчиков, на которой можно было бы создавать и выполнять приложения или серверные приложения. Прежде чем мы приступим к непосредственному рассмотрению деталей этой платформы, необходимо дать общее представление об экосфере облачных вычислений, чтобы читатели могли лучше уяснить место, занимаемое App Engine в этой картине.

12.2. Что такое облачные вычисления

В то время как приложения Django, Pyramid или Turbogears предоставляются вашим провайдером или имеются на ваших собственных компьютерах, приложения Google App Engine принадлежат Google и являются частью более широкого класса служб, объединенных под “зонтиком” облачных вычислений. Главным исходным условием этих служб является то, что их пользователи заимствуют на стороне част

вычислительной инфраструктуры какой-либо компании (или частного лица), идет ли речь о фактическом оборудовании, разработке и выполнении приложений или хостинге программного обеспечения. Если вы используете облачные вычисления, то делегируете вычисления, хостинг и/или обслуживание вашего приложения какому-либо корпоративному субъекту, отличному от вашего собственного.

Такого рода службы имеются лишь в Интернете, а их точное физическое местоположение может быть как известно, так и неизвестно. К этим службам относятся все, начиная с аппаратного обеспечения¹ и кончая приложениями. Они также обеспечивают работу операционных систем, баз данных, файловых систем и дисковой памяти, вычислительных устройств, систем обмена сообщениями, электронной почты, систем мгновенного обмена сообщениями, виртуальных машин, систем кеширования (многоуровневого, начиная с сервера Memcached и кончая сетями доставки контента (content delivery network — CDN)) и т.п. Эта отрасль отличается высоким уровнем активности: провайдеры постоянно добавляют все новые и новые службы. Оплата за пользование этими службами осуществляется либо на основе подписки, либо на временном принципе (в зависимости от времени пользования).

Одной из основных причин, объясняющих, почему компании используют службы облачных вычислений, обычно являются издержки. Однако потребности разных компаний настолько различаются между собой, что каждой компании необходимо выполнить собственное исследование, чтобы определить, следует ли пользоваться службами облачных вычислений. Вы являетесь владельцем только что созданной компании и не можете позволить себе покупку всего необходимого вам оборудования (и не желаете арендовать центр обработки и хранения данных или оборудование компании, расположенной где-то неподалеку от вас)? Нет проблем, арендуйте один или тысячу компьютеров у компании Amazon или воспользуйтесь диском сверхбольшой емкости от компании Google. Остались в прошлом времена, когда основателям небольших компаний приходилось начинать свою деятельность, полагаясь лишь на собственные силы и инвестируя немалые средства в указанную инфраструктуру (как правило, за счет банковских ссуд). В наши дни они могут полностью сосредоточиться на своих приложениях и задачах, которые необходимо решить.

Та же ситуация выглядит несколько по-другому, если взглянуть на крупные предприятия или компании, входящие в перечень Fortune 500, которые располагают вполне достаточными финансовыми возможностями, чтобы купить все необходимое им оборудование, но понимают, что не смогут использовать его с максимальной эффективностью. От вас не требуется создавать такой облачный бизнес, как у компании Amazon (подробнее об этом — в следующем разделе), но вы вполне можете создать собственное, или *частное*, облако для предоставления облачных служб в рамках своей компании; возможно, вы посчитаете целесообразным создать *гибридное облако* и часть своей инфраструктуры у себя в компании (возможно, это будет та часть, которая обрабатывает самые важные для вас данные), а другие части (вычисления, приложения, хранение и т.п.) передать в *облако общего пользования*, например Google или Amazon.

Фирмам, которые задействуют облачные службы, зачастую приходится принимать во внимание такие факторы, как физическое местоположение, безопасность, соглашение об уровне услуг (Service Level Agreement — SLA) и совместимость; в зависимости от отрасли, в которой они работают, или своей основной юрисдикции

¹ Термин “аппаратное обеспечение” означает физические устройства (которые могут также включать дисковую и оперативную память), а также системы обеспечения электропитания, воздушного охлаждения и сетевого соединения.

они могут быть вынуждены поступать так. Очевидно, когда компаниям приходится прибегать к аутсорсингу приложений, данных и т.п., им необходимы гарантии того, что их интеллектуальная собственность защищена, находится там, где это разрешено управляющими органами фирмы (если они есть), и доступ к этим ресурсам возможен в любое время. Если эти требования соблюдаются, фирма должна определить подходящие для себя уровни облачных вычислений.

12.2.1. Уровни службы облачных вычислений

Облачные вычисления возможны на трех уровнях службы. На рис. 12.1 представлен вид каждого уровня службы, а также некоторые репрезентативные продукты на каждом соответствующем уровне. Самый нижний уровень, известный как *инфраструктура как служба* (Infrastructure-as-a-Service — IaaS), обеспечивает вычислительные мощности как таковые: сами по себе компьютеры (физические или виртуальные), внешнюю память (обычно дисковую), а также вычисления. Amazon Web Services (AWS) обеспечивает свои службы Elastic Compute Cloud (EC2) и Simple Storage System (S3) на уровне IaaS. Google также обеспечивает службу хранения IaaS под названием Google Cloud Storage.

Система Google App Engine действует на среднем уровне облачных вычислений, известном как *платформа как служба* (Platform-as-a-Service — PaaS). Этот уровень предоставляет пользователям платформу выполнения для их приложений. Самым высоким уровнем является *программное обеспечение как служба* (Software-as-a-Service — SaaS). На этом уровне пользователи просто обращаются к приложениям, которые являются собственностью данной среды и к которым можно обратиться лишь посредством Интернета. В качестве примера SaaS можно привести службы электронной почты, базирующиеся в веб (например, Gmail, Yahoo! Mail и Hotmail).

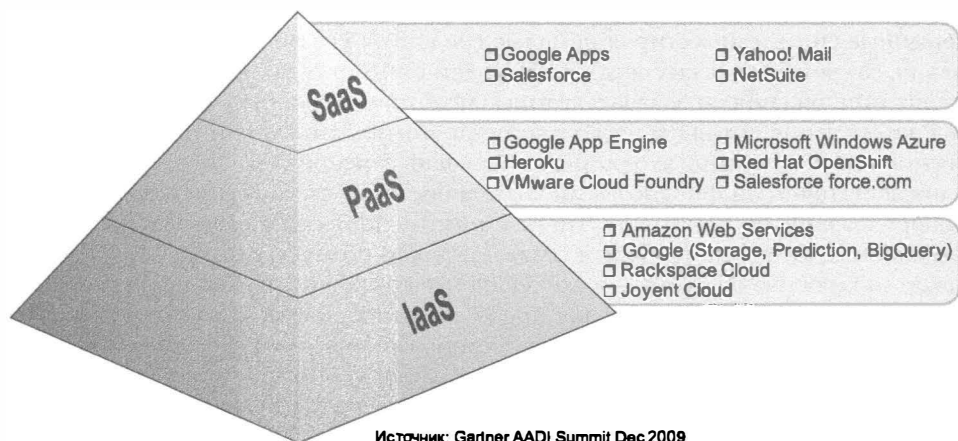


Рис. 12.1. Три уровня службы облачных вычислений

Из этих трех уровней IaaS и SaaS являются самыми известными, тогда как уровень PaaS не купается в лучах славы так часто, как его “собратья”. Однако, возможно, ситуация изменится, поскольку именно уровень PaaS является, наверное, самым влиятельным из указанных трех уровней. Вместе с PaaS вы получаете IaaS бесплатно, но он включает много чрезвычайно дорогостоящих служб, которые к тому же являются чрезвычайно громоздкими, чтобы можно было самостоятельно управляться с ними.

К ним может относиться все, что находится на уровне IaaS и за его пределами, например: операционная система, базы данных, лицензирование программного обеспечения, сети и балансирование нагрузки, серверы (веб и прочее), программные “заплаты” и апгрейды, мониторинг, предупреждение и оповещение, устранение “прорех” в системе безопасности, системное администрирование и т.п. Ключевой выгодой от использования этого уровня службы в отличие от использования своего собственного оборудования является отсутствие простоев производственных мощностей, обусловленных приобретением больших вычислительных мощностей, чем в действительности необходимо (возможно, вы инвестировали средства в приобретение этих вычислительных мощностей исходя из первоначального прогноза веб-трафика). Ужасно осознавать, что сделанные вами инвестиции (к тому же весьма внушительные) используются недостаточно эффективно и не окупают себя.

Несмотря на то что концепция облачных вычислений насчитывает уже не один десяток лет — Джон Гейдж из Sun Microsystems придумал свой запоминающийся слоган, *The Network is the Computer* (“Сеть — это компьютер”), еще в 1984 году, — эта концепция получила свое практическое (и выгодное с коммерческой точки зрения) воплощение лишь в середине “нулевых” годов, точнее говоря, в начале 2006 года, когда компания Amazon объявила о создании инфраструктуры AWS. Именно необходимость срочного решения проблемы простоев производственных мощностей подвигла Amazon к созданию AWS. Чтобы бизнес розничной торговли Amazon мог справиться с трафиком и потребностями сезона праздничных распродаж в режиме онлайн, этой компании пришлось закупить достаточный объем вычислительных ресурсов.

Как пишет компания Amazon в своей “белой книге”², “к 2005 г. мы потратили более десяти лет и сотни миллионов долларов на создание и эксплуатацию крупномасштабной, надежной и эффективной ИТ-инфраструктуры, которая обеспечивает функционирование одной из крупнейших в мире платформ розничной торговли в режиме онлайн”.

Однако нетрудно представить, чем были заняты эти внушительные вычислительные мощности в остальное время года, когда интенсивность онлайн-торговли не столь высока. Если называть вещи своими именами, то большую часть времени эти вычислительные мощности простаивали. У руководства компании Amazon возник естественный вопрос: почему бы не сдавать в аренду эти простаивающие ЦП и дисковую память, как обычно поступают любые компании, располагающие избыточным оборудованием. Именно так они и поступили. Впоследствии этот почин подхватило несколько других крупных высокотехнологичных компаний: Google, Salesforce, Microsoft, RackSpace, Joyent, VMware, а затем и многие, многие другие, вовремя присоединившиеся к набиравшей все большую популярность моде на облака.

В то время как службы Amazon EC2 и S3 четко сосредоточены на инфраструктурном уровне, новый рынок начал открываться для тех, кто хотел бы привлечь сторонние ресурсы для выполнения своих приложений, и, в частности, для тех, кто умеет разрабатывать собственные системы программного обеспечения, способные использовать корпоративные данные Salesforce (отражающие отношения с клиентами). Именно это подвигло компанию Salesforce на создание *force.com*, первой платформенной службы, предназначенной именно для такой цели. Разумеется, далеко не каждому требуется приложение Salesforce, написанное на еще одном патентованном языке программирования, поэтому Google разработала более универсальную PaaS-службу под названием App Engine, которая появилась в апреле 2008 г.

² http://media.amazonwebservices.com/AWS_Overview.pdf

12.2.2. Что такое App Engine

Почему мы рассказываем о системе App Engine в книге, посвященной Python? Является ли App Engine ядром этого языка или центральным пакетом какого-то стороннего разработчика? Несмотря на то что она не является ни тем ни другим, появление и существование этой службы оказало огромное влияние на сообщество пользователей Python и на соответствующий рынок в целом — вообще говоря, столь огромное, что я нисколько не сомневался в целесообразности включения в эту книгу отдельной главы, посвященной Google App Engine. (То же самое случилось с *Python Web Development with Django*, книгой, которую я написал в соавторстве со своими глубокоуважаемыми коллегами Джеффом Форсье (Jeff Forcier) и Полом Биссексом (Paul Bissex).)

В то время как веб-платформы имеют вполне ожидаемые сходства и различия, App Engine представляет собой весьма примечательное отступление от всех них — и не только потому, что оно является платформой разработки, но и потому, что комплектуется службами хостинга приложений. Именно поэтому вам даже *следовало бы* создавать приложения с помощью App Engine. Сейчас пользователи располагают гораздо более простой альтернативной разработке приложения и поиску подходящего места для его хостинга — или, хуже того, построению собственной инфраструктуры для поддержки своего приложения. Вся эта дополнительная работа предполагает гораздо больше, чем просто проектирование, кодирование и тестирование приложения.

Вместо того чтобы иметь дело с провайдером Интернета или самостоятельно решать проблему хостинга, разработчики выгружают свои приложения в систему Google, которая берет на себя заботы о всей логистике эксплуатации этих приложений в режиме онлайн. Обычный веб-разработчик пользуется теми же ресурсами, что и вся компания Google, выполняет свои приложения в тех же центрах обработки и хранения данных и работает на том же оборудовании, которыми пользуется сам этот интернет-гигант. Вообще говоря, с помощью App Engine и прочих своих облачных служб Google фактически предоставляет для общего доступа такой же интерфейс прикладного программирования (API), каким пользуются разработчики этой компании. Сюда относятся API App Engine, такие как Datastore (Megastore, Bigtable), Blobstore, Image (Picasa), Email (Gmail), Channel (GTalk) и т.п. Кроме того, в наши дни разработчику уже не приходится заботиться о компьютерах, операционных системах, организации работы в сетях, электропитании и охлаждении оборудования, балансировании нагрузки и т.п.

Все это, конечно, замечательно, но какое место в этой картине занимает язык Python?

Когда в 2008 г. служба App Engine была впервые запущена в действие, единственной поддерживаемой языковой исполняющей средой был Python. Язык Java появился годом позже, однако язык Python занимает особое место, поскольку имел языковую исполняющую среду, поддерживаемую платформой App Engine. Тем, кто сейчас занимается программированием на языке Python, уже известно, что этот чрезвычайно удобный в использовании язык облегчает коллективную разработку программного обеспечения, значительно ускоряет разработку программ и не требует наличия у своих пользователей специального компьютерного образования как необходимого условия эффективного использования этого инструмента. Такие качества языка Python способствуют его высокой популярности среди разработчиков, имеющих разный опыт работы и разные профессиональные пристрастия. Сам создатель языка

Python является инженером, работающим в команде разработчиков App Engine, не говоря уж о вашем покорном слуге. Учитывая новаторскую природу App Engine и его тесные связи с сообществом пользователей Python, я чувствую себя обязанным помочь вам в освоении этой службы!

Четыре основных компонента App Engine составляют всю систему этой службы: языковые исполняющие среды, расширяемая аппаратная инфраструктура, веб-консоль администрирования и набор инструментальных средств разработки программного обеспечения (SDK), предоставляющий в распоряжение пользователей необходимые им инструменты: сервер разработки и доступ к интерфейсам прикладного программирования App Engine.

Языковые исполняющие среды

Что касается языковых исполняющих сред, то все оставшееся время мы намерены посвятить языку Python (что вполне естественно), однако следует помнить, что во время написания этой книги в распоряжении разработчиков были также языки Java и Go. К тому же благодаря поддержке Java разработчики имеют возможность кодировать на языках, которые располагают подходящими интерпретаторами, способными выполняться на виртуальной машине Java (JVM). Речь идет о таких языках, как Ruby, PHP, JavaScript и Python, для которых используются интерпретаторы JRuby, Quercus, Rhino и Jython соответственно, плюс Scala и Groovy. Самым загадочным является, пожалуй, интерпретатор Jython: многие не понимают, зачем пользователям выполнять то или иное Jython-приложение, если они могли бы поступить гораздо проще — воспользоваться непосредственной поддержкой Python. Однако не следует забывать о пользователях, которые хотят разрабатывать новые проекты на Python, но при этом располагают пакетами Java. Неудивительно, что в таком случае они хотят воспользоваться уже имеющимися у них пакетами, но не хотят (или не могут позволить себе) переносить эти библиотеки в среду Python.

Аппаратная инфраструктура

По сути, аппаратная инфраструктура является для пользователей “черным ящиком”: как правило, пользователю мало что известно об оборудовании, на котором выполняется его код. У вас вполне может сложиться впечатление, что здесь пахнет системой Linux и что оборудование находится где-то в центрах обработки и хранения данных, подключенных к глобальной сети. Возможно, вам даже приходилось слышать о *Bigtable*, нереляционной СУБД, которую App Engine использует в качестве хранилища своих данных. Большинству людей вряд ли потребуется знать что-то помимо этого: помните, что в случае облачных вычислений это уже не является вашей заботой. Чрезвычайно трудная работа и подробности, касающиеся эксплуатации и обеспечения доступа пользователей к такой инфраструктуре и использования ее возможностей, остаются за кулисами и скрыты от глаз пользователей.

Веб-администрирование и состояние системы

В оставшихся разделах этой главы мы рассмотрим разные характеристики интерфейса прикладного программирования Python. Следует иметь в виду, что в ходе разработки ваши приложения не будут задействовать полные версии интерпретаторов Python (или Java). Поскольку ваше приложение использует ресурсы совместно с приложениями других пользователей, то, по соображениям безопасности, все

приложения должны выполняться в так называемой *песочнице*, т.е. некоей ограниченной среде. Разумеется, при этом вы утрачиваете какую-то степень контроля в обмен на расширяемость и предоставление вам компонентов, которые чрезвычайно трудно создавать самостоятельно.

Взамен App Engine предоставляет вам базирующуюся на веб консоль администрирования (называемую для краткости *админ-консолью*). Админ-консоль позволяет разработчикам проанализировать свое приложение, его трафик, данные, ход выполнения, начисление оплаты, настройки, эффективность использования, квоты и т.д. На рис. 12.2 показана копия экрана админ-консоли приложения.



Рис. 12.2. Консоль администрирования приложения Google App Engine. (Публикуется с разрешения компании Google.)

Имеется также страница состояния системы в целом (рис. 12.3), с помощью которой вы можете отслеживать функционирование App Engine в целом, по всем приложениям.

Учтите, что словосочетание “по всем приложениям” следует понимать буквально. По состоянию на зиму 2010 г. Google App Engine ежедневно обслуживает свыше одного миллиарда веб-страниц. После того как вы создадите и введете в эксплуатацию свое приложение, вы внесете свой вклад в количество веб-страниц, обслуживаемых Google App Engine. Несмотря на то что такое количество веб-страниц действительно производит впечатление, не нужно забывать, что, поскольку App Engine совместно используется всеми разработчиками, вы должны научиться правильно вести себя в “песочнице”. Работать в “песочнице” не так уж плохо, как может показаться на первый взгляд, поскольку App Engine предоставляет в распоряжение разработчиков много служб и API.

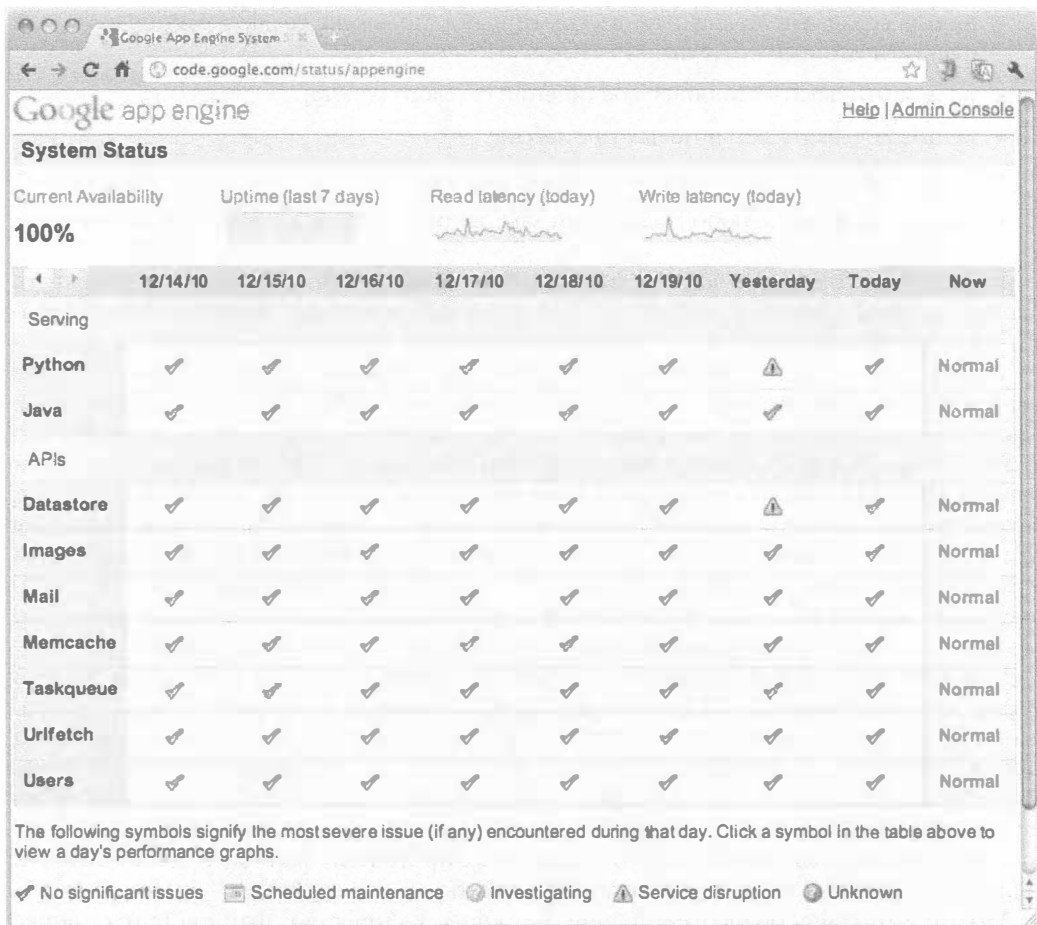


Рис. 12.3. Страница состояния системы для приложений Google App Engine. (Публикуется с разрешения компании Google.)

12.3. “Песочница” и набор SDK App Engine

Нет ничего удивительного в том, что разработчики не хотели бы, чтобы другие приложения имели возможность получать доступ к исходным кодам их собственных приложений и их данным. Таким образом, вы поступили бы честно, если бы уважали право на неприкосновенность приложений и связанных с ними данных, являющихся собственностью других разработчиков. Существуют определенные ограничения, которые распространяются на всех пользователей “песочницы” и которые не следует пытаться обойти. (Время от времени Google отменяет такие ограничения, если это не влечет за собой неприятных последствий.) К числу запрещенных действий относится следующее (приведенный ниже перечень запрещенных действий не является исчерпывающим):

- нельзя создавать локальный дисковый файл, но можно создать распределенный файл с помощью системы API Files;
- нельзя открывать входящее соединение сетевого сокета;
- нельзя создавать ветви новых процессов;
- нельзя делать вызовы (операционной) системы;
- нельзя выгружать любой исходный код, не относящийся к языку Python.

Вследствие этих ограничений в состав набора SDK App Engine входят API верхнего уровня, компенсирующие любую потерю функциональности, вызванную указанными ограничениями.

Кроме того, поскольку версия языка Python, исполняемая платформой App Engine (в настоящее время это версии 2.5 и 2.7), представляет собой определенное подмножество полного дистрибутива, у вас нет доступа ко всем возможностям среды Python и, в частности, к тем, которые скомпилированы на языке C. Некоторые из модулей и пакетов среды Python, скомпилированных на языке C, доступны пользователям. Однако версия 2.7 поддерживает значительно большее количество библиотек языка C, в том числе некоторые из широко известных внешних пакетов, таких как NumPy, lxml и PIL. В то время как поддержка библиотек C версией 2.5 реализована в форме “белого списка”, версия 2.7 обеспечила доступность настолько большего их количества, что этот список в настоящее время можно было бы назвать “черным списком”.

Библиотеки языка C, разрешенные (включенные в “белый список”) в версии Python 2.5 и запрещенные (включенные в “черный список”) в версии Python 2.7, описаны на сайте <http://code.google.com/appengine/kb/libraries.html> (аналогичный список существует и для классов Java). Если, однако, вы хотите использовать какие-либо Python-пакеты сторонних разработчиков, то можете без проблем связать их со своим исходным кодом, если в них не используется ничего, кроме Python (например, если они не содержат каких-либо исполняемых программ, файлов .so или .dll, и т.п.), и если в них не используются модули/пакеты, не включенные в “белый список”.

Учтите, что существует ограничение на общее количество файлов (в настоящее время 10 000), которые вы можете выгрузить, еще одно ограничение на суммарный размер всех выгруженных файлов (в настоящее время 150 Мбайт) — сюда входят файлы приложений или статические активы, такие как HTML, CSS, JavaScript и т.п., — а также ограничение на размер каждого файла (в настоящее время 32 Мбайт). Если вы хотите ознакомиться с ныне действующим перечнем ограничений на размеры, обратитесь на сайт http://code.google.com/appengine/docs/python/runtime.html#Quotas_and_Limits, поскольку команда разработчиков делает все зависящее от них для отмены ограничений там, где это возможно. Вместе с тем существует несколько способов, которые позволяют обойти эти ограничения.

Если ваше приложение обслуживает медиа-файлы, размер которых превышает ограничение на размер одного файла, сохраняйте их в хранилище App Engine Blobstore (табл. 12.1), где можно хранить файлы любых размеров, т.е. где отсутствует ограничение на размер каждого отдельно взятого файла (“блób”). Если вас волнует суммарное количество файлов с расширением .ру, то можете сохранять их в zip-файл и выгружать именно такой файл. Независимо от количества заархивированных вами файлов с расширением .ру, вы платите штраф только за один zip-файл. Разумеется, размер этого zip-файла также не должен превышать ограничение на размер одного файла, но в этом случае вам по крайней мере не нужно задумываться о количестве файлов. Подробнее об использовании zip-файлов можно прочитать

в статье, опубликованной на сайте <http://docs.djangoproject.com/en/dev/ref/settings> (обратите внимание на примечание, помещенное сверху этой статьи).

После того как я упомянул об ограничениях, можно вернуться к ограничениям на выполнение (не допускается использование сокетов, файлов, процессов или системных вызовов). Не имея возможности пользоваться этими “строительными блоками”, вряд ли вам удастся создать сколько-нибудь полезное приложение. Не торопитесь отчаиваться: этому делу можно помочь!

12.3.1. Службы и API

Для того чтобы помочь вам справиться со своей работой, Google предоставляет в ваше распоряжение все больше и больше “строительных блоков”, которые позволяяют компенсировать неудобства, связанные с перечисленными выше ограничениями. Например, *зачем* вам может понадобиться открыть сетевой сокет? Вы хотите обмениваться данными с другими серверами? В этом случае можно воспользоваться API URLfetch. А как насчет отправки и получения электронной почты? API Email был создан именно для этой цели. Аналогично API XMPP (eXtensible Messaging and Presence Protocol, или просто Jabber) можно использовать для мгновенной отправки и получения сообщений (instant messages, IM). То же самое касается доступа к вторичному сетевому кешу (API Memcache), задействования обратного AJAX или проталкивания в браузер (API Channel), доступа к базе данных (API Datastore) и т.д. В табл. 12.1 перечислены все службы и API, которыми могут пользоваться разработчики App Engine на момент написания этой книги.

Таблица 12.1. Службы и API Google App Engine (некоторые из них являются экспериментальными)

| Служба/API | Описание |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| App Identity | Используйте эту службу, когда ваше приложение содержит код, которому необходимо идентифицировать себя или другие API, которые требуют такой информации |
| Appstats | Базирующаяся на событиях микроплатформа, которая помогает оценить производительность вашего приложения |
| Backends | Если стандартные сроки завершения запроса/ответа или очереди задач не отвечают вашим потребностям (т.е. недостаточно велики), то можете использовать службу Backends для бесконечного выполнения кода App Engine |
| Blobstore | Пользуясь службой Blobstore, вы можете использовать приложения для обслуживания объектов данных (“блотов”), которые чересчур велики для Datastore (например, медиа-файлов) |
| Capabilities | Дает возможность приложениям обнаруживать факт недоступности Datastore или Memcache App Engine. Это позволяет предоставлять пользователям обслуживание в периоды простоя |
| Channel | Служба, с помощью которой ваше приложение может проталкивать данные непосредственно на браузер; известен также под названиями Reverse Ajax, проталкивание в браузер, Comet |
| Cloud SQL | Используйте реляционную базу данных (вместо предусмотренного по умолчанию расширяемого нераспределяемого хранилища данных) |
| Cloud Storage | Читайте или записывайте файлы непосредственно в службу Google Cloud Storage с помощью хорошо знакомого вам API Files (см. соответствующее описание ниже в этой таблице) |
| Conversion | Используйте эту службу для взаимных преобразований форматов HTML, PDF, а также текстового формата и формата изображений |

| Служба/API | Описание |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Cron | Служба Cron позволяет планировать решение задач на определенные моменты времени или через определенные интервалы времени |
| Datastore | Распределенное, расширяемое, нереляционное постоянное хранилище для ваших данных |
| Denial-of-Service | Используйте эту службу для создания фильтров, блокирующих IP-адреса/семейства, которые подвергают ваше приложение DoS-атакам (Denial of Service) |
| Download | При возникновении аварийной ситуации разработчики могут загрузить код, выгруженный ими в Google |
| Files | Создавайте распределенные (Blobstore или Cloud Storage) файлы с помощью общего файлового интерфейса Python |
| Search | Выполняйте поиск текста, меток даты/времени и т.п. в своем хранилище данных |
| Images | Работайте с данными изображений; например, создавайте миниатюрные изображения (пиктограммы), вырезайте, изменяйте масштаб и поворачивайте изображения |
| Logs | Дает возможность пользователям обращаться к приложению и запрашивать регистрацию — и даже выполнять сброс во время выполнения для запросов, выполняющихся достаточно долго |
| Mail | Этот API позволяет вашему приложению отправлять и/или принимать электронную почту |
| MapReduce | Использовался для выполнения распределенных вычислений по отношению к достаточно крупным наборам данных. Этот API включает фазы отображения (преобразования), перемешивания и сокращения |
| Matcher | Чрезвычайно расширяемая инфраструктура приведения в соответствие в реальном времени: регистрирует запросы на приведение в соответствие с потоком объектов |
| Memcache | Стандартный распределенный кеш данных в оперативной памяти (подобно Memcached) между вашим приложением и постоянной памятью |
| Namespaces (Multitenancy) | С помощью службы Namespaces можно создавать приложения, рассчитанные на многих участников, распределяя данные Google App Engine по разным "отсекам" |
| NDB (новая база данных) | Новый, экспериментальный интерфейс хранилища данных верхнего уровня Python-App Engine |
| OAuth | Обеспечивает посторонним лицам безопасный способ доступа к данным от имени определенного пользователя, не требуя при этом авторизации (логинов/паролей и т.п.) |
| OpenID | Объединенная служба идентификации, с помощью которой пользователи могут войти в систему с Google Accounts и учетных записей OpenID |
| Pipeline | Управляет одновременно многими задачами с длительным временем выполнения/потоками заданий и представляет в требуемом порядке их результаты |
| Prospective Search | В отличие (в какой-то мере) от API полнотекстового поиска, который дает возможность пользователям отыскивать существующие данные, Prospective Search позволяет пользователям запрашивать данные, которые еще не созданы: сформируйте свои запросы и, когда искомые данные будут помещены в память, выполняйте обращение к этому API (это можно представлять себе как сочетание механизма запуска базы данных и очереди задач) |
| Socket | Позволяет пользователям создавать исходящие сокет-соединения и выполнять обмен информацией посредством таких соединений |
| Task Queue | Пользователи могут выполнять фоновые задачи (если необходимо — параллельно) без вмешательства со своей стороны |
| URLfetch | Взаимодействует с другими приложениями в режиме онлайн посредством запросов/ответов HTTP/S |

Окончание табл. 12.1

| Служба/API | Описание |
|------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Users | Служба идентификации в системе App Engine управляет процессом вхождения пользователя в систему |
| WarmUp | Загружает приложения в экземпляры до поступления трафика, чтобы сократить время обслуживания запроса |
| XMPP | Наделяет ваше приложение способностью общаться (мгновенно отправлять и/или принимать сообщения) посредством протокола Jabber/XMPP |

Конечно, все это выглядит впечатляюще, однако пора переходить от слов к делу! Первое, что вам нужно сделать, — это выбрать платформу, с помощью которой вы будете создавать свои приложения.

12.4. Выбор платформы для системы App Engine

Если вы разрабатываете приложение, не взаимодействующее непосредственно с пользователем (это означает, что другие приложения лишь обращаются к вашему приложению за обслуживанием), то выбор платформы не имеет особенно большого значения. В настоящее время в вашем распоряжении есть несколько вариантов, которые перечислены в табл. 12.2.

Таблица 12.2. Платформы для разработки с помощью Google App Engine

| Фреймворк | Описание |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| webapp, webapp2 | Предусмотренный по умолчанию “облегченный” вариант веб-платформы, который поставляется вместе с App Engine SDK |
| bottle | “Облегченный” вариант веб-микроплатформы на Python; поставляется вместе с адаптером App Engine (gae) |
| Django | Популярная полномасштабная веб-платформа на Python (не все возможности доступны для использования) |
| Django-nonrel | Служит “мостом” между исполняемыми Django-приложениями в нереляционных хранилищах данных, таких как App Engine |
| Flask | Еще одна микроплатформа (наподобие bottle), базирующаяся на Werkzeug и Jinja2 (наподобие описанной ниже Kay) и ориентированная на удобство настройки в соответствии с потребностями пользователя, но без “родного” уровня абстракции данных. Datastore App Engine вы можете использовать напрямую |
| GAE Framework | Упрощенный вариант, базирующийся на Django. Эту платформу можно применять в случае, если вы хотите воспользоваться уже существующими инфраструктурными приложениями, такими как users, blog, admin и т.п. Эту платформу можно представлять себе как упрощенный (Django+Pinax) для App Engine |
| Google App Engine Oil (GAE0) | Если webapp является чересчур упрощенной, а Django чересчур сложной, то эта платформа типа “model-view-template”, подобно Django, также наведена платформами Rails и Zend (Ruby) |
| Kay | Также подобна Django, но использует платформу нижнего уровня Werkzeug, машину формирования шаблонов Jinja2 и babel для трансляции языков |
| MVCEngine | Платформа, навеянная Rails и ASP.NET |

| Фреймворк | Описание |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pyramid | Еще одна популярная полномасштабная веб-платформа, базирующаяся на Pylons и gevent.bfg |
| tipfy | “Облегченная” платформа, более мощная, чем webapp, разработанная именно для App Engine. Это также привело к созданию платформы webapp2, а это значит, что ее первоначальный разработчик уже не поддерживает данную платформу |
| web2py | Еще одна полномасштабная веб-платформа для Python, которая обладает абстракцией более высокого уровня. Это означает, что данной платформой удобнее пользоваться, чем другими, однако она скрывает от глаз пользователей больше подробностей (что можно считать как достоинством, так и недостатком) |

Большинство начинающих пользователей App Engine, выбрав по умолчанию webapp или webapp2, со временем приходят к выводу о достаточно широких возможностях этих платформ, входящих в состав системы App Engine. Это очень хороший выбор, поскольку, несмотря на относительную простоту, платформа webapp предоставляет пользователю базовые инструменты, с помощью которых он может создавать весьма эффективные приложения. Однако немалое число веб-разработчиков, которых можно считать ветеранами Python, уже давно пользуются платформой Django и предпочитают именно этот подход. Вследствие ограниченности среды App Engine вам по умолчанию не предоставляется доступ ко всем возможностям Django. Однако кое-какая связь между App Engine и Django все же имеется.

Некоторые компоненты Django уже интегрированы в систему App Engine, а компания Google предоставляет некоторые версии платформы Django (правда, несколько устаревшие) на серверах App Engine, в результате чего пользователям не приходится выгружать всю инсталляцию Django вместе со своими приложениями. К их числу относятся версии Django 0.96, 1.2 и 1.3 (к тому времени, когда вы будете читать книгу, могут быть добавлены новые версии Django). Тем не менее существует несколько критических компонентов Django, которые не были реализованы в App Engine; самым важным среди них является Object-Relational Mapper (ORM), который традиционно рассчитан на наличие фундамента в виде реляционных баз данных SQL.

Я использую здесь слово “традиционно”, поскольку в настоящее время предпринимаются усилия, направленные на то, чтобы платформа Django поддерживала также нереляционные базы данных (NoSQL). Однако на момент написания этой книги ни один из таких проектов еще не интегрирован в дистрибутив Django. Возможно, к тому времени, когда вы будете читать эту книгу, Django уже будет поддерживать либо реляционные, либо нереляционные базы данных. В дополнение к предложениям, касающимся Django 1.3 и 1.4, одним из остальных широко известных проектов является Django-nonrel. Он представляет собой ветвь платформы Django, которой комплектуются адаптеры для Google App Engine, а также MongoDB (плюс еще несколько на подходе). Кроме того, ведутся работы по созданию JOIN для адаптеров NoSQL, но на данный момент все это также пребывает на стадии разработки. Если в ходе написания этой книги появится какой-то материал, представляющий интерес для разработчиков нереляционных баз данных Django, мы обязательно упомянем о нем.

Платформа tipfy представляет собой “облегченный” вариант, разработанный именно для App Engine. Платформу tipfy можно представлять себе как webapp++ или “webapp 2.0”, поскольку она включает в себя возможности, отсутствующие у

webapp. Этот набор возможностей включает следующие функции (но не ограничивается ими): интернационализация, управление сеансами, альтернативные формы установления подлинности (Facebook, FriendFeed, Twitter и т.п.), доступ к модулю Adobe Flash (доступ к протоколу AMF плюс Flash-сообщения), ACL (access control lists — списки управления доступом) и дополнительные машины построения шаблонов (Jinja2, Mako, Genshi). Платформа Tipfy основана на стандарте WSGI и подсоединяется к набору утилит Werkzeug, которые составляют фундамент любого приложения, совместимого с WSGI. Подробнее о платформе tipfy можно узнать на ее веб-странице по адресу <http://tipfy.org>.

Платформа web2py — одна из четырех широко известных полномасштабных платформ для Python (в дополнение к Django, TurboGears и Pyramid). Это вторая платформа, совместимая с Google App Engine. Создатели web2py стремились предоставить возможность разработчикам создавать быстродействующие, расширяемые, защищенные и машинезависимые веб-приложения, использующие ту или иную СУБД, будь то реляционное или нереляционное хранилище данных Google App Engine; web2py может работать с достаточно широким спектром баз данных. Уровень абстракции баз данных (database abstraction layer — DAL) преобразовывает ORM-запросы в SQL в реальном масштабе времени и использует это в качестве своего интерфейса с базой данных. Естественно, для приложений App Engine вы по-прежнему действуете в рамках реляционных ограничений, представленных платформой Datastore (т.е. использование оператора JOIN не допускается). Она также поддерживает ряд веб-серверов, таких как Apache, ligHTTPD или любой сервер, совместимый со стандартом WSGI. Использование web2py является естественным выбором для действующих разработчиков web2py, которые хотят перенести свои приложения в систему App Engine.

Для разработки своих приложений вы можете выбрать любую из этих платформ. Как альтернативный вариант, можно воспользоваться любой платформой, совместимой с платформой WSGI. В данном случае мы используем “наименьший общий знаменатель” (webapp); рекомендуем вам выполнять все приведенные ниже примеры с помощью webapp2.

А теперь немного истории. Одного увлеченного разработчика системы App Engine перестала устраивать выбранная им платформа, что послужило для него мотивацией к созданию платформы tipfy. Затем у него возникло желание усовершенствовать платформу webapp, и он, отказавшись от использования tipfy, создал версию webapp2, которая оказалась настолько удачной, что Google интегрировала webapp2 как часть версии 2.7 SDK времени исполнения (отсюда цитата в начале главы 11).

12.4.1. Платформы: webapp, затем Django

В главе 11 мы рассказывали о Django и о том, как создать блог с помощью такой платформы. В настоящей главе мы собираемся сделать то же самое, но с помощью webapp, предусмотренной по умолчанию. Мы покажем вам, как построить практически то же самое с помощью среды разработки App Engine. При желании пользователи могут также создать учетную запись Google или другую идентификацию OpenID (или воспользоваться уже существующей) и настроить приложение для выполнения в интерактивной среде производства App Engine. Мы также покажем вам, как сделать это (впрочем, делать это вовсе необязательно). Чтобы задать какое-либо приложение в режиме онлайн, вам не понадобится кредитная карточка, но вам понадобится мобильный телефон с возможностью обмена текстовыми сообщениями или SMS.

В завершение этой главы мы перенесем это приложение на платформу Django и выполним *результат переноса* в системе App Engine (состоящей из среды разработки и среды производства). Концепциям и возможностям App Engine можно было бы посвятить отдельную книгу, и хотя мы не собираемся приводить здесь исчерпывающие сведения об App Engine, изложенного здесь материала будет вполне достаточно для того, чтобы вы могли свободно ориентироваться в разных аспектах продукта App Engine.

Загрузка и инсталляция набора SDK App Engine

Для начала вам нужно получить набор SDK App Engine для своей платформы разработчика. Для загрузки предлагаются разные варианты файлов, поэтому ваша задача — выбрать вариант, подходящий для вашей системы. Посетите домашнюю страницу Google App Engine по адресу <http://code.google.com/appengine>, а затем щелкните на ссылке Downloads. Выполнив переход по этой ссылке, вы найдете подходящие файлы для своей системы. Предусмотрены также файлы для Java-разработчиков, но мы сосредоточимся только на языке Python.

Пользователи систем Linux или *BSD должны загрузить zip-файл, распаковать соответствующий архив и установить требуемую папку (`google_appengine`) в подходящем для себя месте, например `/usr/local`, а также вставить ссылку на команды `dev_appserver.py` и `appcfg.py` в каком-либо подходящем месте (например, `/usr/local/bin`). Как альтернативный вариант, вы можете просто добавить `/usr/local/google_appengine` в описание своего пути. (Можете пропустить оставшийся материал этого раздела, а также следующий раздел, посвященный использованию приложения Launcher, и перейти непосредственно к разделу “Создание приложения Hello World вручную”).

Пользователям персональных компьютеров под управлением операционной системы Windows следует загрузить файл `.msi`; а пользователям компьютеров Mac — файл `.dmg`. После того как вы найдете подходящий для себя файл, щелкните на нем дважды или запустите его, чтобы установить набор SDK App Engine. В ходе этого процесса будет также установлено приложение Google App Engine Launcher. Его можно использовать для управления приложениями App Engine, размещенными на вашем компьютере разработчика, а также для того, чтобы помочь вам выгрузить их на сервер Google для выполнения в производственной среде App Engine в интерактивном режиме.

Использование приложения Launcher для создания программы “Hello World” (только для пользователей Windows и Mac)

После запуска приложения Launcher вы увидите панель управления, подобную тем, которые представлены на рис. 12.4 и 12.5.

С помощью соответствующих кнопок панели управления вы можете вызвать (и убрать) свой сервер разработки (Run); просмотреть свой журнал (Logs); просмотреть свою консоль администрирования (SDK console); изменить параметры конфигурации (Edit); выгрузить свое приложение на производственные серверы App Engine (Deploy) или перейти на консоль администрирования своего интерактивного приложения (Dashboard). Давайте рассмотрим процесс создания нового приложения. В ходе разработки нашей новой программы воспользуемся несколькими кнопками приложения Launcher.

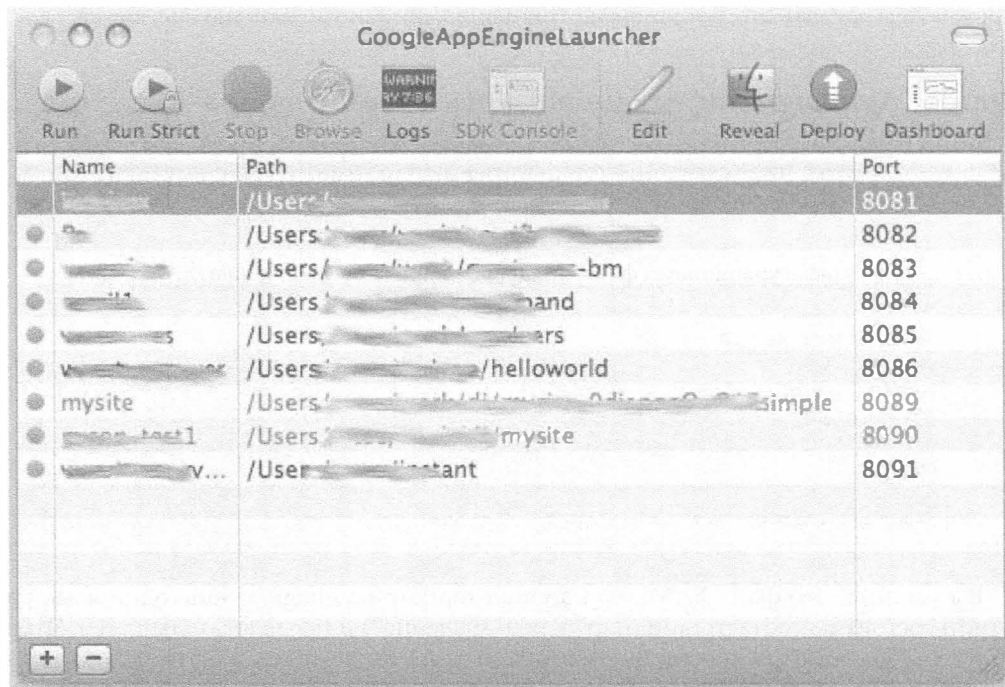


Рис. 12.4. Приложение Launcher App Engine для Mac

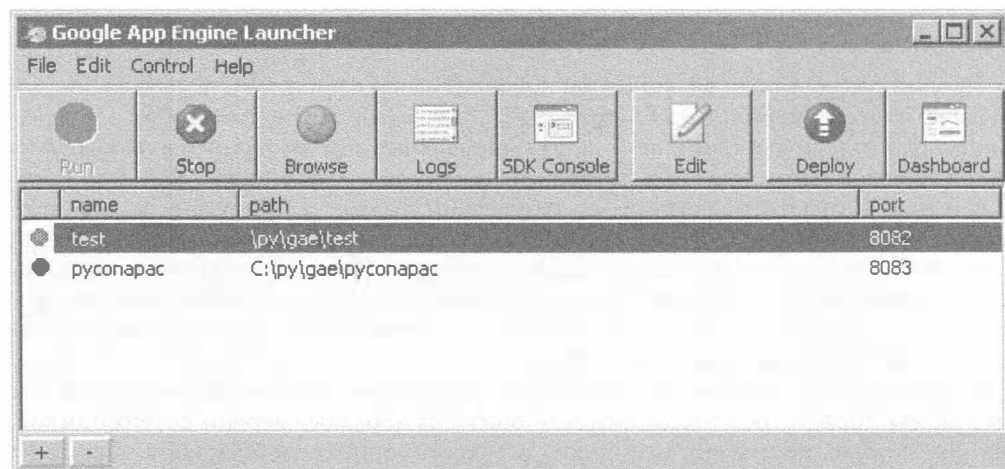


Рис. 12.5. Приложение Launcher App Engine для Windows

Зайдите в меню и раскройте перечень вариантов для создания нового приложения. Присвойте своему приложению уникальное имя (имя “helloworld”, наверное, уже выбрано). Можете также указать несколько других параметров, например, папку, в которой будут создаваться новые стандартные файлы, а также номер порта сервера. После того как эта работа будет выполнена, вы увидите свое приложение на главной панели приложения Launcher, и это означает, что оно готово к выполнению. Но,

прежде чем сделать это, взглянем на три файла, созданные автоматически: `app.yaml`, `index.yaml` и `main.py`.

Файлы App Engine, предусмотренные по умолчанию

Файл `app.yaml` представляет ваши конфигурационные параметры. Этот файл, предусмотренный по умолчанию и генерируемый автоматически, выглядит примерно так, как показано в примере 12.1.

Пример 12.1. Конфигурационный файл, предусмотренный по умолчанию (`app.yaml`)

```
1 application: APP_ID
2 version: 1
3 runtime: python
4 api_version: 1
5
6 handlers:
7   - url: .*
8     script: main.py
```

Вы увидите, что файл YAML (yet another markup language — еще один язык разметки) состоит из соответствий (пар “ключ–значение”) и последовательностей. Чтобы получить более подробную информацию об этом типе файла, обратитесь по адресу <http://yaml.org> и <http://en.wikipedia.org/wiki/Yaml>.

Построчное объяснение

Строки 1–4

Первый раздел представляет собой конфигурацию в чистом виде: назначение имени вашему приложению App Engine (`APP_ID`), за которым следует номер его версии. Для разработки можете выбрать любое понравившееся имя, например *blog*. Если же вы собираетесь выгрузить его в интерактивную производственную среду App Engine, то вам нужно будет проявить более творческий подход и придумать какое-нибудь более оригинальное имя (во всяком случае, такое, которое еще не выбрано кем-либо другим). Выбирая имя для приложения, важно учитывать следующее: имена нельзя переносить и использовать повторно; как только имя присвоено, вы теряете контроль над ним — даже если удалите соответствующее приложение. Поэтому к выбору имен нужно относиться очень внимательно.

Номер версии — это уникальная строка, которую вы можете сформировать сами. Именно вы должны решить, как будете обозначать разные версии своего приложения. Вы можете, например, использовать традиционную систему обозначений (0.8, 1.0, 1.1, 1.1.2, 1.2 и т.д.) или воспользоваться какой-то другой, например v1.6 или 1.3beta. Это не более чем строка символов, однако в ней можно использовать лишь алфавитно-цифровые символы и дефисы. Вы можете создать до десяти версий своего приложения (старших или младших — не имеет значения), но после этого вы не сможете выгрузить ни одной новой версии, пока не удалите хотя бы одну уже существующую.

Под номером версии указывается тип среды исполнения. В данном случае это Python и версия 1 этого API. Вы можете также использовать `app.yaml` для Java и JRuby, вставив “Go” между Java и JRuby, и другие среды исполнения для JVM;

файл `app.yaml` используется по очереди, чтобы сгенерировать файлы `web.xml` и `appengine-web.xml`, которые на самом деле необходимы для вашего сервлета (или сервлетов).

Строки 6–8

Несколько последних строк указывают ваши программы обработки. Как и в случае файла Django `URLconf`, вы должны указать какое-либо регулярное выражение, которое соответствовало бы запросам клиента, а также предоставить соответствующую программу обработки. На платформе Django эти пары “url-сценарий” программы обработки соответствуют файлу `URLconf` уровня проекта, который переправляет запросы на `URLconf` уровня приложения. Аналогично в `app.yaml` директива сценария направляет запрос в данный Python-сценарий, который содержит больше конкретных URL и отображает их на классы программы обработки, точно так же, как `URLconf` какого-либо Django-приложения указывает на функцию представления.

Если вы хотите узнать больше о конфигурировании своего приложения, ознакомьтесь с соответствующей документацией на сайте <http://code.google.com/appengine/docs/python/config/appconfig.html>.

Теперь рассмотрим файл `index.yaml`:

```
indexes:
# AUTOGENERATED
# This index.yaml is automatically updated whenever the dev_appserver
. . .
```

Файл `index.yaml` требуется каждый раз, когда вам нужно будет создать для своего приложения специализированные указатели. Чтобы ускорить выполнение запросов App Engine к хранилищу данных, вам нужно иметь для каждого запроса соответствующий указатель (индекс). (Указатели для простых запросов создаются автоматически — ваше вмешательство не требуется.) Вообще говоря, вам не следует беспокоиться об этом до тех пор, пока ваши запросы не станут достаточно сложными. Если вы хотите получить более подробную информацию об использовании указателей, обратитесь к официальной документации на сайте <http://code.google.com/appengine/docs/python/config/indexconfig.html>.

Последний файл, который автоматически генерируется Launcher от вашего имени, является главным файлом приложения (`main.py`), как показано в примере 12.2.

Пример 12.2. Главный файл приложения (`main.py`)

```
1  from google.appengine.ext import webapp
2  from google.appengine.ext.webapp import util
3
4  class MainHandler(webapp.RequestHandler):
5      def get(self):
6          self.response.out.write('Hello world!')
7
8      def main():
9          application = webapp.WSGIApplication([('/', MainHandler)],
10                                              debug=True)
11          util.run_wsgi_app(application)
12
13  if __name__ == '__main__':
14      main()
```

Построчное объяснение

Строки 1–2

Первые две строки импортируют платформу `webapp`, а также вызывают ее функцию-утилиту `run_wsgi_app()`.

Строки 4–6

После этих вступительных строк вы обнаруживаете класс `MainHandler`. Это является основной функцией данного примера. Здесь определяется метод `get()` для обработки запросов HTTP GET — отсюда его название. Экземпляр программы обработки будет обладать атрибутами как для запроса, так и для ответа. В нашем примере мы выписываем только HTML/текст, чтобы вернуть его пользователю посредством файла `response.out`.

Строки 8–11

Затем наступает очередь функции `main()`, которая порождает экземпляр приложения и выполняет его. В вызове, порождающем экземпляр `webapp.WSGIApplication`, вы обнаружите пары — точнее говоря, пока только одну пару, которая определяет, какой процесс программы (программ) обработки что запрашивает. В нашем случае единственным URL, который обрабатывает наше приложение в данный момент, является `/`, и эти запросы будут обрабатываться классом `MainHandler`, описанным выше.

Строки 13–14

Наконец, мы видим уже знакомые нам строки для определения выполнения, исходящие из того, был ли исходный Python-файл импортирован или исполнялся непосредственно как сценарий. Если же этот код незнаком вам, то рекомендуем вернуться к материалу глав 3 и 12 в книге *Core Python*.

В приведенном выше коде нет ничего сложного, даже если что-то из этого вам встретилось впервые в жизни. С этого момента мы будем постоянно вносить в это приложение какие-то изменения с целью его усовершенствования или добавления в него новых функций.

Наведение минимального порядка в коде

Прежде чем приступить к наращиванию нашего приложения, внесем в файле `main.py` несколько “косметических” правок, которые не оказывают никакого влияния на исполнение этого кода (пример 12.3).

Пример 12.3. Наведение порядка в главном файле приложения (`main.py`)

```
1 from google.appengine.ext import webapp
2 from google.appengine.ext.webapp.util import run_wsgi_app
3
4 class MainHandler(webapp.RequestHandler):
5     def get(self):
6         self.response.out.write('Hello world!')
7
8 application = webapp.WSGIApplication([
9     ('/', MainHandler),
10 ], debug=True)
```

```
11
12 def main():
13     run_wsgi_app(application)
14
15 if __name__ == '__main__':
16     main()
```

Что и зачем мы сделали

1. Мы не хотим, чтобы экземпляры `WSGIApplication` создавались каждый раз, когда выполняется это приложение. Переместив `webapp.WSGIApplication` из `main()` в блок глобального кода, мы лишь однократно создаем экземпляр этого класса, вместо того чтобы создавать его в результате каждого запроса. Выигрыш в производительности получается лишь незначительный, однако это лишь простая оптимизация, которую можно выполнять в любом аналогичном Python-приложении, независимо от того, используем ли мы систему App Engine или нет. Единственной (минимальной) платой за такое усовершенствование является то, что наше приложение теперь становится глобальной, а не локальной переменной.
2. Поскольку мы используем лишь одну функцию из `webapp.util`, мы можем упростить импорт, добавляя одно такое имя исключительно для того, чтобы ускорить обращение к функции `run_wsgi_app()`. Разница между функциями `util.run_wsgi_app()` и `run_wsgi_app()` несущественна, если вы выполняете их один или два раза, но если запросы к вашему приложению исчисляются миллионами, то разница может оказаться весьма значительной.
3. Указание пар обработчиков в отдельных строках (строке) облегчает добавление новых обработчиков; например:

```
(('/', MainHandler),
 ('/this', DoThis),
 ('/that', DoThat),
 ...)
```

Вообще говоря, это все, что на данный момент заслуживает дополнительного разъяснения. Это придает приложению Django-подобный вид, если можно так выразиться.

12.5. Поддержка версии Python 2.7

2.7

Первоначальный выпуск среды Python для системы Google App Engine поддерживал версию 2.5 (а именно 2.5.2 на сервере). Недавно компания Google выпустила новую версию 2.7 среды исполнения (а именно 2.7.2 на сервере). На момент написания этой книги поддержка версии 2.7 все еще носит экспериментальный характер, поэтому мы намерены оставить все оставшиеся примеры кода в версии 2.5; впрочем, для разработки вы можете использовать версию 2.6 или 2.7. Однако в эту новую среду разработки внесено несколько изменений, которые следует обязательно иметь в виду. Мы также расскажем о некоторых различиях в коде, чтобы вы могли поэкспериментировать с ним в оставшейся части главы и адаптировать его к версии 2.7, если вы предпочитаете именно эту среду исполнения, а не версию 2.5.

12.5.1. Различия общего характера

Первым и одним из самых важных различий является то, что среда исполнения версии 2.7 поддерживает параллельное выполнение. Учитывая модель формирования цен, принятую в системе App Engine, вы оплачиваете предоставляемые вам услуги исходя из количества экземпляров вашего приложения, которые обслуживают трафик. Поскольку среда исполнения версии 2.5 не поддерживает параллельное выполнение, должны порождаться новые экземпляры, если ваши выполняющиеся экземпляры оказываются не в состоянии справиться с принимаемым вами трафиком. Это может привести к росту затрат. Если же используемая вами среда исполнения поддерживает параллельное выполнение, то ваше приложение может отвечать на запросы асинхронным образом и существенно снизить потребность в дополнительных экземплярах.

Далее, чрезвычайно желательные и ранее запрещенные к использованию библиотеки языка С сейчас разрешены к использованию. Они включают PIL, lxml, NumPy и simplejson (называемую просто json). Поддержка версии 2.7 включает также систему формирования шаблонов Jinja2 наряду с шаблонами Django. Если хотите ознакомиться со всеми различиями между средами исполнения версий 2.5 и 2.7, обратитесь к официальной документации на сайте <http://code.google.com/appengine/docs/python/python27/newin27.html>.

12.5.2. Различия в коде

Существуют также небольшие различия в коде. Рассмотрим эти различия, поскольку речь идет об изменениях, которые вы должны будете внести в свой код, приведенный в этой главе, чтобы выполнить свое приложение в среде исполнения версии 2.7. Изменения в файле `app.yaml` касаются поля `runtime`. Кроме того, вы наверняка захотите включить возможность параллельного выполнения посредством директивы `threadsafe`. Еще одним важным изменением является переход к чистому WSGI: вместо того чтобы указывать какой-либо сценарий для исполнения, вы должны указать на какой-либо объект (прикладной объект). Все необходимые различия выделены *курсивом* в примере 12.4.

Пример 12.4. Образец конфигурационного файла Python 2.7 (`app.yaml`)

```

1 application: APP_ID
2 version: 1
3 runtime: python27
4 api_version: 1
5 threadsafe: true
6
7 handlers:
8   - url: .*
9     script: main.application

```

Особенностью среды исполнения версии 2.7 является наличие новой и усовершенствованной платформы webapp под названием webapp2. Поскольку вместо CGI мы используем WSGI, мы можем убрать излишний `main()` внизу. Все изменения в `main.py` отражены в примере 12.5, код которого, как нетрудно убедиться, не только короче, но и удобнее для чтения.

Пример 12.5. Образец файла главного приложения Python 2.7 (main.py)

```
1 from google.appengine.ext import webapp2
2
3 class MainHandler(webapp2.RequestHandler):
4     def get(self):
5         self.response.out.write('Hello world!')
6
7 application = webapp2.WSGIApplication([
8     ('/', MainHandler),
9 ])
```

Обратите внимание: объект `application` в `main.py` — это `main.application`, на который есть ссылка в файле `app.yaml`. Подробнее о различиях между файлами `main.py`, используемыми в версиях 2.5 и 2.7, читайте на сайте <http://code.google.com/appengine/docs/python/tools/webapp/overview.html>.

Тем, кто хочет узнать больше об использовании среды выполнения версии 2.7 и получить дополнительную информацию о представленных выше изменениях, рекомендуем ознакомиться с документацией на сайте <http://code.google.com/appengine/docs/python/python27/using27.html>.

12.6. Сравнение с платформой Django

Система App Engine не структурирует какой-либо веб-сайт как проект, составленный из одного или нескольких приложений. Напротив, все объединяется в единственном приложении. Мы уже упоминали, что файл `app.yaml` в какой-то степени похож на `urls.py` проектного уровня на платформе Django, поскольку он отображает указатели URL на программы обработки. Он также содержит элементы `settings.py`, поскольку является конфигурационным файлом.

Файл `main.py` представляет собой сочетание `urls.py` платформы Django плюс `views.py`. При создании приложения по стандарту WSGI у вас есть одна или несколько программ обработки, которые назначают класс, экземпляр которого будет обрабатывать соответствующие запросы. Определения этого класса, а также их соответствующие обработчики `get()` или `post()` также создаются в этом файле. Эти обработчики будут ближайшими к функции просмотра.

В главе 11 мы смогли протестировать наше приложение с помощью сервера разработки. У системы App Engine есть свой собственный сервер разработки, и по ходу дела мы воспользуемся им.

12.6.1. Запуск приложения Hello World

Есть два способа запустить какое-либо приложение на сервере разработки. Если вы находитесь в приложении Launcher, выберите строку интересующего вас приложения, а затем щелкните на кнопке Run. Через несколько секунд вы увидите, что эта пиктограмма поменяла свой цвет на зеленый. Затем можете щелкнуть на кнопке Browse, чтобы запустить веб-браузер, который открывает ваше приложение.

Чтобы запустить приложение посредством командной строки, выберите путь, на котором находится файл `dev_appserver.py`, а затем задайте команду

```
$ dev_appserver.py DIR.
```

где `DIR` — имя папки вашего приложения (которая содержит файлы `app.yaml` и `main.py`). Разумеется, если в данный момент у вас открыта папка, в которой находятся оба эти файла, можно воспользоваться упрощенным вариантом той же команды:

```
$ dev_appserver.py.
```

Это выглядит немножко не так, как в среде Django, где используется инструмент командной строки, ориентированный на определенный проект (`manage.py`), в отличие от общей команды, инсталлированной для всех приложений App Engine. Еще одна небольшая разница заключается в том, что сервер разработки на платформе Django стартует на порте 8080, тогда как система App Engine использует порт 8000. Это означает лишь то, что ваш URL должен измениться на `http://localhost:8080/` или `http://127.0.0.1:8080`. В случае использования одного из приложений Launcher, когда вы создаете новое приложение, это приложение автоматически назначит ему какой-либо уникальный номер порта. В результате вам либо придется пользоваться этим номером, либо изменить его вручную.

12.6.2. Создание приложения Hello World вручную

Если вы не используете приложение Launcher, то, наверное, не нуждаетесь в какой-либо помощи при вводе представленного выше кода. Поскольку на данном этапе использование файла `index.yaml` является необязательным, вам требуются лишь “скелетные” файлы `app.yaml` и `main.py`. Вы можете ввести их вручную или перейти на веб-сайт данной книги и загрузить их из папки Chapter 12. После того как вы создадите оба эти файла, можно запустить сервер разработки, воспользовавшись той же командой, которая была описана выше (`dev_appserver.py`).

*Выгрузка вашего приложения на Google в интерактивном режиме

Возможно, это было бы несколько преждевременно, но если хотите, то можете не ограничиваться выполнением своего приложения на сервере разработки. Вы можете также выгрузить его на сервер Google и запустить в интерактивном режиме в производственной среде, сделав таким образом свое простое приложение Hello World доступным для всего мира (разумеется, за исключением мест, где недоступна служба Google), что вполне соответствовало бы его названию (“Здравствуй, мир”). Конечно же, это совершенно необязательно, поэтому, если такой вариант не представляет для вас интереса, перейдите к следующему разделу и продолжайте создание своего блога.

Система App Engine предоставляет бесплатный уровень обслуживания, на котором вы можете разрабатывать простые приложения с невысоким уровнем трафика, не затрачивая на это ни цента. Вам потребуется лишь мобильный телефон, который поддерживает SMS, а также учетная запись в службе Google (Google Account), но кредитная карточка не понадобится, если только вы не собираетесь превысить бесплатную квоту, доступную для всех приложений. Для того чтобы создать учетную запись в системе App Engine, обратитесь к сайту <http://appengine.google.com> и зарегистрируйтесь на нем.

Чтобы выгрузить свое приложение (и его статические файлы, если таковые имеются), воспользуйтесь либо одним из приложений Launcher (только в системах Mac или Windows), либо инструментом командной строки, `appcfg.py`. Для этого отправьте команду обновления и перейдите в папку верхнего уровня, где находится ваш

файл `app.yaml`. Ниже приведен пример выполнения `appcfg.py` в текущем каталоге. Обратите внимание: вам придется подтвердить свои права (ввести правильный адрес электронной почты и пароль) как разработчика данного приложения, что продемонстрировано в приведенном ниже фрагменте.

```
$ appcfg.py update .
Application: APP_ID; version: 1.
Server: appengine.google.com.
Scanning files on local disk.
Initiating update.
Email: YOUR_EMAIL
Password for YOUR_EMAIL: *****
Cloning 2 static files.
Cloning 3 application files.
Uploading 2 files and blobs.
Uploaded 2 files and blobs
Precompilation starting.
Precompilation completed.
Deploying new version.
Checking if new version is ready to serve.
Will check again in 1 seconds.
Checking if new version is ready to serve.
Will check again in 2 seconds.
Checking if new version is ready to serve.
Closing update: new version is ready to start serving.
Uploading index definitions.
```

Чтобы выгрузить приложение, потребуется примерно одна минута (как правило, не больше). Приведенный выше пример выгружался примерно 3 секунды.

Еще через несколько секунд после завершения загрузки вы (и любой другой человек на нашей планете), обратившись на <http://12-X.appspot.com>, увидите на своем экране приветствие “Hello World!”. Здорово!



Тщательно выбирайте имя для своего приложения

Прежде чем выгрузить исходный код и статические файлы для своего приложения, не забудьте выбрать уникальное, еще никем не используемое имя (указываемое в `app.yaml`). Помните, что имена приложений присваиваются навсегда. Их нельзя использовать повторно или переносить, даже если соответствующее приложение блокируется и/или удаляется.

12.7. Преобразование приложения Hello World в простой блог

Теперь, когда вам удалось успешно создать и выполнить простое приложение Hello World, у вас должна появиться возможность, запустив браузер, перейти на свой веб-сайт. Находясь в приложении Launcher, щелкните на кнопке Browse, а если не используете его, то укажите любой веб-браузер на <http://localhost:8080>. После этого вы должны увидеть нечто подобное тому, что показано на рис. 12.6.

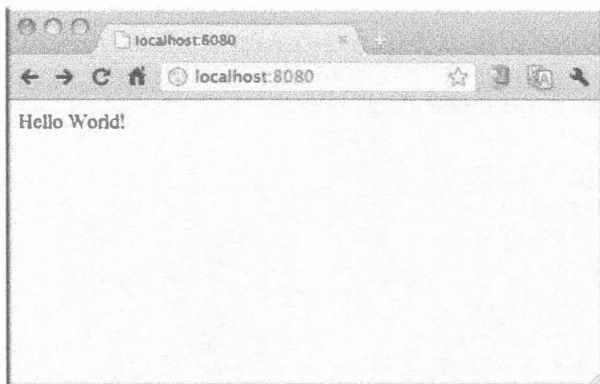


Рис. 12.6. Приложение Hello World на платформе Google App Engine

На следующем этапе вы должны приступить к преобразованию своего приложения в нечто более полезное с практической точки зрения. Мы намерены повторить наш пример с платформой Django, преобразовав это простое приложение Hello World в блог. Мы делаем это для того, чтобы предоставить вам возможность сравнить разработку на платформах Django и webapp.

12.7.1. Быстрый просмотр изменений: преобразование простого текста в HTML за 30 секунд

Прежде всего подтвердите, что вам нужно лишь обновить свой код, чтобы увидеть изменения, представленные в данном приложении на сервере разработки. Для этого добавьте в строку вывода тег `<h1>` и закройте ее. Измените текст (если вам в голову не приходят более интересные идеи, пусть этот текст выглядит, например, так: “The Greatest Blog”). В итоге получаем: `<h1>The Greatest Blog</h1>`. Как и ранее, вы должны сохранить внесенное вами изменение (сохранение нужно выполнить после внесения любых изменений в ваш исходный код), подтвердить, что можете вернуться в свой браузер, обновить соответствующую страницу, а затем подтвердить внесенные изменения (рис. 12.7).



Рис. 12.7. Изменения в программе Hello World 2 тотчас же отражаются на обновленной странице браузера

12.7.2. Добавление формы

Теперь нам предстоит сделать более значительный шаг в разработке вашего приложения: наделить его способностью принимать информацию, вводимую пользователем. Нам предстоит вставить форму с полями, с помощью которой пользователи могут создавать новые сообщения (“посты”) в блоге. Этими двумя полями являются название и содержимое (или тело) сообщения. Теперь ваш модифицированный метод `MainHandler.get()` должен выглядеть примерно так:

```
class MainHandler(webapp.RequestHandler):
    def get(self):
        self.response.out.write('''
            <h1>The Greatest Blog</h1>
            <form action="/post" method=post>
            Title:
            <br><input type=text name=title>
            <br>Body:
            <br><textarea name=body rows=3 cols=60></textarea>
            <br><input type=submit value="Post">
            </form>
            <br>
            ''')
```

Весь этот метод состоит из соответствующей веб-формы. Разумеется, если бы это было реальное приложение, то вся разметка HTML была бы в определенном шаблоне. На рис. 12.8 представлены обновленный экран и новые поля ввода.



Рис. 12.8. Добавление полей формы в приложение Blog

Теперь можете заполнять эти поля по своему усмотрению, как показано на рис. 12.9.



Рис. 12.9. Заполнение полей формы в приложении Blog

Подобно приведенному ранее примеру с платформой Django, мы еще не вполне способны обрабатывать эти данные. Когда пользователь заполняет и передает форму на данной стадии, наш контроллер еще не в состоянии обрабатывать эти данные, поэтому если вы попытаетесь передать такую форму, то либо получите сообщение об ошибке, либо увидите пустой экран. Нам нужно добавить обработчик запроса POST, который будет выполнять обработку новых сообщений в блоге. Именно этим мы сейчас и займемся, создав новый класс `BlogEntry` и метод `post()`:

```
class BlogEntry(webapp.RequestHandler):
    def post(self):
        self.response.out.write('<b>%s</b><br><hr>%s' % (
            self.request.get('title'),
            self.request.get('body'))
        )
```

Обратите внимание: наш метод называется `post()` (в отличие от `get()`). Это объясняется тем, что рассматриваемая нами форма передает запрос POST. Если вы хотите также поддерживать GET, то вам понадобится еще один метод под названием `get()`. Используемый вами класс и его метод, конечно, сами по себе замечательны, но ваше приложение не сможет обратиться к программе обработки, если пара “URL-класс” не будет указана при создании объекта приложения. Это должно выглядеть примерно так:

```
application = webapp.WSGIApplication([
    ('/', MainHandler),
    ('/post', BlogEntry),
], debug=True)
```

Сделав это добавление, можете заполнять поля формы и передавать ее в свое приложение. Результат, который вы видите на экране (рис. 12.10), в точности соответствует тому, что указывает наша программа обработки `post()`: на экране отображается название и содержимое нашего сообщения.



Рис. 12.10. Результаты передачи формы

12.7.3. Добавление службы Datastore

Результат, представленный на рис. 12.10, выглядит просто замечательно, однако как блог данное приложение совершенно бесполезно: вы ничего не сохраняете. Это именно то место, где мы разошлись с платформой Django. На платформе Django мы были *вынуждены* создать базу данных, и первой строкой написанного нами кода была соответствующая модель данных. Подход, используемый в системе App Engine, в большей мере ориентирован на приложения: мы начали с создания приложения, даже не располагая какой-либо моделью данных. Более того, нам *не нужна* какая-либо база данных: вы можете просто использовать кеш, сохранить свои данные в службе Blodstore или где-либо еще в облаке.

Механизмом хранения данных в системе App Engine является ее хранилище данных. Компания Google проводит четкое различие между хранилищем данных и базой данных, чем, собственно говоря, и обусловлено некоторое различие в терминологии. Это было сделано для того, чтобы пользователи с самого начала понимали, что в случае системы App Engine речь вовсе не идет о системе управления реляционной базой данных (СУРБД). Хранилище данных Google App Engine представляет собой надстройку над службой Bigtable³ и служит распределенным, расширяемым, нереляционным, постоянным местом хранения данных. В нем также используется технология Google под названием Megastore⁴, обеспечивающая строгую согласованность и высокий коэффициент готовности.

Следует иметь в виду, что это хранилище данных используется лишь тогда, когда вы развертываете свое приложение в интерактивном режиме в производственной среде App Engine. Во время работы сервера разработки вы можете сохранять свои данные в двоичном формате (по умолчанию) или запросить сохранение в SQLite, воспользовавшись флагом `--use_sqlite` при выполнении `dev_appserver.py`.

³ <http://labs.google.com/papers/bigtable.html>.

⁴ <http://research.google.com/pubs/pub36971.html>.

Настало время создать нашу модель данных. Проанализируйте и сравните класс модели на платформах Django и App Engine и обратите внимание на потрясающее сходство:

```
# Django
class BlogPost(models.Model):
    title = models.CharField(max_length=150)
    body = models.TextField()
    timestamp = models.DateTimeField()

# App Engine
class BlogPost(db.Model):
    title = db.StringProperty()
    body = db.TextProperty()
    timestamp = db.DateTimeProperty(auto_now_add=True)
```

В случае приложений App Engine вы должны добавить эту модель в уже имеющийся у вас файл `main.py`: эквивалентный ему файл `models.py` отсутствует, если только вы не создали его вручную. Не забудьте добавить службу Datastore, воспользовавшись следующим импортом:

```
from google.appengine.ext import db
```

Если вы относитесь к числу “нереляционных” пользователей платформы Django (т.е. предпочитаете выполнять приложение Django на платформе App Engine), то оставьте свой класс таким, как он был определен изначально (для Django), вместо того чтобы использовать модели данных App Engine.

Какие бы классы вы ни выбрали, сейчас можете запросить сохранение своих данных с помощью исходного механизма постоянного хранения. Первым шагом является создание соответствующего класса. Сохранение актуальных данных требует таких же действий, какие мы выполняли в системе Django: создание экземпляров, ввод пользовательских данных и последующее сохранение. Для нашего приложения необходимо заменить код в методе `post()`. В том виде, как он выглядит сейчас, он лишь выводит на экран информацию, введенную пользователем; в этом трудно усмотреть что-то заведомо полезное или постоянное.

В данном случае название и тело достаточно просты: после создания экземпляра извлеките их из данных представленной формы и назначьте их как атрибуты. Отметка времени необязательна, поскольку мы решили, что она будет создаваться автоматически, при создании экземпляра. Как только объект будет “завершен”, мы сохраним его в хранилище Datastore App Engine, вызвав метод `put()` соответствующего экземпляра данных, а затем направив пользователя на главную страницу нашего приложения (точно так же мы поступали ранее, в версии Django).

Ниже представлен код нового метода `BlogEntry.post()`, в котором отражены все описанные выше изменения.

```
class BlogEntry(webapp.RequestHandler):
    def post(self):
        post = BlogPost()
        post.title = self.request.get('title')
        post.body = self.request.get('body')
        post.put()
        self.redirect('/')
```

Обратите внимание: мы полностью заменили наш первоначальный метод `post()`, который лишь механически повторял информацию, вводимую пользователем. В более раннем примере данные вообще не сохранялись в долговременном хранилище. В результате перечисленных выше модификаций произошло кардинальное изменение ситуации: теперь в хранилище данных сохраняются все сообщения. Аналогично соответствующее изменение нужно внести в нашу программу обработки GET.

В частности, следует отобразить предыдущие сообщения в блоге, чтобы показать, что мы действительно начали сохранять пользовательские данные в долговременном хранилище. В нашем простом примере мы решили отображать указанную форму, сопровождая ее дампом любых существующих объектов `BlogPost`. Внесите следующие изменения в метод `MainHandler.get()`:

```
class MainHandler(webapp.RequestHandler):
    def get(self):
        self.response.out.write('''
        <h1>The Greatest Blog</h1>
        <form action="/post" method=post>
        Title:
        <br><input type=text name=title>
        <br>Body:
        <br><textarea name=body rows=3 cols=60></textarea>
        <br><input type=submit value="Post">
        </form>
        <hr>
        ''')

        #posts = db.GqlQuery("SELECT * FROM BlogEntry")
        posts = BlogPost.all()
        for post in posts:
            self.response.out.write(''<hr>
            <strong>%s</strong><br>%s
            <blockquote>%s</blockquote>''' % (
                post.title, post.timestamp, post.body)
            )
```

Код, передающий данную HTML-форму клиенту, остается неизменным. Под этим кодом мы добавляем код, извлекающий результаты из хранилища данных, чтобы отобразить их пользователю. В системе App Engine предусмотрены два способа, с помощью которых можно запросить ваши данные.

Действия в “объектном” стиле оказываются ближе всего к механизму запроса, предусмотренному на платформе Django: вы запрашиваете `BlogPost.all()` (в отличие от того, как это делается на платформе Django: `BlogPost.objects.all()`). В системе App Engine также предусмотрена альтернатива для тех, кому удобнее пользоваться SQL, — сокращенный синтаксис языка запросов, известный как GQL.

Поскольку в вашем распоряжении нет полного варианта SQL (а также оператора JOIN) и поскольку это в меньшей степени соответствует духу языка Python, настоятельно рекомендуем использовать более свойственный данной системе объектный подход. Однако если вы совершенно не можете обойтись без этого, то закомментированная строка, расположенная непосредственно над нашим вызовом `BlogPost.all()`, служит эквивалентом на языке GQL. Наконец, цикл, расположенный в конце приведенного выше кода, осуществляет проход по всем элементам и отображает соответствующие данные для каждого сообщения (поста).

Как показано на рис. 12.11, после внесения указанных изменений и повторного входа на тот же блог мы видим нечто, отличное от того, что видели раньше.



Рис. 12.11. Результаты передачи формы (сохраненные в хранилище данных)

Рис. 12.11 и 12.13 демонстрируют, что теперь, когда мы уверены в том, что пользовательские данные сохраняются, мы можем продолжать добавлять входы в блог.



Рис. 12.12. Заполнение формы для второго BlogPost



Рис. 12.13. Второй объект BlogPost сохранен и отображается на экране

12.7.4. Постепенные усовершенствования

Подобно нашему примеру на платформе Django, попытаемся сделать наш блог еще более полезным с практической точки зрения, переставив все входы в хронологической последовательности и показывая лишь десять самых последних из них. Вот изменения, которые нам нужно внести в строку запроса (и эквивалентные варианты для GQL):

```
#post = db.GqlQuery("SELECT * FROM BlogEntry ORDER BY timestamp  
DESC LIMIT 10")  
posts = BlogPost.all().order('-timestamp').fetch(10)
```

Сравните этот запрос с Django-запросом и обратите внимание на их сходство:

```
posts = BlogPost.objects.all().order_by('-timestamp')[:10]
```

Все прочее остается неизменным. Тем, кто хочет узнать больше о выполнении запросов в Google App Engine, рекомендуем обратиться на страницу документации сайта <http://code.google.com/appengine/docs/python/datastore/creatinggettinganddeletingdata.html>.

12.7.5. Консоль Разработки/SDK

Приложение Datastore Viewer

Несмотря на то что App Engine меркнет в сравнении с приложением admin на платформе Django, в системе App Engine все же имеется консоль разработки. Чтобы отобразить ее на экране в приложении Launcher, щелкните на кнопке SDK Console. Если у вас нет приложения Launcher, введите вручную особый URL, `http://localhost:8080/_ah/admin/datastore`. После того как вы обратитесь по этому адресу, будет запущено приложение Datastore Viewer, как показано на рис. 12.14.

Здесь вы можете создать новый экземпляр любого из элементов, которые определили для своего приложения. В данном случае у нас есть только элемент `BlogPost`. Вы можете также увидеть содержимое объектов в хранилище данных. На рис. 12.15 представлены два исходных сообщения, созданные нами ранее.

Интерактивная консоль

Ранее мы видели, как платформа Django обеспечивает доступ к оболочке Python в процессе разработки. Несмотря на то что в системе App Engine точно такая возможность отсутствует, вы можете получить аналогичный доступ. Щелкните на ссылке `Interactive Console`, размещенной слева в навигационных ссылках на консоли SDK, чтобы попасть на веб-страницу, слева на которой находится панель кодирования, а справа — выходные данные. Отсюда вы можете вводить любые команды Python и наблюдать за их выполнением. Пример выполнения представлен на рис. 12.16.

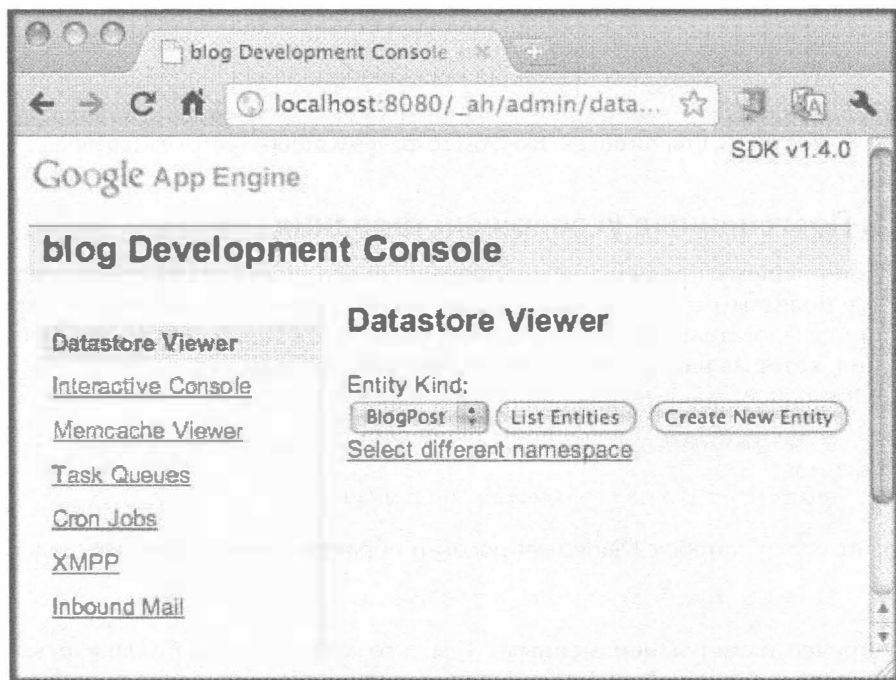


Рис. 12.14. Datastore Viewer на консоли SDK App Engine

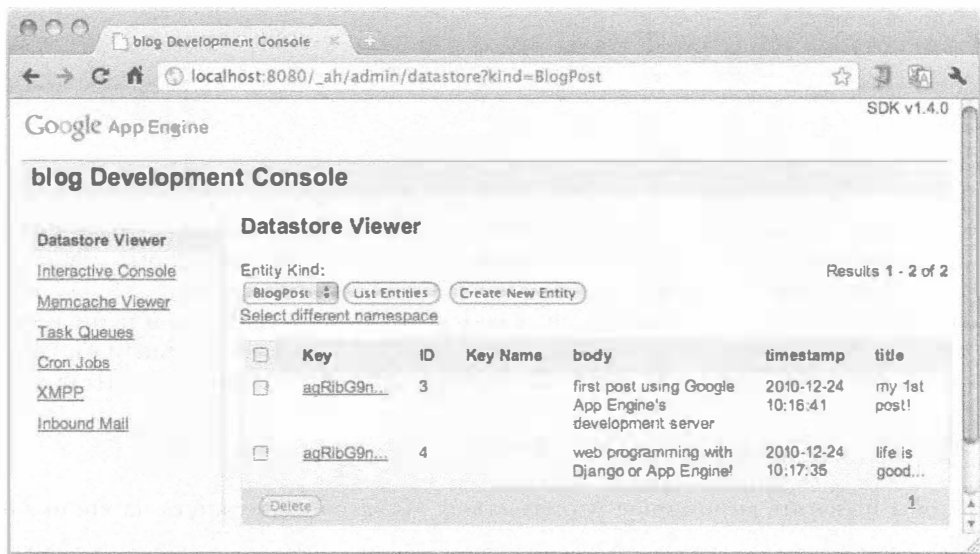


Рис. 12.15. Просмотр существующих объектов BlogPost

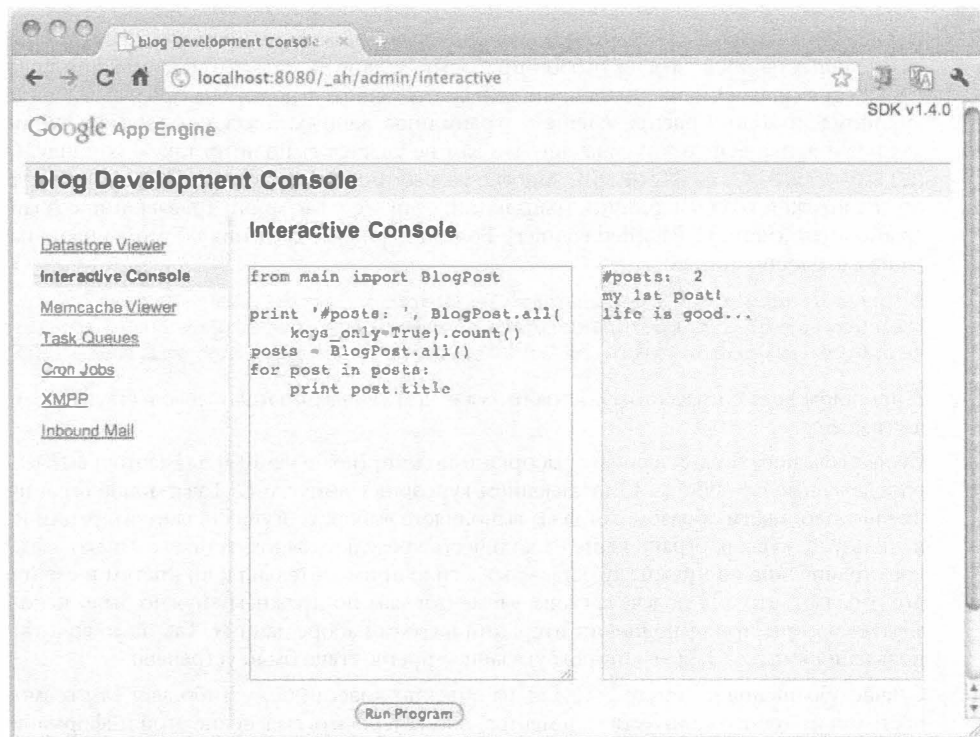


Рис. 12.16. Выполнение кода на интерактивной консоли

Код, выполняемый на интерактивной консоли, достаточно прост, как показано в приведенном ниже сценарии.

```

from main import BlogPost

print '#posts: ', BlogPost.all(keys_only=True).count()
posts = BlogPost.all()
for post in posts:
    print post.title

```

Как было сказано выше, этот фрагмент кода достаточно прост. Однако определенный интерес для вас может представлять первый оператор печати, который отображает текущее совокупное количество объектов BlogPost в (локальном) хранилище данных. У вас может возникнуть мысль об использовании BlogPost.all(), но он возвращает объект Query, который не является последовательностью и не подменяет собой len(), поэтому вы не можете вызвать len() на нем. Единственным вариантом для вас является метод count(), который вы получите, обратившись на сайт по адресу

[http://code.google.com/appengine/docs/python/datastore/
queryclass.html#Query_count](http://code.google.com/appengine/docs/python/datastore/queryclass.html#Query_count)

Чтобы испытать мгновенное удовольствие, достаточно щелкнуть на кнопке Run Program.



Подсчет (или отсутствие такового)

Несмотря на то что подсчет в случае использования платформы Django и реляционной базы данных не представляет особой проблемы, нужно сказать, что система App Engine плохо справляется с этой задачей, поскольку в действительности она предназначена для крупномасштабного распределенного хранилища данных. Здесь нет никаких таблиц, как нет и языка SQL, а это означает, что вам не удастся выполнить такую команду, как SELECT COUNT(*), из BlogPost. Многие разработчики, приложениям которых не требуется подсчет, создают счетчик транзакций; если же у вас много транзакций, создайте “расколотый счетчик” (sharded counter). Более подробные сведения об этом можно найти на следующих сайтах:

http://code.google.com/appengine/articles/sharding_counters.html
http://code.google.com/appengine/docs/python/datastore/queriesandindexes.html#Query_Cursors
http://googleappengine.blogspot.com/2010/08/multi-tenancy-support-high-performance_17.html

В прошлом дела с подсчетом обстояли хуже, чем сейчас, поэтому довольствуйтесь тем, что имеете.

Существовавшее ограничение на выборки и подсчет (не более 1000 элементов) вызывало определенные неудобства. С добавлением курсоров в выпуске 1.3.1 указанное ограничение исчезло; таким образом, когда вы выполняете выборку, осуществляете итерации или используете курсор, ограничение на количество результатов отсутствует. Однако указанное ограничение по-прежнему оставалось в силе применительно к подсчетам и сдвигам; это означает, что для подсчета своих элементов вам по-прежнему нужно было использовать курсоры при выполнении итераций в своем наборе данных. Так было вплоть до появления выпуска 1.3.6, в котором указанное препятствие было устранено.

Сейчас обращение к методу count() на объектах класса Query либо дает вам возможность узнать точное количество элементов, либо отсрочить выяснение этой информации. Как указывается в документации по методу count(), вам не следует использовать этот метод для подсчета большого количества элементов: “Лучше всего использовать метод count() лишь в случаях, когда ожидаемое количество является небольшим, или указать определенный лимит. Для метода count() максимальный лимит не предусмотрен. Если вы не указываете лимит, Datastore продолжает подсчет до тех пор, пока он не будет

завершен, или до исчерпания лимита времени (тайм-аута)". Разумеется, здесь не могут быть учтены все ваши пожелания, но это, несомненно, — заметное улучшение по сравнению с тем, что было доступно разработчикам в App Engine до начала 2010 г.

Практика показывает, что подсчетом увлекаться не следует, но если вы все же выполняете подсчет, то нужно надлежащим образом организовать функционирование счетчика. В том, что касается хранилища данных App Engine, вам нужно выработать определенный образ мышления. Взамен некоторых функций, которыми вы, возможно, привыкли пользоваться и которые кажутся вам довольно удобными, вы получаете репликацию и масштабируемость — две возможности, практическая реализация которых требует больших затрат.

Еще одна рекомендация для тех, кто не может обойтись без подсчета: отдавайте предпочтение подсчету "только с ключами". Иными словами, когда вы создаете объект типа запроса, передавайте на True набор флагов `key_only`, чтобы вам не приходилось полностью выбирать элементы из хранилища данных, например `BlogPost.all(keys_only=True)`. Ниже приведены ссылки, которые помогут вам решить эту проблему.

<http://code.google.com/appengine/docs/python/datastore/queryclass.html#Query>
http://code.google.com/appengine/docs/python/datastore/modelclass.html#Model_all
http://code.google.com/appengine/docs/python/datastore/queriesandindexes.html#Queries_on_keys

Наконец, команда разработчиков App Engine опубликовала ряд статей, которые помогут вам получить всю необходимую информацию о Datastore. Ищите эти статьи по адресу

<http://code.google.com/appengine/articles/datastore/overview.html>

Следует помнить и о том, что код, который вы выполняете на интерактивной консоли, имеет непосредственный доступ к вашему локальному хранилищу данных. Подобно нашему примеру с блогом на платформе Django, используйте фрагмент Python для автоматического генерирования дополнительных элементов, как показано в приведенном ниже коде для рис. 12.17.

```
from datetime import datetime
from main import BlogPost

for i in xrange(10):
    BlogPost(
        title='post #%d' % i,
        body='body of post #%d' % i,
        timestamp=datetime.now()
    ).put()
    print 'created post #%d' % i
```

Рис. 12.18 демонстрирует, что сейчас у нас есть возможность выполнять сортировку в обратной последовательности с помощью отметки времени и видеть два исходных объекта `BlogPost`, а также десять объектов, только что сгенерированных нами на рис. 12.17.

```
from main import BlogPost

print '#posts: ', BlogPost.all(
    keys_only=True).count()
posts = BlogPost.all().order(
    '-timestamp')
for post in posts:
    print post.title
```

Вы можете даже вернуться к приложению Datastore Viewer, чтобы увидеть более подробную информацию о каждом элементе (рис. 12.19).

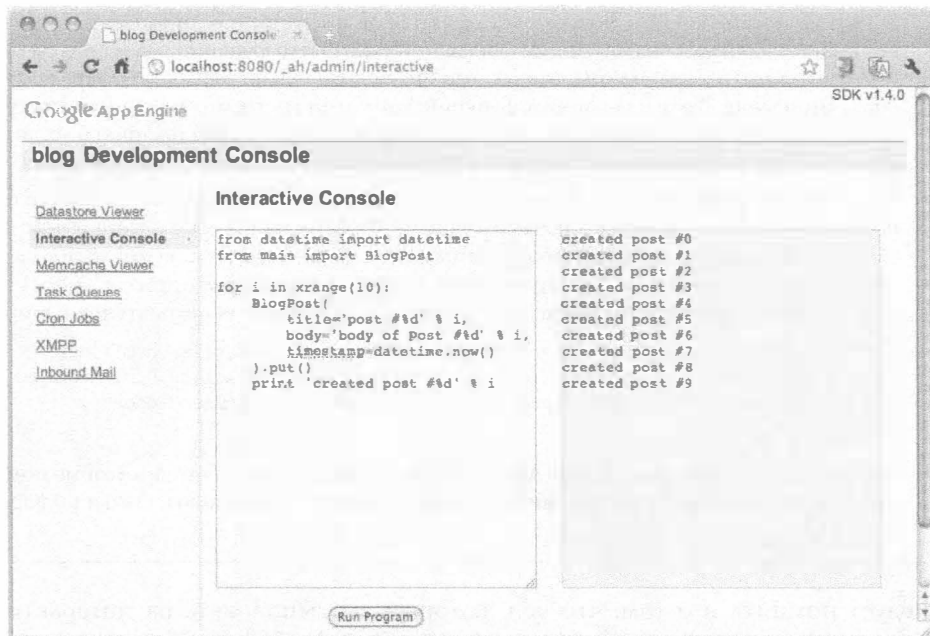


Рис. 12.17. Создание дополнительных элементов с помощью Python

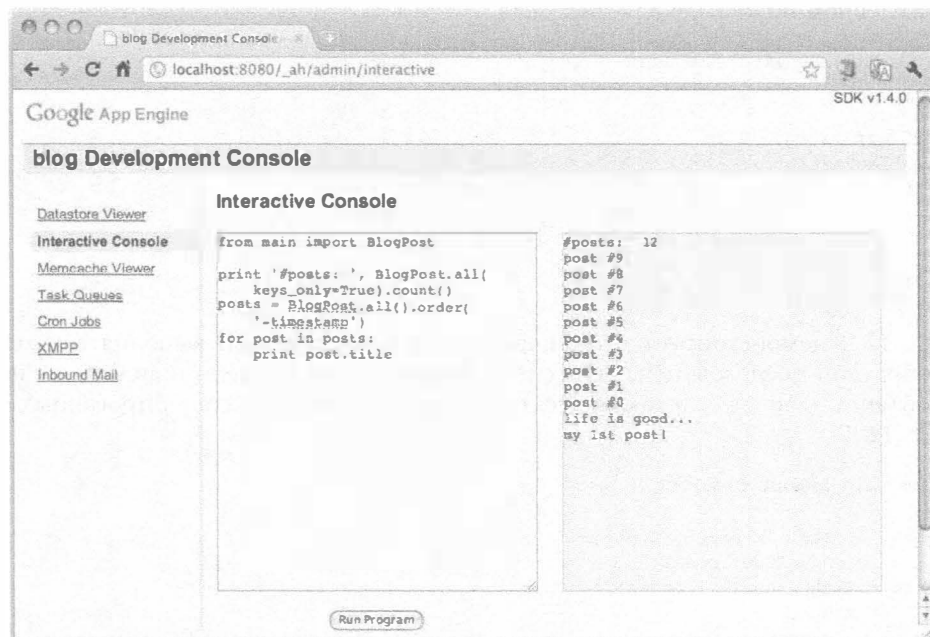


Рис. 12.18. Совместное отображение старых и новых элементов

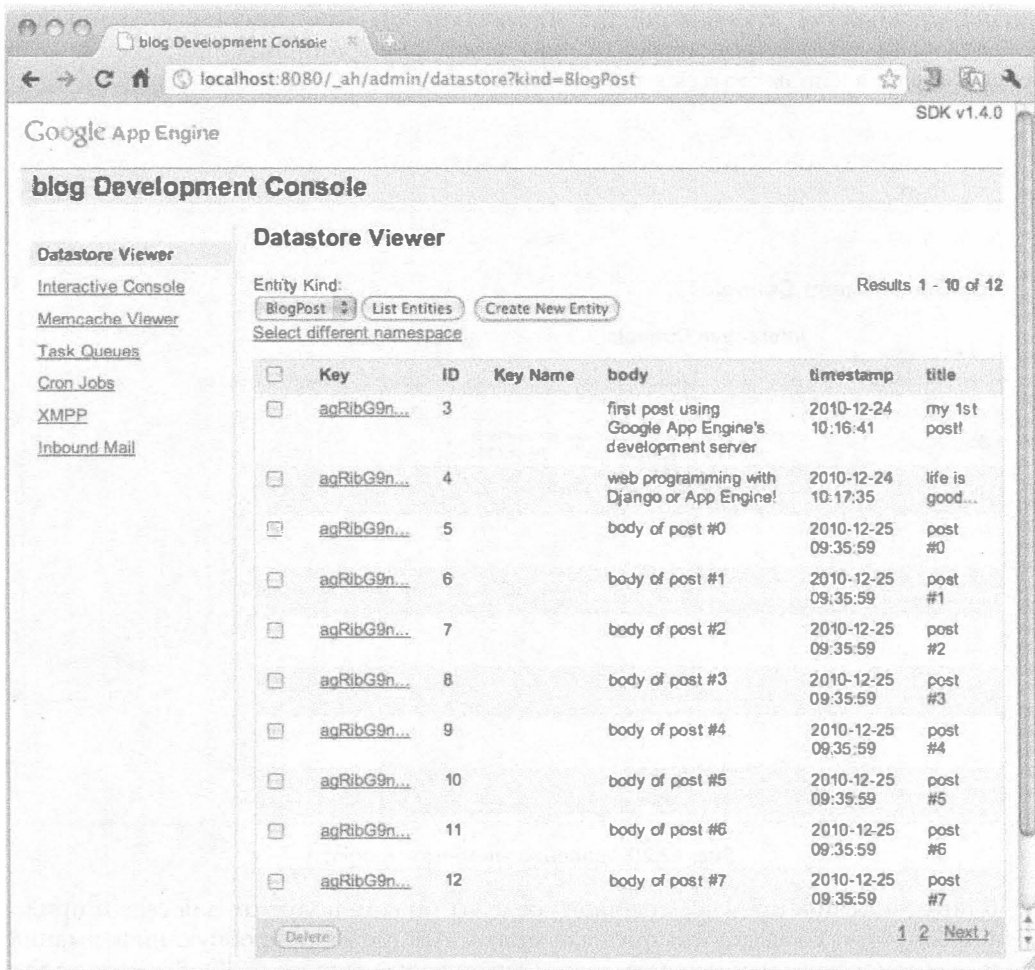


Рис. 12.19. Изменение порядка отображения элементов с помощью интерактивной консоли

Если вы не хотите засорять свои данные этими фиктивными входами BlogPost, удалите их с помощью приведенного ниже фрагмента кода, выполнение которого представлено на рис. 12.20 (после возврата на интерактивную консоль).

```
from google.appengine.ext import db
from main import BlogPost
```

```
posts = BlogPost.all(keys_only=True
    ).order('-timestamp').fetch(10)
db.delete(posts)
print 'DELETED newest 10 posts'
```

Если вы вырежете и вставите этот фрагмент “дампа данных”, то впоследствии сможете подтвердить, что удаление действительно сработало.

То, что мы можем выполнить все эти действия в ходе разработки, просто замечательно. В какой-то момент вы наверняка захотите обеспечить аналогичную

функциональность в каком-нибудь интерактивном приложении и хранилище Datastore в производственной среде. Существуют два аналогичных инструмента, которыми можно воспользоваться в таком случае.

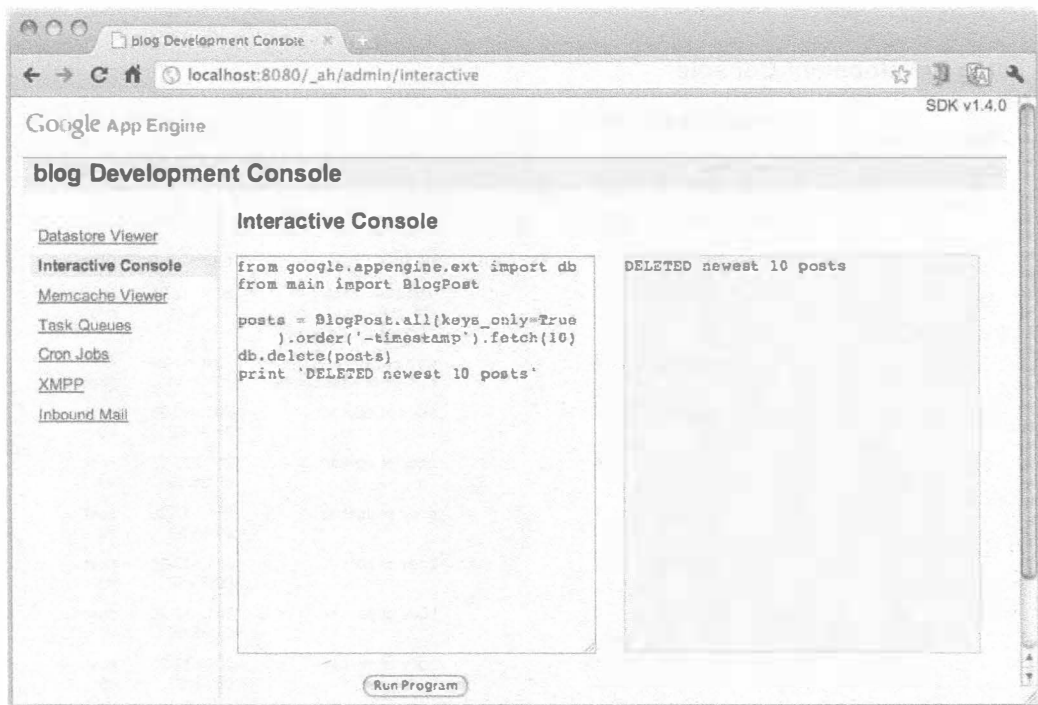


Рис. 12.20. Удаление элементов BlogPost

В интерактивной производственной среде вы можете получить для своего приложения оболочку, воспользовавшись удаленным API (более подробную информацию по этому вопросу вы найдете в разделе “Оболочка удаленного API”). Вы можете также выполнять групповое удаление или копирование элементов в какое-либо другое приложение App Engine, если активизируете приложение Datastore Admin для своей административной консоли.

Итак, вы ознакомились с кратким введением в консоль SDK. Разумеется, она не обладает столь широкими возможностями, как ее “интерактивный” аналог, административная консоль. Тем не менее она является весьма полезным инструментом разработки. Вскоре мы вернемся к консоли SDK, но сначала добавим в наше приложение еще одну службу: кеширование.

12.8. Добавление службы Memcache

Пользователи, которые лишь недавно приступили к работе с системой App Engine, нередко жалуются на слишком медленный доступ к базе данных App Engine. Конечно, определение “медленный” — весьма относительно, но вы наверняка обратите внимание на снижение производительности по сравнению с использованием стандартной реляционной базы данных. Однако при этом не следует забывать, что

в данном случае вам приходится идти на компромисс: в обмен на распределенное, расширяемое и дублируемое хранилище в облаке вы получаете небольшое снижение производительности, поскольку, как известно каждому из нас, за все приходится платить. Один из способов повысить скорость обработки запросов заключается в том, чтобы перенести данные “поближе” к вашему приложению путем кеширования, вместо того чтобы обращаться к хранилищу данных.

Пропускная способность сайтов с высоким уровнем трафика редко бывает ограничена скоростью, с которой соответствующий веб-сайт может пересылать данные клиенту. Узким местом почти всегда является генерирование этих данных: возможно, база данных не успевает отвечать на запросы достаточно быстро, или центральный процессор сервера “увяз” в многократном выполнении одного и того же кода по каждому запросу. К тому же извлечение одних и тех же данных или вычисления, производимые с одними и теми же данными, для многих запросов является не чем иным, как неэффективным использованием ресурсов.

Размещая данные на более высоком уровне и ближе к месту, из которого исходит запрос, требуется меньше “усилий” со стороны базы данных или кода, который генерирует результаты, возвращаемые клиенту. Промежуточный кеш является идеальным местом для временного хранения извлекаемых данных. Таким образом, в случае поступления идентичных запросов со стороны разных клиентов последним можно многократно отправлять одни и те же данные, не прибегая к повторным выборкам или вычислениям при обслуживании разных пользователей. Это особенно важно для пользователей App Engine, если оказывается, что ваше приложение при выполнении разных запросов снова и снова обращается к одним и тем же элементам.

Общая схема действий для кеширования объектов (будь то в системе App Engine или где-либо еще) выглядит так: проверить, содержит ли кеш требуемые данные. Если да, вернуть их; в противном случае выполнить поиск и кешировать его результаты.

Если бы вам нужно было отобразить такую схему действий в виде псевдокода, то он мог бы выглядеть примерно так, как показано в приведенном ниже фрагменте для некоего постоянного KEY, который мы используем для сохранения кешированных данных.

```
data = cache.get(KEY)
if not data:
    data = QUERY()
    cache.set(KEY, data)
return data
```

Неудивительно, что такое решение очень удачно реализуется на языке Python. Нам не хватает лишь значения для KEY, базы данных QUERY и этого импортирования API нижнего уровня, совместимого со службой Memcache:

```
from google.appengine.api import memcache
```

Мы добавляем в код нашего приложения несколько строк для метода Main Handler.get(), который обрамляет извлечение данных, обращаясь к хранилищу данных лишь в случае, если мы еще не кешировали соответствующий набор данных:

До:

```
...
posts = BlogPost.all().order('-timestamp').fetch(10)
for post in posts:
```


После:

```
...
posts = memcache.get(KEY) # сначала проверяем кеш
if not posts:
    posts = BlogPost.all().order('-timestamp').fetch(10)
    memcache.add(KEY, posts) # кешируем объект
for post in posts:
    ...
```

Не забудьте задать для своего кеша ключ, т.е. `KEY = 'posts'`.

С помощью вызова `add()` мы фактически кешировали соответствующий объект либо до того момента, когда удалим его (см. ниже), либо до того момента, когда исключим его, чтобы освободить место для более свежих данных (т.е. данных, извлеченных позднее). Между прочим, интерфейс API Memcache использует алгоритм LRU (Least Recently Used — алгоритм замещения блока данных с наиболее длительным отсутствием обращений). Третьей альтернативой является кеширование объекта с указанием определенного “срока годности”. Если бы, например, мы хотели кешировать этот объект на одну минуту, то изменили бы свое обращение таким образом:

```
memcache.add(KEY, posts, 60)
```

Последним элементом этой мозаики является объявление кеша недействительным при появлении нового входа (сообщения) в блоге. Для этого нам придется полностью очищать (сбрасывать) кеш каждый раз, когда в нашем коде для `BlogEntry.post()` в хранилище данных отправляется новая точка входа:

```
...
post.put()
memcache.delete(KEY)
self.redirect('/')
```

После внесения указанных изменений можете смело испытывать этот код в своем браузере. Впрочем, поскольку наш набор данных очень мал, вам будет весьма затруднительно определить, откуда именно вы получаете свои данные, от службы Memcache или из хранилища данных. Проще всего сделать это, взглянув на приложение Memcache Viewer на консоли SDK (рис. 12.21).

Чтобы увидеть этот механизм в действии, вам понадобится пара окон браузера: одно, открытое для вашего приложения, другое — для приложения Memcache Viewer на консоли SDK. Позаботьтесь о том, чтобы у вас в приложении было несколько объектов `BlogPost`, а затем несколько раз обновите главную страницу своего приложения. Теперь обновите страницу Memcache Viewer, чтобы оценить коэффициент использования службы Memcache. Я выполнил эти операции, чтобы вы могли увидеть мои результаты использования службы Memcache, представленные на рис. 12.22.

Вы, наверное, обратили внимание не только на один “промах” кеша, но и на возросшее количество “попаданий” в каждом последующем проходе; это означает, что обращение к хранилищу данных случилось только в результате первого запроса. Таким образом, указанные изменения в коде помогли пользователям сократить время получения требуемых данных после первоначального получения данных. Тем, кто хочет узнать больше об использовании API Memcache в App Engine, рекомендуем обратиться к странице документации на сайте <http://code.google.com/appengine/docs/python/memcache>.

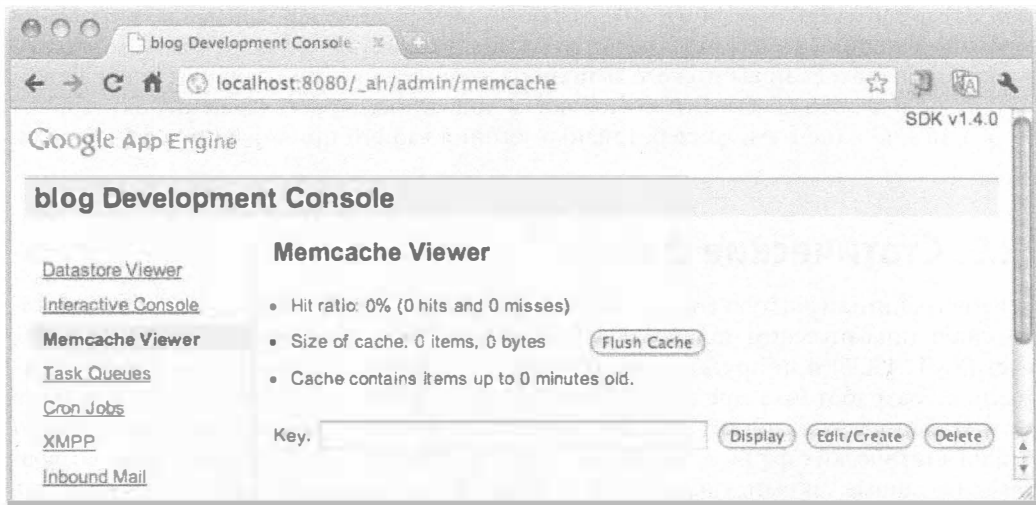


Рис. 12.21. Приложение Memcache Viewer; в данном случае мы видим пустой кеш

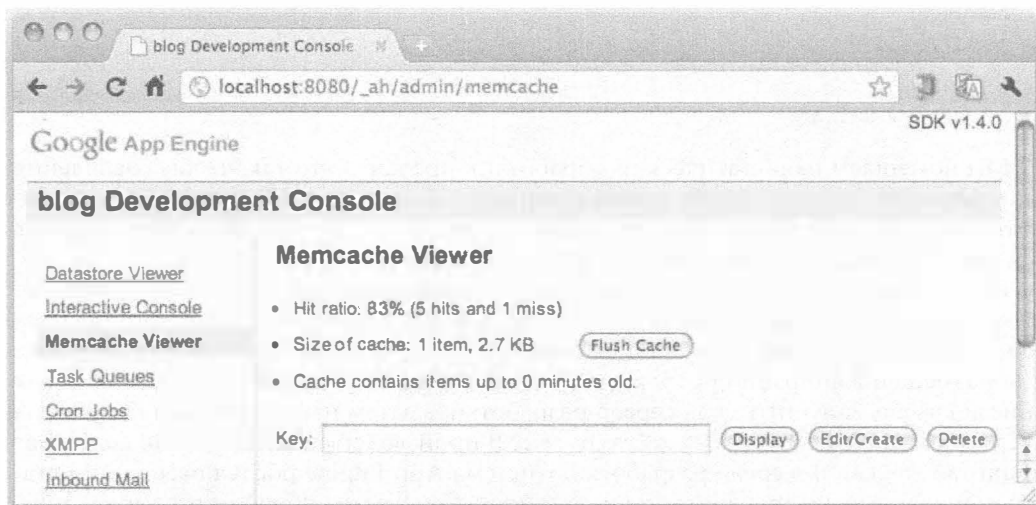


Рис. 12.22. В данном случае приложение Memcache Viewer зафиксировало незначительное использование кеша

В главе 11 у нас не было возможности поговорить о кешировании. На платформе Django есть много уровней службы кеширования, в том числе кеширование объектов (именно то, что мы сейчас реализовали), плюс кеширование QuerySet, которое помогает “затолкать” низкоуровневое кеширование объектов еще дальше от взоров пользователей. Подробнее о различных типах кеширования, предусмотренных в платформе Django, можно узнать в главе 12 книги *Python web Development with Django*.

Кеширование объектного уровня — это лишь один из способов избавить сервер от выполнения лишней работы по доставке данных в ваш компьютер. Однако данные не всегда поступают из базы данных. Обслуживание веб-страниц обычно включает также много статических файлов. В этом отношении система App Engine обеспечивает

разные варианты оптимизации для разработчиков, например, запрос кеширования “вверх по течению” путем использования заголовков HTTP Cache-Control в соответствующих местах. Если вы можете выполнять кеширование на краю или посредством “заместителей”, это обеспечит возможность подачи некоторых из ваших активов непосредственно клиентам, даже без использования вашего приложения в службе App Engine.

12.9. Статические файлы

Веб-страницы зачастую содержат статические элементы, которые сопутствуют какому-либо динамическим данным. К их числу относятся изображения, элементы CSS, текст (XML, JSON или другая разметка) и файлы JavaScript. Вместо того чтобы требовать от разработчика предоставления соответствующих программ обработки для обслуживания таких данных, укажите в своей конфигурации `app.yaml` определенный каталог статических файлов, предложив таким образом службе App Engine возвращать эти данные такими, как они есть. В этом случае от вас лишь требуется добавить в раздел `handlers` своей конфигурации `app.yaml` конкретную программу обработки. Это будет выглядеть примерно так:

```
handlers:
- url: /static
  static_dir: static
- url: .*
  script: main.py
```

Мы помещаем наш статический обработчик прежде всего так, чтобы совпадения запросов пути `/static` обрабатывались первыми. Всеми другими путями будут заниматься обработчики в `main.py`. Это означает, что вам нет необходимости выполнять код приложения, чтобы обслужить статические файлы.

Более того, почему бы вам просто не найти какие-то случайные файлы с расширением `.js`, `.css` или какой-либо другой имеющийся у вас статический контент (скажем, `main.css`), создать папку под именем “static” прямо в каталоге верхнего уровня (где размещен ваш файл `app.yaml` и `main.py`), обновить свой файл `app.yaml`, как описано выше, запустить свой сервер разработки, а затем направить свой браузер на `http://localhost:8080/static/main.css?` В производственной среде это сработает точно так же, как и в среде разработки. Система App Engine обслуживает ваши статические данные, не требуя помощи со стороны программ обработки в вашем приложении.

12.10. Добавление службы Users

В главе 11, где речь шла о нашем Django-блоге, мы не добавили никакой идентификации (пользователи, пароли, учетные записи и т.п.), но воспользовались собственной системой идентификации Django в приложении TweetApprove. Аналогично мы можем выполнить идентификацию в этом блоге, воспользовавшись службой Google Accounts. Это, конечно, лучше, чем предоставлять возможность *любому* пользователю, посещающему вашу страницу, добавлять новые сообщения в блог; если бы мы допустили такую возможность, то наш блог стал бы похож на гостевую книгу, т.е. место на веб-сайте, где посетители могут оставлять свои сообщения, отзывы, замечания

и пожелания. Выполнение идентификации в данном случае не должно восприниматься пользователями как неприятный сюрприз. Допустим, вы хотите создать еще один отраслевой блог наподобие TechCrunch, Engadget и т.п. Этот блог должен поддерживать нескольких авторов, и вы хотите, чтобы никто, кроме этих авторов, не мог оставлять сообщения в данном блоге.

12.10.1. Идентификация с помощью службы Google Accounts

Когда вы создаете свое приложение в приложении App Engine, то по умолчанию идентификация выполняется с помощью службы Google Accounts. Однако если вы не дополните свое приложение каким-либо механизмом идентификации — либо в параметрах конфигурации, либо в фактическом коде приложения, — то это будет практически то же самое, что отказ от идентификации как таковой: любой желающий сможет оставлять сообщения в вашем блоге. Давайте добавим проверку идентификации, вставив пару строк в самом начале метода `MainHandler.get()`, в результате чего он примет следующий вид:

```
...
from google.appengine.api import users
...
class MainHandler(webapp.RequestHandler):
    def get(self):
        user = users.get_current_user()
        if user:
            self.response.out.write('Hello %s' % user.nickname())
        else:
            self.response.out.write('Hello World! [

```

Если вы не хотите добавлять особый код, предлагающий пользователям зарегистрироваться в системе (подобно тому, как мы сделали выше), реализуйте эту функцию на уровне конфигурации `app.yaml`. Просто добавьте директиву `login`:

required; любой URL, который обращается к этой программе обработки, заставит пользователя зарегистрироваться, прежде чем он сможет получить доступ к вашему приложению или его содержимому. Ниже приведен пример того, как использовать эту директиву для блокирования доступа к вашей главной программе обработки, если пользователь не задаст требуемое регистрационное имя в Google Accounts:

```
- url: .*
script: main.py
login: required
```

Еще одной альтернативой является задание директивы `login: admin`. В таком случае доступ к этой программе обработки возможен только со стороны зарегистрировавшегося администратора, в роли которого может выступать какой-либо VIP-пользователь, приложение, а также средство доступа к данным или обработки данных. Пользователи, которые не являются администраторами, увидят на своем экране страницу ошибки, гласящую, что требуется доступ со стороны администратора. Подробнее об этих директивах можно прочитать на сайте http://code.google.com/appengine/docs/python/config/appconfig.html#Requiring_Login_or_Administrator_Status.

12.10.2. Объединенная идентификация

Если вы испытываете какие-либо проблемы с созданием своей собственной идентификации или не хотите требовать от всех своих пользователей, чтобы они обзавелись собственными учетными записями в службе Google Accounts, тогда вас, наверное, устроит объединенное регистрационное имя в службе OpenID. При использовании службы OpenID вы можете позволить пользователям регистрироваться в вашем приложении с помощью учетных записей, которые они создали у разных провайдеров, в том числе (но не только) на сайтах Yahoo!, Flickr, Word-Press, Blogger, LiveJournal, AOL, MyOpenID, MySpace и даже Google.

Если вы используете объединенное регистрационное имя, то вам придется внести в ваш вызов небольшое изменение, которое создает ссылки на регистрационное имя, добавляя параметр `federated_identity`, такой, как `users.create_login_url(federated_identity=URL)`, где `URL` — любой из поставщиков OpenID (`gmail.com` (Google), `yahoo.com`, `myspace.com`, `aol.com` и т.д.). Будущая поддержка, оказываемая объединенной идентификации, будет интегрирована с новым Google Identity Toolkit (GIT).

Получить более подробную информацию о службах Users, GIT и OpenID можно, воспользовавшись следующими ссылками:

```
http://code.google.com/appengine/docs/python/users/overview.html
http://code.google.com/appengine/articles/openid.html
http://openid.net
http://code.google.com/apis/identitytoolkit/
```

12.11. Оболочка удаленного API

Для того чтобы воспользоваться оболочкой удаленного API, добавьте в свой файл `app.yaml` (непосредственно над программами обработки для вашего приложения) следующий вход:

```
- url: /remote_api
  script: $PYTHON_LIB/google/appengine/ext/remote_api/handler.py
  login: admin

- url: .*
  script: main.py
```

Если у вас здесь предусмотрен какой-то другой раздел для статических файлов (как мы поступили в предыдущем разделе), не имеет значения, какой порядок следования используется при создании конфигурации программы обработки для удаленного API. Здесь важно лишь то, что оба они расположены над главной программой обработки. В предыдущем примере мы проигнорировали все, что относится к статическому файлу, и добавили в явном виде регистрацию администратора, поскольку совершенно уверены, что вы не хотели бы, чтобы кто-либо из посторонних пользователей имел доступ к вашему хранилищу данных производственной среды.

Вам понадобится локальная версия модели (моделей) данных вашего приложения. Когда вы находитесь в требуемой директории, выдайте следующую команду (подставив вместо ID ваше приложение интерактивной производственной среды) и укажите информацию, подтверждающую ваше право на доступ:

```
$ remote_api_shell.py APP_ID
Email: YOUR_EMAIL
Password: *****
App Engine remote_api shell
Python 2.5.1 (r251:54863, Feb 9 2009, 18:49:36)
[GCC 4.0.1 (Apple Inc. build 5465)]
The db, users, urlfetch, and memcache modules are imported.
APP_ID> import sys
APP_ID> sys.path.append('.')
APP_ID> from main import *
APP_ID> print Greeting.all(keys_only=True).count()
24
```

Оболочка удаленного API предоставляет в ваше распоряжение интерактивный интерпретатор Python для выполнения вашего приложения в интерактивном режиме. Существует много других применений самого по себе удаленного API, самыми важными из которых являются массовая выгрузка данных из хранилища данных вашего приложения и загрузка данных в это хранилище. Подробнее об использовании удаленного API можно прочитать в официальной документации, выложенной на сайте http://code.google.com/appengine/articles/remote_api.html.

12.11.1. Функция Datastore Admin

Функция Datastore Admin — это сравнительно недавно реализованная функция, которая добавляет соответствующий компонент на административную консоль (но не на консоль сервера разработки SDK) вашего интерактивного приложения. Она позволяет вам выполнять групповое удаление определенных типов элементов (или всех их); кроме того, позволяет копировать элементы в другое интерактивное приложение. Единственное предостережение: во время копирования ваше приложение должно находиться в режиме “только чтение”. Для того чтобы активизировать функцию Datastore Admin, добавьте следующий раздел в свой файл `app.yaml`:

```
builtins:
- datastore_admin: on
```

Вам вовсе необязательно запоминать это, поскольку все, что от вас требуется, — это щелкнуть на ссылке Datastore Admin на своей административной консоли. Если вы еще не активизировали ее, она предупредит вас, что данная конфигурация отсутствует в вашем файле `app.yaml`.

После того как вы активизируете ее, щелчок на ссылке Datastore Admin приведет к появлению экрана (или двух экранов) регистрации, после чего вы должны увидеть на экране что-то вроде того, что показано на рис. 12.23.

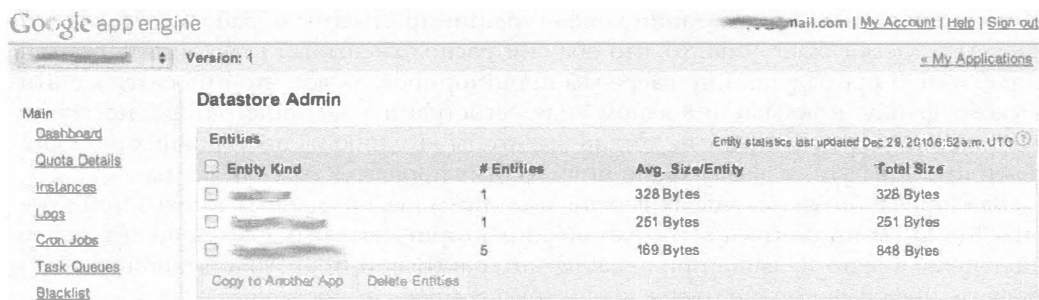


Рис. 12.23. Пример экрана Datastore Admin в App Engine

Пример файла `app.yaml` с активизированной функцией Datastore Admin, а также файла `appengine_config.py`, необходимого для того, чтобы позволить другим приложениям копировать элементы в текущее приложение, приведен в хранилище образцов кода на сайте http://code.google.com/p/google-app-engine-samples/source/browse/#svn%2Ftrunk%2Fdatastore_admin.

Подробнее о Datastore Admin и ее возможностях можно прочитать на сайтах:

- <http://code.google.com/appengine/docs/adminconsole/datastoreadmin.html>
- <http://googleappengine.blogspot.com/2010/10/new-app-enginesdk-138-includes-new.html>

12.12. Быстрый обзор (с использованием кода на языке Python)

Учитывая огромные возможности, заложенные в платформе App Engine в целом, не было бы ничего удивительного в том, если бы этой теме была посвящена отдельная книга. Однако поскольку наша цель заключается в том, чтобы просто ввести вас в курс дела, а затем предоставить возможность самостоятельно вникать в подробности, будем считать на этом свою задачу выполненной. Прежде чем перейти к следующей главе, мы хотели бы привести несколько кратких примеров кода (которыми вы можете воспользоваться в своей последующей практической работе), причем функции, реализуемые в этих примерах, необязательно включать в наше блог-приложение. Разумеется, эти примеры кода будут отражены в упражнениях, помещенных в конце данной главы.

12.12.1. Отправка электронной почты

В нашем приложении Twitter/Django из главы 11 мы показали, как пользоваться службой электронной почты в Django. Отправить электронную почту на платформе App Engine так же просто, как и на платформе Django. Все, что от вас требуется, — это импортировать функцию `mail.send_mail()` и воспользоваться ею. Основной принцип пользования ею очень прост: `mail.send_mail(ОТ, КОМУ, ТЕМА, ТЕЛО)`, где:

| | |
|-------------|----------------------------------------------------------------------------------------------|
| <i>ОТ</i> | Строка, представляющая адрес электронной почты отправителя (подробнее об этом поле см. ниже) |
| <i>КОМУ</i> | Либо строка, либо совокупность строк, представляющих получателя (получателей) почты |
| <i>ТЕМА</i> | Строка, организованная как часть строки Subject: |
| <i>ТЕЛО</i> | Строка, представляющая содержание соответствующего сообщения (обычный текст) |

Есть и другие поля сообщения, которые вы можете передать в функцию `send_mail()`; подробнее об этих полях можно прочитать на сайте <http://code.google.com/appengine/docs/python/mail/emailmessagefields.html>.

Для того чтобы уберечь пользователей от соблазна рассылать нежелательную электронную почту, в качестве адреса From: можно указывать лишь один из следующих вариантов:

- адрес электронной почты зарегистрированного администратора (разработчика) соответствующего приложения;
- текущий пользователь, если он зарегистрировался;
- любой допустимый принимающий адрес электронной почты для соответствующего приложения (в форме `xxx@APP_ID.appspotmail.com`).

Ниже приведен фрагмент кода, который включает импорт и один из возможных вариантов вызова `send_mail()`.

```
from google.appengine.api import mail
...
mail.send_mail(
    user and user.email() or 'admin@APP_ID.appspotmail.com', # from
    'corepython@yahoo.com', # to
    'Erratum for Core Python 3rd edition!' # subject
    "Hi, I found a typo recently. It's...", # body
)
```

В интерфейсе прикладного программирования почты предусмотрены также дополнительные функции отправки электронной почты только администратору (администраторам) соответствующего приложения, проверки правильности адресов электронной почты и т.п., а также класс `EmailMessage`. В вашей исходящей электронной почте могут также быть вложения, однако типы файлов-вложений ограничиваются лишь самыми популярными форматами, которые считаются заведомо безопасными; к их числу относятся `.doc`, `.pdf`, `.rss`, `.css`, `.xls`, `.ppt`, `.mp3/.mp4/.m4a`, `.gif`, `.jpg/.jpeg`, `.png`, `.tif/.tiff`, `.htm/.html`, `.txt` и т.п. Самый свежий перечень допустимых типов файлов-вложений вы найдете на сайте <http://code.google.com/appengine/docs/python/mail/overview.html#Attachments>.

Наконец, предусмотрено ограничение на размер входящих и исходящих сообщений: 10 Мбайт (на момент написания этой книги). Самую свежую информацию о подобных квотах и ограничениях, касающихся службы электронной почты, можно найти на следующих сайтах:

<http://code.google.com/appengine/docs/quotas.html#Mail>
http://code.google.com/appengine/docs/python/mail/overview.html#Quotas_and_Limits

Информацию более общего характера об отправке электронной почты можно найти на следующих сайтах:

http://code.google.com/appengine/docs/python/mail/overview.html#Sending_Mail_in_Python
http://code.google.com/appengine/docs/python/mail/overview.html#Sending_Mail
<http://code.google.com/appengine/docs/python/mail/sendingmail.html>

12.12.2. Получение электронной почты

Отправка почты без ее получения бесполезна. Да, ваше приложение также может обрабатывать входящую электронную почту. Это несколько сложнее, чем отправлять электронную почту, но дополнительной работы все же не так много.

Какие добавления нужно внести в код своего приложения

Помимо написания кода для обработки входящей электронной почты вам нужно внести пару дополнений в свой файл конфигурации `app.yaml`, причем самым важным из этих дополнений будет активизация почтового службы. По умолчанию получение входящей электронной почты отключено. Для того чтобы включить его, вам нужно активизировать его в разделе `inbound_services` файла конфигурации `app.yaml` (или добавить такой раздел, если у вас его еще нет).

Кроме того, выше мы уже упоминали о том, что одним из допустимых адресов, с которого вы можете принимать электронную почту, является допустимый адрес приема электронной почты для данного приложения (в форме `xxx@APP_ID.appspotmail.com`). У вас может быть одна программа обработки для всех возможных адресов электронной почты или разные программы обработки для конкретных адресов. Это можно сделать, создав одну или несколько дополнительных программ обработки в вашем файле конфигурации `app.yaml`. Для того чтобы вам было понятно, как создавать программы обработки, вы должны знать, что вся входящая электронная почта будет отправляться (с помощью POST) на тот или иной URL в форме `/_ah/mail/EMAIL_ADDRESS`.

Ниже приведены соответствующие разделы, которые вы должны добавить в свой файл `app.yaml`.

```
inbound_services:
- mail
handlers:
...
- url: /_ah/mail/.+
  script: handle_incoming_email.py
  login: admin
...
```

Первые две строки активизируют входящую электронную почту. Раздел `inbound_services`: также является тем местом, где нужно активизировать прием

XMPP-сообщений (подробнее об этом — в разделе 12.13), Warming Requests и другие будущие службы, о которых вы можете прочитать на странице официальной документации, посвященной конфигурации приложения и файлу `app.yaml` сайта http://code.google.com/appengine/docs/python/config/appconfig.html#Inbound_Services.

Вторая совокупность строк включает программу обработки входящей электронной почты, являющейся частью раздела `handlers:`. Регулярное выражение `/_ah/mail/.+` соответствует всем адресам электронной почты; однако ничто не мешает вам создать отдельные программы обработки для разных адресов электронной почты:

```
- url: /_ah/mail/sales@.+
  script: handle_sales_email.py
  login: admin
- url: /_ah/mail/support@.+
  script: handle_support_email.py
  login: admin
- url: /_ah/mail/.+
  script: handle_other_email.py
  login: admin
```

Вы можете заблокировать доступ вредоносных приложений и нежелательных пользователей к вашей программе обработки электронной почты, воспользовавшись директивой `login: admin`. Когда служба App Engine принимает сообщение электронной почты, она генерирует запросы и передает их (с помощью команды `POST`) в ваше приложение, что приводит к вызову вашей программы обработки как администратора.

Обработка входящей электронной почты

Вы можете обрабатывать электронную почту с помощью метода, предусмотренного по умолчанию; при этом программу обработки можно написать практически так же, как вы создаете какой-нибудь стандартный веб-обработчик и получаете экземпляр класса `mail.InboundEmailMessage`:

```
from google.appengine.api import mail
...
class EmailHandler(webapp.RequestHandler):
    def post(self):
        ...
        message = mail.InboundEmailMessage(self.request.body)
        ...
```

Разумеется, вам еще придется установить эту программу обработки при создании вашего `WSGIApplication`:

```
application = webapp.WSGIApplication([
    ...
    ('/_ah/email/+', EmailHandler),
    ...
], debug=True)
```

Возможной альтернативой является использование заранее определенного вспомогательного класса (`helper`), `InboundMailHandler`, содержащегося в `google.appengine.ext.webapp.mail_handlers`:

```
from google.appengine.ext.webapp import mail_handlers
...
class EmailHandler(mail_handlers.InboundMailHandler):
    def receive(self, msg):
        ...
```

Вместо того чтобы извлекать сообщение электронной почты из запроса, эта операция выполняется автоматически. Таким образом, все, что от вас требуется, — это реализовать метод `receive()`, который вызывается при поступлении сообщения. Вы также получаете ускоренный метод класса `mapping()`, который автоматически генерирует пару, направляющую почту в вашу программу обработки. Вы могли бы воспользоваться этим методом, например, так:

```
application = webapp.WSGIApplication([
    ...
    EmailHandler.mapping(),
    ...
], debug=True)
```

После того как вы получите сообщение, можете просмотреть основное тело электронной почты, будь то обычный текст или код HTML (или сочетание того и другого), а также получить доступ к любым вложениям или другим полям сообщения, таким как “Отправитель”, “Тема” и т.п. Дополнительные сведения о приеме электронной почты можно найти на следующих сайтах:

```
http://code.google.com/appengine/docs/python/mail/overview.html#Receiving_Mail_in_Python
http://code.google.com/appengine/docs/python/mail/overview.html#Receiving_Mail
http://code.google.com/appengine/docs/python/mail/receivingmail.html
```

12.13. Мгновенная отправка сообщений с помощью службы XMPP

Подобно отправке электронной почты, ваше приложение может также отправлять мгновенные сообщения (instant messages — IM) с помощью интерфейса API XMPP в системе App Engine. Аббревиатура XMPP означает eXtensible Messaging and Presence Protocol — открытый протокол обмена сообщениями и присутствия. Однако поначалу этот протокол назывался *Jabber protocol*; он был создан в конце 90-х годов и назван в честь соответствующего сообщества с открытым исходным кодом. С помощью интерфейса API XMPP в системе App Engine вы можете также, помимо отправки сообщений, принимать мгновенные сообщения, проверять, можно ли в данный момент пообщаться с соответствующим пользователем, и отправлять пользователю приглашение к диалогу. Ваше приложение не может взаимодействовать с каким-либо пользователем, если он не получил и не принял приглашение от вашего приложения.

Ниже приведен фрагмент псевдокода, который отправляет какому-либо пользователю приглашение к диалогу, в предположении, что вы правильно заполнили имя пользователя мгновенных сообщений (или Jabber ID), подставив его вместо `USER_ID`:

```
from google.appengine.api import xmpp
...
xmpp.send_invite(USER_ID)
self.response.out.write('invite sent')
```

Ниже приведен еще один фрагмент кода, который отправляет мгновенное сообщение (строка *MESSAGE*) некоему пользователю, после того как он принял ваше приглашение. Как и в предыдущем случае, вместо *USER_JID* нужно указать Jabber ID соответствующего пользователя:

```
...
if xmpp.get_presence(USER_JID):
    xmpp.send_message(USER_JID, MESSAGE)
    self.response.out.write('IM sent')
...
```

Третьей функцией службы XMPP является *get_presence()*, т.е. функция, возвращающая значение *True*, если интересующий вас пользователь находится в режиме *online* и с ним можно общаться, и значение *False*, если интересующий вас пользователь не находится в режиме *online* и с ним нельзя общаться (возможно также, что он еще не принял приглашение от вашего приложения). Подробнее об этих трех функциях, а также об API XMPP можно прочитать на следующих сайтах:

<http://code.google.com/appengine/docs/python/xmpp/overview.html>
<http://code.google.com/appengine/docs/python/xmpp/functions.html>

12.13.1. Прием при мгновенном обмене сообщениями

Организовать прием мгновенных сообщений можно так же, как прием электронной почты, т.е. в разделе *inbound_services*: файла *app.yaml*:

```
inbound_services:
- xmpp_message
```

Кроме того, подобно приему электронной почты, сообщения, поступающие в систему, передаются (посредством *POST*) платформой App Engine в ваше приложение. В этом случае используется такой URL-путь: */_ah/xmpp/message/chat*. Ниже приведен пример того, как принимать сообщения в вашем приложении при мгновенном обмене сообщениями.

```
class XMPPHandler(webapp.RequestHandler):
    def post(self):
        ...
        msg_obj = xmpp.Message(self.request.POST)
        msg_obj.reply("Thanks for your msg: '%s'" % msg_obj.body)
        ...
```

Разумеется, мы должны зарегистрировать нашу программу обработки:

```
application = webapp.WSGIApplication([
    ...
    ('/_ah/xmpp/message/chat/', XMPPHandler),
    ...
], debug=True)
```

12.14. Обработка изображений

В системе App Engine предусмотрен интерфейс API Images, с помощью которого вы можете работать с изображениями, выполняя простые преобразования, такие как поворот, зеркальное отображение, масштабирование и обрезка. Изображения могут передаваться (посредством POST) пользователем или извлекаться из хранилища данных или службы Blobstore.

Ниже приведен фрагмент кода на языке HTML, с помощью которого пользователи могут выгрузить файл изображения.

```
<form action="/pic" method=post enctype="multipart/form-data">
Upload an image:
<input type=file name=pic>
<input type=submit>
</form>
```

Приведенный ниже фрагмент кода создает пиктограмму изображения путем вызова функции `resize()` интерфейса API Images и возвращает ее в браузер:

```
from google.appengine.api import images

class Thumbnailer(webapp.RequestHandler):
    def post(self):
        thumb = images.resize(self.request.get('pic'), width=100)
        self.response.headers['Content-Type'] = 'image/png'
        self.response.out.write(thumb)
```

Ниже приведен соответствующий вход в программе обработки.

```
application = webapp.WSGIApplication([
    . . .
    ('/pic', Thumbnailer),
    . . .
], debug=True)
```

Всю информацию об интерфейсе API Images можно найти на сайте <http://code.google.com/appengine/docs/python/images/usingimages.html>.

12.15. Очереди задач (незапланированные задачи)

Задачи в системе App Engine используются для выполнения дополнительной работы, которую, возможно, придется выполнять в составе вашего приложения, но которая не требуется для генерирования ответа, отправляемого обратно пользователю. Эта вспомогательная работа может включать такие действия, как регистрация той или иной информации в журнале, создание или обновление элементов хранилища данных, отправка уведомлений и т.п.

Система App Engine поддерживает два типа задач. Первый тип называется *очередями с проталкиванием* (*Push Queues*) — это задания, которые ваше приложение создает для как можно более быстрого и одновременного выполнения. Они не допускают внешнего воздействия. Второй тип задач охватывает *очереди с выталкиванием* (*Pull Queues*); это несколько более гибкие задачи. Они также создаются вашим приложением в системе App Engine, однако могут использоваться, или “арендоваться”,

вашим приложением в системе App Engine или каким-либо внешним приложением посредством интерфейса прикладного программирования с переносом репрезентативного состояния (representational state transfer application programmers interface — REST API).

Следующий раздел мы в основном посвятим обсуждению задач очередей с про-талкиванием, а завершим его кратким обсуждением очередей с выталкиванием.

12.15.1. Создание задач

Задачи могут инициироваться обработчиком запроса пользователя или создаваться какой-то другой задачей. Примером последнего варианта может быть ситуация, когда всю работу, выполняемую первой задачей, не удалось завершить своевременно (например, за 30 секунд или 10 минут), в результате чего работа, для выполнения которой была создана первая задача, оказывается еще не завершенной.

Задачи добавляются в очереди задач. Очередям присваиваются имена; очереди могут иметь разные темпы выполнения, темпы *пополнения* или *нарастания*, а также параметры повторения. Пользователи получают одну очередь по умолчанию, но должны указывать другие, если понадобится больше. Добавить задачу к очереди по умолчанию довольно просто: для этого требуется лишь один простой вызов, если к этому времени вы уже импортировали API Taskqueue:

```
from google.appengine.api import taskqueue
taskqueue.add()
```

Все запросы очереди будут переданы (посредством POST) в указатель URL и, следовательно, в соответствующую программу обработки. Если пользователь не создал специализированный URL, запросы будут поступать в указатель URL, предусмотренный по умолчанию, с указанием имени очереди: `/_ah/queue/QUEUE_NAME`. Таким образом, для очереди по умолчанию это будет `/_ah/queue/default`. Это означает, что при создании объекта класса `WSGIApplication` вы должны указать параметр программы обработки:

```
def main():
    run_wsgi_app(webapp.WSGIApplication([
        . . .
        ('/_ah/queue/default', DoSomething),
        . . .
    ]))
```

Разумеется, вам понадобится также код для фактической задачи; например, определенная нами программа обработки `DoSomething`:

```
class DoSomething(webapp.RequestHandler):
    def post(self):
        # do the task here
        . . .
        logging.info('completed task')
```

В конце кода мы добавили вход оперативной регистрации в журнале событий, чтобы подтвердить фактическое завершение задачи. Очевидно, вам необязательно регистрировать то или иное событие, если оно кажется вам неактуальным, однако это может быть эффективным способом подтверждения факта завершения задачи. Более того, такой вход в журнал регистрации событий можно даже использовать в качестве

“заполнителя”, если вы еще не завершили код, предназначенный для выполнения работы фактической задачи. (Разумеется, если вы все же решите регистрировать в журнале те или иные события, позаботьтесь о наличии оператора `import logging`, который должен быть расположен где-то выше.)

Конфигурирование файла `app.yaml`

Что касается конфигурирования, то в вашем файле `app.yaml` можно оставить лишь программу обработки, предусмотренную по умолчанию; этот обработчик будет использоваться для всех URL:

```
handlers:
- url: .*
  script: main.py
```

Такая конфигурация будет направлять указатели URL обычного приложения на главный обработчик `main.py`, однако данная модель также соответствует `/_ah/queue/default`; это означает, что запросы очереди задач также будут отправляться именно сюда, а это, возможно, окажется именно тем, что вам и требуется. Однако проблема с такой конфигурацией заключается в том, что любой посторонний пользователь сможет зайти на ваш адрес `/_ah/queue/default`, даже если он не был создан как задача.

Оптимальным вариантом в таком случае является ограничение этого URL запросами, относящимися только к задачам, путем добавления директивы `login: admin`, как мы поступили ранее, когда конфигурировали приложение на прием электронной почты. Вам придется отделить этот особый URL от всех остальных, как, например, это сделано в следующем модифицированном варианте `app.yaml`:

```
handlers:
- url: /_ah/queue/default
  script: main.py
  login: admin
- url: .*
  script: main.py
```

Создание дополнительных задач и параметры конфигурации

Ранее мы продемонстрировали простейший способ создания задачи путем использования `taskqueue.add()`. Разумеется, существует гораздо больше вариантов, с помощью которых вы можете создать задачу, предназначенную для какой-либо другой очереди (не той, которая предусмотрена по умолчанию), указать желаемую задержку времени перед началом выполнения, способность передавать параметры этой задаче и т.п. В приведенном ниже списке перечислены лишь некоторые из этих вариантов, среди которых пользователь может выбрать одни или несколько подходящих ему.

1. `taskqueue.add(url='/task')`
2. `taskqueue.add(countdown=300)`
3. `taskqueue.add(url='/send_email', params={'groupID': 1})`
4. `taskqueue.add(url='/send_email?groupID=1', method='GET')`
5. `taskqueue.add(queue_name='send-newsletter')`

В первом случае передается конкретный указатель URL. Это относится к тем случаям, если вы предпочитаете использовать специализированный указатель URL (в отличие от того, который предусмотрен по умолчанию). Во втором случае указывается параметр типа счетчика; он задает задержку перед началом выполнения задачи (задача начинает выполняться по истечении указанного количества секунд). В третьем варианте показан пример использования специализированного указателя URL, а также параметров обработчика задачи. Четвертый пример представляет собой то же самое, что и третий, за исключением того, что пользователь запросил GET вместо предусмотренного по умолчанию POST. Последний из рассматриваемых здесь примеров относится к случаю, когда вы определили специализированную очередь задач вместо использования той, которая предусмотрена по умолчанию.

Это лишь некоторые из параметров, поддерживаемых `taskqueue.add()`. Об остальных параметрах можно прочитать на сайте <http://code.google.com/appengine/docs/python/taskqueue/functions.html>.

Во всех наших предыдущих примерах использовалась очередь, предусмотренная по умолчанию. Вы можете также создавать другие очереди; на момент написания этой книги вы могли создать до десяти дополнительных очередей для бесплатных приложений и сто очередей для платных приложений (однако, оговаривалось, что указанные условия могут измениться). Для этого вы должны сконфигурировать их в файле `queue.yaml` в формате, который имеет следующий вид:

```
queue:
- name: default
  rate: 1/s
  bucket_size: 10

- name: send-newsletter
  rate: 1/d
```

Очередь, предусмотренная по умолчанию, создается автоматически, но если вы хотите выбрать для нее другие параметры, укажите их в файле `queue.yaml`, как мы только что поступили, изменив предусмотренный по умолчанию параметр `rate`, равный 5/s, и параметр `bucket_size`, равный 5. (Параметр `rate` означает скорость обработки задач, а параметр `bucket_size` контролирует, как быстро очередь может обрабатывать последующие задачи.) Очередь отправки информационных бюллетеней предназначена для информационного бюллетеня, отправляемого абонентам по электронной почте один раз в сутки. Подробнее о всех конфигурационных параметрах для очередей можно прочитать на сайте <http://code.google.com/appengine/docs/python/config/queue.html>.

Наконец, обсуждая задачи, следует упомянуть еще один вид очереди, который предоставляет разработчикам большую гибкость с точки зрения того, как и когда создаются, а также используются и выполняются задачи. Типы очередей задач, обсуждавшиеся в настоящем разделе, относятся к категории очередей с проталкиванием; это означает, что ваше приложение генерирует задачи по требованию, проталкивая соответствующую работу в очереди по мере необходимости.

Мы упоминали также, что в App Engine предусмотрен альтернативный интерфейс задач, в соответствии с которым задания могут создаваться в *Pull Queues* (т.е. в очередях с вытаскиванием). App Engine может обращаться непосредственно к этим очередям (создавая или используя работу); доступ к этим очередям возможен также со стороны внешних приложений посредством интерфейса REST. Это означает,

что источником соответствующей работы может быть какое-либо приложение App Engine, а выполняться эта работа может, при желании, в каком-либо другом месте. Этим объясняется более гибкий график выполнения. Подробнее об очередях с выталикиванием сказано в документации на сайте <http://code.google.com/appengine/docs/python/taskqueue/overviewpull.html>.

Отправка электронной почты как задача

В одном из предыдущих примеров мы показали, как отправлять электронную почту из вашего приложения. Если вы отправляете лишь какое-то единичное сообщение (возможно, администратору вашего приложения, когда кто-то из пользователей делает вход в сообщение, размещенное в блоге), то было бы не так уж сложно отправить соответствующее сообщение электронной почты как одну из операций по обработке такого запроса. Однако если вам нужно отправлять электронную почту тысячам клиентов, то использование такого подхода было бы, наверное, не самым лучшим решением.

Работа по отправке всей этой электронной почты была бы, наверное, идеальным кандидатом для какой-нибудь задачи. Вместо того чтобы отправлять электронную почту, программа обработки создаст соответствующую задачу, передаст ей параметры (например, все адреса электронной почты или групповой ID группы пользователей, которым предстоит получить данное сообщение), а затем вернет ответ отправителю, в то время как задача будет отправлять электронную почту в подходящее для нее время (без привязки к пользователям).

Допустим, у нас есть веб-шаблон, который позволяет пользователю конфигурировать сообщение электронной почты и группу получателей. Когда пользователи выдают соответствующую форму на URL `/submit`, она обрабатывается классом `FormHandler`, для которой часть этого класса может иметь следующий вид:

```
class FormHandler(webapp.RequestHandler):
    def post(self): # should run at most 1/s
        groupID = self.request.get('group')
        taskqueue.add(params={'groupID': groupID})
        . . .
```

Метод `FormHandler.post()` обращается к методу `taskqueue.add()`, который добавляет задачу в очередь, предусмотренную по умолчанию, передавая идентификатор группы, которая будет принимать по электронной почте информационный бюллетень. Когда эта задача выполняется системой App Engine, она выдает команду POST на адрес `/_ah/queue/default`, для которой нам нужно определить еще один класс обработчика этой задачи.

Поскольку в данном случае мы используем очередь, предусмотренную по умолчанию, мы используем файл `app.yaml` в том виде, в каком он определен в предыдущем подразделе, с дополнительной защитной блокировкой в виде директивы `login: admin`. Теперь наша главная программа обработки (`main.py`) может указывать обработчики для соответствующей формы (в предыдущем примере), а также для обработчика задач, который нам еще предстоит создать:

```
def main():
    run_wsgi_app(webapp.WSGIApplication([
        . . .
        ('/submit', FormHandler),
```

```

        ('/_ah/queue/default', SendNewsletter),
        . . .
    )))

```

Нам нужно определить обработчик задач, `SendNewsletter`, который будет принимать входящий запрос наряду с идентификатором группы в том виде, в каком они отправляются из программы обработки форм. Затем мы переправим эти данные на некую обобщенную функцию, которая будет заниматься рассылкой сообщений электронной почты (информационных бюллетеней). Ниже показан один из способов, с помощью которого можно создать класс `SendNewsletter`.

```

class SendNewsletter(webapp.RequestHandler):
    def post(self): # should run at most 1/s
        groupID = self.request.get('group')
        send_group_email(groupID)
        . . .

```

Здесь, конечно, предполагается, что вы уже создали функцию `send_group_email()`, предназначенную для выполнения задачи приема идентификатора группы, получения адресов электронной почты всех членов (возможно, путем извлечения их из хранилища данных), формирования тела сообщения (из хранилища данных, путем автоматического генерирования, получения с какого-то другого сервера и т.п.) и, конечно же, выполнения фактического обращения к `mail.send_mail()`. Ниже показано, как мог бы выглядеть соответствующий код.

```

from datetime import date
from google.appengine.api import mail
. . .
def send_group_email(groupID):
    group_emails = . . . # получаем адреса членов группы groupID
    msg_body = . . . # получаем сообщение пользователя для членов группы groupID
    send_mail('noreply@APP_ID.appspotmail.com', group_emails,
        '%s Newsletter' % date.today().strftime("%B %Y"), msg_body)

```

Почему мы создали отдельную функцию `send_group_email()`? Нельзя ли было просто перенести эти строки кода в нашу программу обработки, чтобы избежать вызова дополнительной функции? Такая постановка вопроса представляется вполне естественной; но нам кажется, что повторное использование одного и того же кода является еще более достойной целью. Отдельная функция дает нам возможность использовать ту же функцию в каких-то других местах (возможно, как инструмент командной строки, специальную функцию/экран администратора и даже какое-то другое приложение). Если бы вы перенесли этот код в нашу программу обработки, то вам пришлось бы вырезать и вставить ее или в конечном счете разделить ее на две функции.

Ясно, что создавать задачи для выполнения работы приложения, не взаимодействующего непосредственно с пользователем, не так уж трудно. Задачи пользуются большой популярностью у пользователей системы App Engine. Предлагаем вам самим попытаться пользоваться ими. Прежде чем вы приступите к использованию задач, рекомендуем вам рассмотреть возможность использования “облегченного” пакета, если ваши потребности проще, чем потребности других пользователей: мы имеем в виду библиотеку `deferred`.

Пакет `deferred`

Как уже говорилось в предыдущем подразделе, очереди задач в системе App Engine — это великолепный способ делегирования дополнительной работы. Обычно такая работа не предполагает непосредственного взаимодействия с пользователем, и, как правило, разработчики предпочитают, чтобы такие действия не оказывали влияния на продолжительность времени ответа их пользователям. Однако, несмотря на то что задачи обеспечивают разработчику в системе App Engine значительную гибкость в выборе способа создания и выполнения задач, определенные дополнительные усилия понадобятся лишь для того, чтобы решить некоторые простые задачи. Вот тут-то нам и придет на помощь пакет `deferred`.

Пакет `deferred` — это удобный в использовании инструмент, который берет на себя значительную часть работы по созданию и выполнению задач: вам нужно скорректировать свой обработчик форм таким образом, чтобы вы могли создавать задачи; вы должны извлечь и предоставить надлежащие параметры задачи и правила выполнения, создать и сконфигурировать отдельные обработчики задач и т.п. Почему я не могу делегировать все эти действия какой-то задаче? Именно эту возможность и предлагает пользователям пакет `deferred`.

Вам придется иметь дело с единственной функцией, `deferred.defer()`, которую вы используете для создания отсроченной задачи. Это может быть простая задача наподобие вызова регистрации, как показано в приведенном ниже примере.

```
from google.appengine.ext import deferred
deferred.defer(logging.info, "Called a deferred task")
```

Для того чтобы воспользоваться библиотекой `deferred`, от вас потребуется лишь сконфигурировать свое приложение. Отсроченные задачи выполняются (по умолчанию) в очереди, предусмотренной по умолчанию, а, как вы уже знаете, вам не нужно ничего делать специально для этого, если только вы не хотите изменить характеристики очереди, предусмотренной по умолчанию. Вам также не нужно указывать в своем приложении программу обработки, которая будет работать с отсроченной задачей: соответствующие действия реализуются библиотекой `deferred`. Как видно из предыдущего короткого примера, вам нужно лишь передать `deferred.defer()` какую-либо вызываемую функцию Python, а также любые аргументы и/или аргументы-ключевые слова.

Кроме того, вы можете передать аргументы задачи (например, аргументы, перечисленные в последнем разделе), но вам нужно несколько замаскировать их, чтобы они не смешивались с аргументами, передаваемыми в вашу вызываемую функцию `deferred`. Для этого нужно, чтобы они начинались с одиночного символа подчеркивания, который не позволит спутать их с параметрами для вашей исполняемой программы. Например, чтобы сделать такой же вызов, как выше, но отсроченный по меньшей мере на 5 секунд, вы могли бы воспользоваться таким фрагментом кода:

```
deferred.defer(logging.info,
               "Вызываемая отсроченная задача", _countdown=5)
```

Мы можем без труда преобразовать пример рассылки сообщений электронной почты в этот эквивалентный код:

```
class SendNewsletter(webapp.RequestHandler):
    def post(self):
        groupID = self.request.get('group')
```

```
deferred.defer(send_group_email, groupID)
. . .
```

Отсроченные задачи могут вызывать функции, методы и, вообще говоря, любой объект, который является вызываемым или в котором определен `_call_`. Как следует из документации в коде, вызываемыми объектами, которые могут использоваться как отсроченные задачи, являются:

1. Функции, определенные на верхнем уровне модуля.
2. Классы, определенные на верхнем уровне модуля:
 - а) экземпляры тех классов, которые реализуют `_call_`;
 - б) методы экземпляров объектов тех классов;
 - в) методы классов тех классов.
3. Встроенные функции
4. Встроенные методы

Однако не допускается следующее (также задокументировано в коде):

- вложенные функции или замкнутые выражения;
- вложенные классы или объекты из них;
- лямбда-функции;
- статические методы.

Кроме того, все параметры используемого вызываемого объекта должны быть лишь вашими базовыми Python-объектами, такими, как константы, числа, строки, последовательности и типы хеширования. Полный перечень представлен в официальной документации Python на сайте <http://docs.python.org/release/2.5.4/lib/node317.html> (Python 2.5) или <http://docs.python.org/library/pickle.html#what-can-be-pickled-and-unpickled> (самая последняя версия Python).

Еще одним (и последним) ограничением в нашем примере является то, что функция `send_group_email()` должна находиться в каком-то другом модуле, а в нашу главную программу обработки нужно добавить импорт. Причиной этого является следующее: поскольку в момент, когда вы отсрочиваете свою задачу и она преобразовывается в последовательную форму, она фиксирует, что ваш код принадлежит модулю `_main_`, но когда пакет `deferred` исполняет вашу вызываемую функцию после приема ее из запроса POST, который создается этой задачей, исполняется модуль `deferred` (а значит, и соответствующий модуль `_main_`, который не виден в вашем коде). Вы получите сообщение об ошибке, которое выглядит примерно так, как показано ниже, если ваша функция `deferred` называлась `foo()`.

```
Traceback (most recent call last):
```

```
File "/usr/local/google_appengine/google/appengine/ext/deferred/deferred.py", line 258, in post
    run(self.request.body)
```

```
File "/usr/local/google_appengine/google/appengine/ext/deferred/deferred.py", line 122, in run
    raise PermanentTaskFailure(e)
```

```
PermanentTaskFailure: 'module' object has no attribute 'foo'
```

Однако, помещая ее за пределами `main.py` (или какого-либо другого Python-модуля, который содержит вашу главную программу обработки), вы сможете избежать этой путаницы и надлежащим образом импортировать и выполнить ваш код. Если вы хотите быстро освежить в памяти сведения о `_main_`, прочитайте главу о модулях в книге *Core Python*. Более подробные сведения о пакетедереве изложены в оригинальной статье, размещенной на сайте <http://code.google.com/appengine/articles/deferred.html>.

12.16. Профилирование с помощью Appstats

В системе App Engine очень важно уметь правильно оценить эффективность работы вашего приложения. Существенную помощь в решении этой задачи вам окажет приложение Appstats, которое является одним из инструментов в SDK. С помощью этого инструмента пользователи могут оптимизировать работу своих приложений. Приложение Appstats — это не только обычный профайлер кода. Оно также отслеживает всевозможные вызовы интерфейса API, выполняемые вашим приложением, измеряет продолжительность выполнения проходов туда и обратно при обращении к службам, предоставляемым серверной частью системы, посредством вызовов удаленных процедур (Remote Procedure Call — RPC), а также обеспечивает веб-интерфейс, который позволяет вам наблюдать за поведением приложения.

Сконфигурировать приложение Appstats на регистрацию событий довольно просто: нужно лишь создать в корневом каталоге вашего приложения файл `appengine_config.py` (или присоединить к нему, если этот файл уже существует), воспользовавшись следующей функцией:

```
def webapp_add_wsgi_middleware(app):
    from google.appengine.ext.appstats import recording
    app = recording.appstats_wsgi_middleware(app)
    return app
```

Предусмотрены также дополнительные возможности, которые вы можете реализовать здесь (об этих дополнительных возможностях прочитайте в соответствующей документации). После того как вы установите этот код, приложение Appstats начнет регистрировать события, являющиеся результатом функционирования вашего приложения. Средство регистрации событий не отвлекает на себя значительных ресурсов, поэтому вы вряд ли столкнетесь с ощутимым снижением производительности вашего приложения.

Последним этапом является создание административного интерфейса, посредством которого вы ознакомитесь с результатами измерений, выполненных Appstats. Сделать это можно одним из трех способов.

1. Добавление стандартной программы обработки в файл `app.yaml`.
2. Добавление специализированной страницы Admin Console.
3. Инициализация этого интерфейса как встроенной функции.

12.16.1. Добавление стандартной программы обработки в файл `app.yaml`

Для того чтобы добавить стандартную программу обработки в файл `app.yaml` (естественно, в разделе `handlers:`), воспользуйтесь следующим кодом:

```
- url: /stats.*
  script: $PYTHON_LIB/google/appengine/ext/appstats/ui.py
```

12.16.2. Добавление специализированной страницы Admin Console

Если вы хотите добавить пользовательский интерфейс Appstats как специализированную страницу Admin Console, сделайте это в разделе `admin_console` файла `app.yaml`, как показано ниже.

```
admin_console:
  pages:
    - name: Appstats UI
      url: /stats
```

12.16.3. Инициализация этого интерфейса как встроенной функции

Вы можете объявить пользовательский интерфейс Appstats как встроенную функцию, инициализировав его в разделе `builtins` файла `app.yaml`:

```
builtins:
  - appstats: on
```

Эта инициализация конфигурирует пользовательский интерфейс приложения Appstats таким образом, что по умолчанию будет использоваться путь `/_ah/stats`.

Со всеми возможностями, предоставляемыми приложением Appstats его пользователям, можно ознакомиться, воспользовавшись следующими ссылками:

```
http://code.google.com/appengine/docs/python/tools/appstats.html
http://googleappengine.blogspot.com/2010/03/easy-performance-profilingwith.html
http://www.youtube.com/watch?v=bvp7CuBWvGA
```

12.17. Служба URLfetch

Одно из ограничений, которые вам следует иметь в виду при использовании App Engine, заключается в том, что вы не можете создавать сетевые сокеты. Такое ограничение может сделать практически бесполезными большинство приложений; однако набор SDK действительно обеспечивает функциональность верхнего уровня как программа-посредник. Одним из главных применений способности создавать и использовать сокеты является взаимодействие с другими приложениями в Интернете. В связи с этим в системе App Engine предусмотрена служба URLfetch, с помощью которой ваше приложение может выполнять в режиме онлайн HTTP-запросы (GET, POST, HEAD, PUT, DELETE) к другим серверам. Ниже приведен пример того, как пользоваться этой службой.

```
from google.appengine.api import urlfetch
...
res = urlfetch.fetch('http://google.com')
if res.status_code == 200:
    self.response.out.write(
```

```
'First 100 bytes of google.com:<p>%s</p>' %
res.content[:100])
. . .
```

В дополнение к модулю `urlfetch` в системе App Engine вы можете также пользоваться модулями стандартной библиотеки `urllib`, `urllib2` и `httplib`, модифицированным таким образом, чтобы можно было взаимодействовать посредством службы `URLfetch` (которая, естественно, выполняется на расширяемой инфраструктуре Google).

Однако вы должны учитывать ряд ограничений, например, взаимодействие с серверами через HTTPS, а также заголовки запросов, которые невозможно модифицировать или задавать. Подробно эти ограничения описаны в документации на сайте <http://code.google.com/appengine/docs/python/urlfetch/overview.html>. Там же приведен обзор того, как пользоваться службой `URLfetch`.

Наконец, поскольку некоторые полезные нагрузки характеризуются значительным временем ожидания, предусмотрена также асинхронная служба `URLfetch`. Вам также предоставляется возможность проведения опроса, чтобы узнать, выполнен ли запрос, или сделать обратный вызов. Подробно асинхронная служба `URLfetch` описана в документации на сайте <http://code.google.com/appengine/docs/python/urlfetch/asynchronousrequests.html>.

12.18. Быстрый обзор (без использования кода Python)

Это еще один раздел “быстрого обзора”, в котором мы познакомим вас с конфигурируемыми возможностями. Этот раздел не будет иллюстрироваться исходным кодом.

12.18.1. Служба Cron (планируемые задачи/задания)

Крон-задание (`cronjob`) — это задача, которая выполняется в запланированные моменты времени и иницируется на компьютерах POSIX. Система App Engine предоставляет своим пользователям службу крон-типа. В этом случае код Python фактически не используется; исключением является лишь программа обработки, которая запускается на выполнение в соответствующий момент времени.

Для того чтобы воспользоваться службой `Cron`, нужно создать файл `cron.yaml`, у которого может быть, например, такое содержимое:

```
cron:
- url: /task/roll_logs
  schedule: every day
- url: /task/weekly_report
  schedule: every friday 17:00
```

По мере необходимости можете также указать поля `description:` и `timezone:`. Формат временного графика (`schedule`) достаточно гибок. Подробнее о крон-заданиях можно прочитать в документации на сайте <http://code.google.com/appengine/docs/python/config/cron.html>.

12.18.2. Запросы разогрева

Цель запросов разогрева в том, чтобы сократить время ожидания, с которым сталкиваются пользователи вашего приложения, когда требуется “раскрутить” новые экземпляры для обслуживания еще большего количества пользователей. Допустим, вы хорошо справляетесь с обслуживанием своего приложения в единственном экземпляре. Однако, если к нему внезапно обратится большое число пользователей, его трафик может резко возрасти. Когда выполняющийся в данный момент экземпляр перестанет справляться с резко возросшей нагрузкой, для обслуживания всех запросов придется вводить в действие новые экземпляры.

Если бы не было возможности разогрева, то первому пользователю, обратившемуся к новому экземпляру вашего приложения, пришлось бы ждать ответа дольше, чем если бы он обратился к уже выполняющемуся экземпляру. Эта дополнительная задержка вызвана необходимостью подождать, пока будет загружен новый экземпляр (лишь после этого он сможет обслужить запрос первого пользователя). Если бы у вас была возможность просто разогреть новый экземпляр, загрузив ваше приложение еще до того, как у него появится какой-то трафик, то пользователям не пришлось бы испытывать неудобства от такой задержки. Именно для этого и нужны запросы разогрева.

Аналогично другим возможностям системы App Engine, запросы разогрева не иницируются по умолчанию. Для того чтобы их инициировать, добавьте одну строку в разделе `inbound_services` файла `app.yaml`:

```
inbound_services:  
- warmup
```

Кроме того, когда новый экземпляр начнет функционировать в режиме онлайн, система App Engine выдаст запрос GET на `/_ah/warmup`. Если вы создаете для этого специальную программу обработки, то можете также предварительно загрузить любые данные в своем приложении. Необходимо лишь помнить, что если ваше приложение еще не принимает никакого трафика и у него нет выполняющихся экземпляров, то самый первый запрос все же иницирует запрос загрузки для этого пользователя (даже если разогрев уже иницирован). Этому пользователю просто не повезло!

Причина этого вполне очевидна: запрос разогрева не приносит никакой пользы и, по сути, лишь увеличивает время ожидания, поскольку он уже должен быть выдан. Вряд ли вы будете в восторге от того, что вам придется нести потери в результате выдачи запроса разогрева в дополнение к запросу загрузки, прежде чем ваше приложение сможет ответить этому первому пользователю! Запросы разогрева в действительности полезны лишь в случае, если уже есть серверы, обрабатывающие трафик к вашему приложению, что дает возможность App Engine разогреть новые экземпляры.

Эта возможность относится к числу конфигурируемых возможностей и тоже не требует какого-либо кода Python. Подробнее о запросах разогрева читайте на следующих сайтах:

```
http://code.google.com/appengine/docs/adminconsole/instances.html#Warming\_Requests  
http://code.google.com/appengine/docs/python/config/appconfig.html#Inbound\_Services
```


12.18.3. Защита от хакерских атак типа “отказ в обслуживании”

Система App Engine предоставляет своим пользователям упрощенную форму защиты от систематических хакерских атак типа “отказ в обслуживании” (Denial-of-Service — DoS), направленных против вашего приложения. Для этого вы должны создать файл `dos.yaml` с разделом `blacklist:`, как в приведенном ниже примере.

```
blacklist:
- subnet: 89.212.115.11
description: block DoS offender
- subnet: 72.14.194.1/15
description: block offending subnet
```

Вы можете включать в этот “черный список” отдельные IP-адреса или подсети как для IPv4, так и для IPv6. После того как вы выгрузите файл `dos.yaml`, запросы, поступающие с указанных адресов и подсетей, не смогут обратиться к коду вашего приложения. Вас не будут тарифицировать за любые ресурсы, которые используются для блокирования компьютеров, отправляющих трафик с адресов и подсетей, включенных в ваш “черный список”.

Официальная документация по DoS-проекту представлена на сайте <http://code.google.com/appengine/docs/python/config/dos.html>.

12.19. Обеспечение замкнутости поставщика

Последний вопрос, который нам предстоит обсудить перед тем, как вы отправитесь в полет на облака, посвящен обеспечению *замкнутости поставщика* (vendor lock-in). Когда мы говорим о замкнутости поставщика, то речь, вообще говоря, идет о системах, которые в силу своей природы *чрезвычайно затрудняют или вообще делают невозможным попадание* данных и/или логики в другие подобные или конкурирующие системы. На протяжении своей, еще относительно короткой жизни система App Engine уже успела приобрести репутацию системы, которая заставляет пользователей использовать интерфейс API Google, в то же время затрудняя миграцию приложений с этой платформы.

В то время как компания Google настоятельно рекомендует применять их интерфейсы API с целью извлечения максимальной пользы от этой системы, пользователи должны понимать, что за получение тех или иных преимуществ приходится чем-то расплачиваться. Представляется вполне справедливым, что в обмен на возможность использования преимуществ расширяемой инфраструктуры компании Google (менеджмент которой является исключительно менеджментом данной компании) вы должны использовать их интерфейсы API для написания своего кода. Опять-таки нельзя получить что-либо, не пожертвовав чем-либо другим, не так ли? Между тем практическая реализация такой расширяемости является одной из самых трудных и дорогостоящих работ. Однако компания Google пытается, по мере возможности, бороться с замкнутостью, предоставляя при этом своим пользователям возможность сполна использовать преимущества системы App Engine.

Например, несмотря на то, что в системе App Engine предусмотрена платформа `webapp` (или `webapp2`), ничто не мешает вам пользоваться другими платформами, имеющими открытый код и совместимыми с системой App Engine. К их числу относятся, в частности, платформы Django, web2py, Tipfy, Flask и Bottle. Что же касается интерфейса API Datastore, то вы можете полностью обойти эту службу, если

используете нереляционную систему Django и библиотеку `djangoappengine`. Эти библиотеки позволяют исполнять чистые Django-приложения непосредственно поверх системы App Engine; таким образом, ничто не мешает вам перемещать свои приложения между системой App Engine и любым традиционным хостингом, который поддерживает платформу Django. К тому же эта возможность не ограничивается языком Python; команда разработчиков системы App Engine постаралась сделать свои интерфейсы API настолько совместимыми со стандартами Java Specification Request (JSR), насколько это возможно. Если вы знаете, как написать Java-сервлет, то ваши познания легко перенести на систему App Engine.

Наконец, существуют две серверные системы с открытым исходным кодом, которые, по заявлениям их разработчиков, совместимы с клиентом App Engine: AppScale и TyphoonAE. Последняя используется как более традиционный проект с открытым исходным кодом, тогда как первая активно разрабатывается в Калифорнийском университете (Санта-Барбара). Более подробные сведения об обоих этих проектах можно найти на их соответствующих домашних страницах: <http://appscale.cs.ucsb.edu> и <http://code.google.com/p/typhoonae>. Если вам необходим полный контроль своего приложения и вы не хотите выполнять его в каком-либо из центров обработки и хранения данных Google, создайте свою собственную платформу с любой из этих систем.

12.20. Ресурсы

Платформе App Engine можно было бы посвятить целую книгу (некоторые авторы именно так и поступили); к сожалению, в этой главе нам не остается ничего другого, как представить ее описание лишь в самых общих чертах. Если же вы хотите получить более подробные сведения о системе App Engine, воспользуйтесь перечисленными ниже источниками.

- **Blobstore.** Позволяет пользователям обслуживать объекты данных (блобы), которые оказались чересчур велики для Datastore (например, медиа-файлы)
<http://code.google.com/appengine/docs/python/blobstore/overview.html>
- **Capabilities.**
<http://www.slideshare.net/jasonacooper/strategies-for-maintaining-app-engine-availability-during-read-only-periods>
<http://code.google.com/appengine/docs/python/howto/maintenance.html>
- **Channel.** Служба, которая позволяет вашему приложению проталкивать данные непосредственно к браузеру; также известна под названиями Reverse Ajax, browser push, Comet
<http://googleappengine.blogspot.com/2010/12/happy-holidays-from-app-engine-team-140.html>
<http://blog.myblive.com/2010/12/multiuser-chatroom-with-app-engine.html>
<http://code.google.com/p/channel-tac-toe/>
<http://arstechnica.com/web/news/2010/12/app-engine-gets-streaming-api-and-longer-background-tasks.ars>

- *High Replication Datastore*

<http://googleappengine.blogspot.com/2011/01/announcing-high-replication-datastore.html>

<http://code.google.com/appengine/docs/python/datastore/hr/overview.html>

- *Mapper*. Первый сегмент службы MapReduce позволяет пользователям выполнять итерации по отношению к постоянным данным пользователя

<http://googleappengine.blogspot.com/2010/07/introducing-mapper-api.html>

<http://code.google.com/p/appengine-mapreduce/>

- *Matcher*. Значительно расширяемая инфраструктура согласования в реальном масштабе времени: регистрирует запросы для поиска совпадений с потоком объектов

<http://www.onebigfluke.com/2010/10/magical-api-from-future-app-engines.html>

http://groups.google.com/group/google-appengine/browse_thread/thread/5462e14c31f44bef

<http://code.google.com/p/google-app-engine-samples/wiki/AppEngineMatcherService>

- *Namespaces*. Позволяет создавать приложения, рассчитанные на многих участников, распределяя данные Google App Engine по разным “отсекам”

http://googleappengine.blogspot.com/2010/08/multi-tenancy-support-high-performance_17.html

<http://code.google.com/appengine/docs/python/multitenancy/overview.html>

<http://code.google.com/appengine/docs/python/multitenancy/multitenancy.html>

- *OAuth*. Обеспечивает посторонним лицам безопасный способ доступа к приложениям и данным от имени определенного пользователя, не требуя при этом авторизации (логинов/паролей и т.п.)

<http://code.google.com/appengine/docs/python/oauth/overview.html>

<http://oauth.net>

- *Pipeline*. Управляет одновременно многими задачами с длительным временем выполнения/потоками заданий и представляет в требуемом порядке их результаты. (См. также *Fantasm*, еще один простой менеджер технологического потока, разработанный одной из сторонних фирм.)

<http://code.google.com/p/appengine-pipeline/wiki/GettingStarted>

<http://code.google.com/p/appengine-pipeline/>

<http://news.ycombinator.com/item?id=2013133>

<http://googleappengine.blogspot.com/2011/03/implementing-workflows-on-app-engine.html>

В табл. 12.3 приведены веб-адреса многих платформ разработки, представленных в этой главе.

Таблица 12.3. Платформы для разработки с помощью Google App Engine

Проект	URL
Google App Engine	http://code.google.com/appengine
Bigtable	http://labs.google.com/papers/bigtable.html
Megastore	http://research.google.com/pubs/pub36971.html
webapp	http://code.google.com/appengine/docs/python/gettingstarted/usingwebapp.html
webapp2	http://code.google.com/appengine/docs/python/tools/webapp
	http://code.google.com/appengine/docs/python/gettingstartedpython27/usingwebapp.html
	http://code.google.com/appengine/docs/python/tools/webapp
	http://webapp-improved.appspot.com/
Django	http://djangoproject.com
Django-nonrel	http://www.allbuttonspressed.com/projects/django-nonrel
djangoappengine	http://www.allbuttonspressed.com/projects/djangoappengine
Bottle	http://bottlepy.org
Flask	http://flask.pocoo.org/
Tipfy	http://tipfy.org
web2py	http://web2py.com
AppScale	http://appscale.cs.ucsb.edu
TyphoonAE	http://code.google.com/p/typhoonae

12.21. Резюме

Как следует из обширного материала, представленного как в этой главе, так и в главе 11, на сегодняшний день Django и Google App Engine являются двумя самыми мощными и гибкими веб-платформами в сообществе Python. Добавьте к этому все остальные веб-платформы (TurboGears, Pyramid, web2py, web.py и т.д.), которые сами по себе являются весьма внушительными, и вы получите впечатляющую экосистему платформ и довольно широкий выбор для тех, кто разрабатывает веб-приложения на языке Python. Еще важнее то обстоятельство, что у каждой из веб-платформ на языке Python есть немалое число преданных сторонников и разработчиков, отдающих предпочтение соответствующей веб-платформе.

Программисты из числа тех, кого принято называть мастерами на все руки, могут даже время от времени переходить с одной платформы на другую, в зависимости от того, какую из них они считают наиболее подходящей для выполнения конкретной работы, которую предстоит выполнить разработчику. Очень хорошо, что сообщество программистов сплотивилось вокруг некоторых из этих более крупных и широко известных платформ, поскольку, несмотря на то, что цитата, приведенная в начале этой главы, звучит несколько иронично, в ней все же есть зерно истины, и наш мир был бы гораздо хуже, если бы каждому из разработчиков пришлось создавать свою собственную веб-платформу.

И наконец, последнее замечание: ни один из примеров, приведенных в этой главе, не относится к Python 3, поскольку ни одна из платформ еще не поддерживает его. Когда же эта ситуация изменится в лучшую сторону, мы обязательно сделаем соответствующие исходные коды доступными для вас онлайн, а также в будущих изданиях этой книги.

12.22. Упражнения

Система Google App Engine

- 12.1. *Основные сведения.* Какое отношение имеет Python к Google App Engine?
- 12.2. *Основные сведения.* В чем отличие Google App Engine от других сред разработки?
- 12.3. *Конфигурация.* В чем заключаются некоторые различия между конфигурационными файлами Django и App Engine?
- 12.4. *Конфигурация.* Назовите места, где приложения Django выполняют отображение типа “URL в обработчик”. Сделайте то же самое для приложений App Engine.
- 12.5. *Конфигурация.* Как добиться, чтобы приложения Django выполнялись (в основном) в немодифицированном виде на Google App Engine?
- 12.6. *Конфигурация.* Чтобы выполнить это упражнение, перейдите на сайт <http://code.google.com/appengine/>, а затем загрузите и установите последнюю версию SDK Google App Engine для своей платформы.
 - a) Воспользуйтесь приложением Launcher, если работаете на PC под Windows или на Mac, и создайте приложение под названием “helloworld”. Если же вы работаете на других платформах, создайте следующую пару файлов с указанным здесь содержанием:

```
i. Первым файлом является: app.yaml
приложение: helloworld
version: 1
runtime: python
api_version: 1
```

```
обработчики:
- url: .*
  script: main.py
ii. Вторым файлом является: main.py
```

```
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
class MainHandler(webapp.RequestHandler):
    def get(self):
        self.response.out.write('Hello world!')

application = webapp.WSGIApplication([
    ('/', MainHandler),
], debug=True)

def main():
    run_wsgi_app(application)
```

```
if __name__ == '__main__':  
    main()
```

б) Начните свое приложение с использования *Launcher* или выполнения `'dev_appserver.py DIR'`, где *DIR* — каталог, в котором размещаются файлы `app.yaml` и `main.py`, а затем обратитесь на `http://localhost:8080` (или подходящий номер порта), чтобы убедиться в том, что ваш код работает, а “Hello World!” действительно появляется в вашем браузере. Замените этот отображаемый результат на что-либо, отличное от строки “Hello World!”

- 12.7. *Учебник*. Попробуйте дополнить учебник *Getting Started*, который можно найти на сайте `http://code.google.com/appengine/docs/python/gettingstarted`. Предупреждение: не пытайтесь просто копировать код, который вы найдете в нашей книге. Надеюсь, вы модифицируете это приложение таким образом, чтобы оно делало что-то, несколько отличающееся от того, что предлагается на этом сайте, и/или добавьте в него какие-то новые функции.
- 12.8. *Связь*. Электронная почта является критически важной функцией приложения. В одном из более ранних упражнений вы добавили рассылку электронной почты, когда был создан новый вход в блог. Прodelайте то же самое со своим блог-приложением *App Engine*.
- 12.9. *Изображения*. Предоставьте возможность пользователям выложить по одной фотографии или картинке на блог и создайте привлекательную галерею блог-постов.
- 12.10. *Курсоры и разбиение на страницы*. Подобно блог-приложению *Django*, отображение десяти самых последних постов (сообщений в блог) является неплохой идеей, но предоставление пользователям возможности просматривать постранично более ранние посты — еще более интересная идея. Используйте курсоры и добавьте в свое приложение разбиение на страницы.
- 12.11. *Связь*. Предоставьте возможность пользователям “общаться” с вашим приложением с помощью IM (т.е. путем мгновенного обмена сообщениями). Создайте меню команд для передачи блог-входов, извлечения самых последних входов и реализации любых других возможностей, которые могли бы, на ваш взгляд, вызвать повышенный интерес пользователей.

Разработка с помощью Django или App Engine

- 12.12. *Система управления данными пользовательского облака*. Создайте систему мониторинга погоды. Предоставьте возможность работать с вашей системой многим пользователям, применив для этого любую подходящую вам систему идентификации пользователей. Каждый пользователь должен иметь определенную совокупность кодов, идентифицирующих его географическое местоположение (почтовый, или ZIP-код, код аэропорта [город, штат], [город, страна] и т.п.). Каждый пользователь должен быть представлен целой “сеткой” актуальных для него мест, наряду с текущим прогнозом и расширенным 3–5-дневным прогнозом. Можете воспользоваться одним из многих погодных API, работающих онлайн.

- 12.13. Система управления финансами.** Создайте систему управления портфелем акций/ценных бумаг. Этот портфель включает обычные акции (на любой фондовой бирже), взаимные фонды, фонды, торгуемые на бирже (exchange-traded funds — ETF), американские депозитарные расписки (American depository receipt — ADR), индексы фондовой биржи или какие-либо другие ценные бумаги, которым присвоен соответствующий биржевой символ (так называемый “тикер”, с помощью которого можно быстро отыскать интересующие вас ценные бумаги). Если вы проживаете не в Соединенных Штатах, выберите тот вариант механизма торговли ценными бумагами, который используется в вашей стране.
- 12.14. Применение спортивной статистики.** Допустим, вы являетесь заядлым любителем боулинга и участником соревнований в этом виде спорта (в том числе и на международном уровне). Разумеется, было бы не так уж сложно разработать приложение, которое фиксировало бы ваши спортивные достижения, подсчитывало ваши средние показатели и т.п., но от вас, как разработчика такого приложения, требуется нечто большее. В частности, вы должны показывать результаты анализа тенденций и изменение средних результатов на протяжении суток, а также предоставлять возможность пользователям вводить наряду с результатами количество открытых фреймов. Это позволит им проверять, насколько удачно они сыграли и удаётся ли им в течение всего вечера выбивать *Brooklyns*. Кроме того, вам нужно предусмотреть флажок, с помощью которого можно было бы указывать, санкционирована ли та или иная игра, а также разрешить привязку ссылок на видеоклипы к соответствующим играм. Живите и дышите своим любимым видом спорта, даже когда находитесь вдали от боулинг-клуба. Создайте сетевой сервер, который позволит вам обращаться к этим данным по Интернету, когда вы находитесь за городом, или со своего мобильного телефона.
- 12.15. Система управления учебным курсом.** Разработайте систему управления учебным курсом. Она должна обеспечивать пользователям возможность войти в систему, задав имя пользователя и пароль, и получить в свое распоряжение “кабинет” для общения в режиме онлайн, форумы для общения в режиме офлайн (так называемая “внеполосная связь” — Out-of-Band [OOB] communication) и место, где можно передать преподавателю выполненное домашнее задание и получить оценку. Что касается преподавателей, то им нужно предоставить возможность добавлять новые задания и усложнять уже существующие, участвовать вместе со студентами в дискуссиях и форумах, выкладывать объявления, касающиеся учебного процесса, статические файлы и отправлять сообщения студентам. Для практического воплощения своего решения выберите либо платформу Django, либо Google App Engine, либо, еще лучше, воспользуйтесь Django-non-rel, чтобы создать приложение Django, которое может выполняться в традиционной среде с хост-системой или с помощью Google на системе App Engine.
- 12.16. Менеджер кулинарных рецептов.** Разработайте приложение для управления виртуальным собранием кулинарных рецептов. Это несколько отличается от управления, например, коллекцией музыкальных произведений, все звуковые файлы которой (в формате MP3 или каких-либо других форматах) находятся на жестком диске вашего компьютера. Дело в том, что эти

кулинарные рецепты существуют лишь онлайн. Когда пользователи вводят URL кулинарных рецептов, ваше приложение должно предоставлять им возможность распределяться по разным категориям (но фактический URL должен сохраняться лишь один раз). Кроме того, пользователя нужно предупреждать, когда та или иная ссылка перестает “работать”; это можно делать с помощью электронной почты, IM/XMPP и даже с помощью SMS, если вам удастся найти подходящий шлюз “электронная почта–SMS” (см. http://en.wikipedia.org/wiki/List_of_SMS_gateways), если у вас нет своей собственной службы SMS. Создайте мини-поисковый агент, чтобы при распечатке кулинарных рецептов отображалась также пиктограмма изображения, размещенного на той же странице, что и URL соответствующего рецепта (если таковое имеется). Вы должны также предоставить своим пользователям возможность выполнять просмотр по категориям/кухням.

ГЛАВА

13

Веб-службы

В этой главе...

- Введение
- Сервер биржевых котировок Yahoo! Finance
- Создание микроблогов в сети Twitter

Я вполне могу обойтись без сети Twitter. Я пользуюсь им, лишь когда у меня появляется свободное время: за обедом, за завтраком, в перерывах между работой, в нерабочее время, сейчас, тогда — одним словом, всегда.
Неизвестный автор (не позже мая 2010 г.)

В этой главе мы кратко ознакомим вас с тем, как пользоваться парой имеющихся на сегодняшний день веб-служб: “старой” службой, сервером биржевых котировок Yahoo, и новой службой Twitter.

13.1. Введение

В Интернете есть немало веб-служб и приложений, оказывающих широкий спектр услуг. В Интернете можно найти интерфейсы прикладного программирования (Application Programming Interface — API) большинства крупных компаний, таких как Yahoo!, Google, Twitter и Amazon (не говоря уж о множестве других). В прошлом интерфейсы API применялись лишь для доступа к данным путем использования этих служб; однако нынешние интерфейсы API — совсем другое дело. Они обладают широким спектром возможностей, причем пользователь может фактически интегрировать службы в свои собственные веб-сайты и веб-страницы, получившие название *гибридных приложений* (mash-up).

Эта область вызывает повышенный интерес специалистов, и в дальнейшем мы продолжим ее обсуждение (REST, XML, JSON, RSS, Atom и т.п.), но сейчас мы хотим вернуться в прошлое и поговорить о более раннем интерфейсе, который, тем не менее, может быть полезен и в наше время (нужно отметить, что он оказался на удивление долговечным). Мы имеем в виду сервер биржевых котировок от Yahoo! на сайте <http://finance.yahoo.com>.

13.2. Сервер биржевых котировок Yahoo! Finance

Если вы посетите веб-сайт Yahoo! Finance и отобразите котировку каких-либо акций, то в разделе Toolbox (в нижней части страницы) под базовыми котировочными данными найдете указатель URL, помеченный как Download Data. Воспользовавшись этой ссылкой, вы можете загрузить файл с расширением .csv, пригодный для импортирования в приложение Microsoft Excel или Intuit Quicken. Соответствующий указатель URL должен выглядеть примерно так (если вы находитесь на странице GOOG):

<http://quote.yahoo.com/d/quotes.csv?s=GOOG&f=s1ld1t1clhgv&e=.csv>

Если параметры MIME для вашего браузера установлены правильно, то он фактически запустит в вашей системе программу, сконфигурированную для обработки CSV-данных (обычно это такие приложения электронных таблиц, как Excel или LibreOffice Calc). Это обусловлено главным образом последней парой переменных (ключевых значений) в указанной ссылке, e=.csv. Эта переменная фактически не используется сервером, поскольку он в любом случае возвращает данные в формате CSV.

Если мы воспользуемся функцией `urllib2.urlopen()`, то увидим, что для любого биржевого символа (тикера) возвращается одна CSV-строка:

```
>>> from urllib2 import urlopen
>>> url = 'http://quote.yahoo.com/d/quotes.csv?s=goog&f=sllldclp2'
>>> u = urlopen(url, 'r')
>>> for row in u:
...     print row
...
"GOOG",600.14,"10/28/2011",+1.47,"+0.25%"
>>> u.close()
```

2.3

Затем нужно вручную выполнить синтаксический анализ этой строки (убрав закрывающий пробел и разделив ее на составные части по запятой). В качестве альтернативы самостоятельному синтаксическому анализу этой строки данных мы можем воспользоваться модулем `csv`, впервые появившимся в Python 2.3, который выполняет как разделение строки на составные части, так и удаление закрывающего пробела. При использовании `csv` мы можем заменить в предыдущем примере цикл `for` на следующий фрагмент кода (предполагается, что все остальные строки остаются неизменными):

```
>>> import csv
>>> for row in csv.reader(u):
...     print row
...
['GOOG', '600.14', '10/28/2011', '+1.47', '+0.25%']
```

Проанализировав поле аргумента `f`, передаваемое серверу посредством строки URL, и ознакомившись с онлайн-подсказкой Yahoo! для этой службы, мы увидим, что символы (`sllldclp2`) соответствуют символу тикера, последней цене, дате, изменению и изменению, выраженному в процентах.

Более подробную информацию можно получить, обратившись на страницы подсказки (Help) веб-сайта Yahoo! Finance; просто выполните поиск по ключевым словам “download data” и “download spreadsheet format”. Дальнейший анализ этого интерфейса API позволяет выявить еще несколько параметров, таких как цена при предыдущем закрытии торгов, процентное изменение текущей цены к цене при предыдущем закрытии торгов, самая высокая и самая низкая цена за 52-недельный период и т.д. Эти параметры, а также форматы возвращаемых компонентов подытожены в табл. 13.1. (Не удивляйтесь цене акций Yahoo! за прошлое десятилетие: именно так обстояли дела в то время.)

Таблица 13.1. Параметры сервера биржевых котировок Yahoo! Finance

Данные биржевых котировок	Имя поля ^a	Возвращаемый формат ^b
Биржевой символ (тикер)	S	"YHOO"
Цена последних торгов	l1	328
Дата последних торгов	d1	"2/2/2000"
Время последних торгов	t1	"4:00pm"
Изменение с момента предыдущего закрытия торгов	c1	+10.625
Изменение с момента предыдущего закрытия торгов, выраженное в процентах	p2	"+3.35%"
Цена при предыдущем закрытии торгов	p	317.375
Цена при последнем открытии торгов	o	321.484375

Данные биржевых котировок	Имя поля ^а	Возвращаемый формат ^б
Самая высокая цена за сутки	h	337
Самая низкая цена за сутки	g	317
Диапазон цен за 52-недельный период	w	"110 – 500.125"
Объем торгов за эти сутки	v	6703300
Рыночная капитализация	jl	86.343B
Доход за одну акцию	e	0.20
Соотношение "цена–доход"	r	1586.88
Название компании	n	"YHOO INC"

^а Первый символ названия поля является алфавитным символом; второй, если таковой имеется, — цифровым.

^б Некоторые значения возвращаются (дополнительно) котируемыми, хотя все они возвращаются от сервера как часть одной CSV-строки.

Сервер представляет имена полей в том порядке, в каком они указаны вами. Просто конкатенируйте их как один аргумент с параметром поля *f* как часть запрашивающего URL. Как указывалось в сноске *b* к табл. 13.1, некоторые из возвращаемых компонентов котируются отдельно. Задача правильного извлечения соответствующих данных возлагается на программу синтаксического анализа. Обращайте внимание на результирующие (под)строки при выполнении синтаксического анализа вручную (в отличие от использования модуля *csv* в нашем предыдущем примере). Если какое-то из значений отсутствует, то сервер котировок возвращает "N/A", как показано в приведенном ниже коде.

Если, например, мы даем серверу запрос поля в виде *f=sl1d1clp2*, то в ответ получим представленную ниже строку, если будет указан правильный тикер акций (в 2000 г. я действительно выдал такой запрос):

```
"YHOO", 166.203125, "2/23/2000", +12.390625, "+8.06%"
```

В случаях, когда интересующие нас акции уже не торгуются на фондовой бирже, мы получим нечто наподобие того, что показано ниже (еще раз обратите внимание на то, как поля, которые возвращались с соответствующими котируемками, по-прежнему возвращаются, даже если для них указано "N/A"):

```
"PBLS.OB", 0.00, "N/A", N/A, "N/A"
```

Вы можете также указать несколько символов тикера акций, например *s=YHOO,GOOG,EBAY,AMZN*. В ответ вы получите одну строку данных (например, такую, как показано выше) для каждой компании. Необходимо лишь помнить, что "[любая] самовольная рассылка котируемых данных, отображаемых на Yahoo!, строго воспрещается", как указывается на страницах подсказки (Help) веб-сайта Yahoo! Finance, поэтому вы можете использовать эти данные лишь для своих личных целей. Необходимо также иметь в виду, что все загруженные вами котируемые данные несколько устарели.

Воспользовавшись имеющимися у нас на этот момент знаниями, попробуем разработать пример приложения, которое считывает и отображает некие данные биржевых котировок для ряда избранных интернет-компаний, как показано в примере 13.1.

Пример 13.1. Биржевые котировки Yahoo! Finance (stock.py)

Этот сценарий загружает и отображает цены акций с сервера котировок Yahoo!.

```

1  #!/usr/bin/env python
2
3  from time import ctime
4  from urllib2 import urlopen
5
6  TICKS = ('yhoo', 'dell', 'cost', 'adbe', 'intc')
7  URL = 'http://quote.yahoo.com/d/quotes.csv?s=%s&f=s1lclp2'
8
9  print '\nPrices quoted as of:%s PDT\n' % ctime()
10 print 'TICKER', 'PRICE', 'CHANGE', '%AGE'
11 print '-----', '-----', '-----', '-----'
12 u = urlopen(URL % ','.join(TICKS))
13
14 for row in u:
15     tick, price, chg, per = row.split(',')
16     print tick, '%.2f' % float(price), chg, per,
17
18 u.close()

```

Выполняя этот сценарий, получаем следующий результат:

```
$ stock.py
```

```
Prices quoted as of: Sat Oct 29 02:06:24 2011 PDT
```

```

TICKER PRICE CHANGE %AGE
-----
"YHOO" 16.56 -0.07 "-0.42%"
"DELL" 16.31 -0.01 "-0.06%"
"COST" 84.93 -0.29 "-0.34%"
"ADBE" 29.02 +0.68 "+2.40%"
"INTC" 24.98 -0.15 "-0.60%"

```

Построчное объяснение

Строки 1–7

Этот сценарий на языке Python 2 использует `time.ctime()` для отображения текущего момента времени, когда информация о биржевых котировках была загружена с Yahoo!, а `urllib2.urlopen()` используется для подключения к службе Yahoo!, позволяющей получить данные биржевых котировок. За операторами импорта следуют символы тикера акций, а также фиксированный URL, который извлекает все соответствующие данные.

Строки 9–12

Этот короткий блок кода отображает временную метку загрузки информации о биржевых котировках, а также использует `urllib2.urlopen()` для запроса соответствующих данных. (Если вы читали предыдущие издания этой книги, то наверняка обратите внимание на то, что благодаря подсказкам со стороны особо наблюдательных читателей мы несколько упростили код вывода!)

Строки 14–18

Как только в нашем распоряжении появляется открытый файлоподобный объект к данным, загруженным из веб, мы просматриваем каждую возвращаемую строку, разбиваем на части список, отделенный запятой, а затем отображаем их на экране.

Подобно считыванию строк из какого-либо текстового файла, символ завершения последней строки также сохраняется, поэтому в конце оператора `print` нам нужно добавить завершающую запятую, чтобы подавить его символ `NEWLINE`; в противном случае весь вывод будет выполняться с двойным интервалом.

Наконец, обратите внимание на то, что некоторые из возвращаемых полей заключены в кавычки. В конце этой главы приведено несколько упражнений, при выполнении которых у вас будет возможность добиться, чтобы ваша выходная информация отображалась лучше, чем предусмотрено по умолчанию.

13.3. Создание микроблогов в сети Twitter

В этом разделе мы исследуем мир микроблогинга с помощью службы Twitter. Это исследование мы начнем с краткого введения в социальные сети, опишем место, которое занимает Twitter в системе социальных сетей, познакомим с разными интерфейсами, реализованными в Python, и, наконец, продемонстрируем простой и в то же время весьма поучительный пример.

13.3.1. Социальные сети

Последние годы стали временем бурного развития социальных сетей. Все началось с очень простой концепции *сетевого журнала* (Web logging, или, коротко, blogging). Этот тип службы предоставляет в распоряжение пользователей учетные записи, с помощью которых они могут размещать в сети те или иные информационные материалы. Это можно представлять себе как некий общедоступный онлайн-журнал или дневник, в котором люди могут публиковать информацию о текущих событиях, высказывать собственное мнение или комментировать те или иные события — одним словом, обнародовать любую информацию, которую они хотят довести до сведения других людей.

Однако, публикуя ту или иную информацию в Интернете, вы, по сути, делитесь ею со всем миром. Пользователи не могут выкладывать информацию лишь для каких-то конкретных лиц или организаций (например, только для своих друзей или членов семьи). В результате появились социальные сети, самыми известными среди которых являются MySpace, Facebook, Twitter и Google+. С помощью этих систем пользователи могут поддерживать связь со своими друзьями, членами семьи, коллегами и другими людьми своего круга. Несмотря на то что каждая из этих служб позволяет пользователям выходить на одну и ту же аудиторию (с точки зрения пользователя), каждый из них по-своему уникален. Используемые ими методы взаимодействия людей существенно различаются между собой; следовательно, они, вообще говоря, не являются прямыми конкурентами друг другу. Ниже вкратце описана каждая из этих служб, а затем мы приступим к более детальному обсуждению Twitter.

Сеть MySpace предназначена главным образом для молодежи (школьников средних и старших классов), с акцентом на музыку; сеть Facebook поначалу ориентировалась на студентов университетов и колледжей, но в настоящее время она открыта для всех. Это более универсальная платформа, чем MySpace; ее сеть может служить хост-системой для приложений (многие считают, что именно эта возможность обеспечила высокую популярность Facebook). Twitter является службой *микроблогинга*,

посредством которого пользователи задают определенный статус (обычно некое мнение); отсюда сравнение с блогингом. Сеть Google+ представляет собой сравнительно недавнее вторжение компании Google в эту новую для себя область деятельности и попытку представить своим пользователям возможности, сопоставимые с возможностями других социальных сетей (однако в Google+ предусмотрены новые возможности, которые отсутствуют в других социальных сетях).

Среди типичных применений социальных сетей самые базовые реализованы в Twitter. Twitter можно использовать для публикации кратких статусных сообщений, называемых *твитами*. Другие пользователи могут следовать за вами, т.е. могут подписаться на ваши твиты. В свою очередь, вы можете следовать за твитами других пользователей, которые кажутся вам интересными.

Сеть Twitter называют службой микроблогинга, поскольку, в отличие от стандартного блога, который позволяет пользователям создавать посты любой длины, твиты ограничены лишь 140 символами на каждое обновление. Такое ограничение длины постов обусловлено главным образом тем, что сеть Twitter поначалу была ориентирована лишь на веб-сообщения и текстовые сообщения на мобильных телефонах посредством Short Message Service (SMS), которые сами по себе могут иметь лишь ограниченную длину (160 символов ASCII). Такое ограничение выгодно и самим пользователям, которым не приходится читать слишком длинные сообщения. К тому же это ограничение заставляет авторов постов выражать свои мысли точно и лаконично.

13.3.2. Сеть Twitter и язык Python

В интерфейсе API Twitter для Python предусмотрено несколько библиотек, которые выложены в документации разработчика Twitter на сайте <https://dev.twitter.com/docs/twitter-libraries#python>. Все они в чем-то похожи, а в чем-то различаются между собой. Поэтому мы рекомендуем вам поэкспериментировать с несколькими из них, чтобы найти такую библиотеку, которая в наибольшей степени отвечает вашему стилю и наклонностям. Итак, чтобы чересчур не ограничивать себя, в этой главе мы будем использовать Twython и Tweepy. Вы найдете их на сайтах <http://github.com/ryanmcgrath/twython> и <http://tweepy.github.com> соответственно.

Как и в случае большинства Python-пакетов, для установки в своей системе одной из этих Twitter-библиотек или обеих этих библиотек вы можете использовать либо `easy_install`, либо `pip`. Если хотите больше поэкспериментировать с этим кодом, исходные деревья для обеих библиотек выложены на GitHub. Как альтернативный вариант, можете просто загрузить самую последнюю версию `.tgz` или `.zip` с GitHub и вызвать типичную команду инсталляции `setup.py`:

```
$ sudo python setup.py install
Password:
running install
running bdist_egg
running egg_info
creating twython.egg-info
...
Finished processing dependencies for twython==1.4.4
```

Библиотекам, подобным Twython, понадобится определенная дополнительная помощь, чтобы они могли взаимодействовать с сетью Twitter. Это зависит от библиотек `httplib2`, `oauth2` и `simplejson`. (Последняя является внешней версией библиотеки `json`, которая присутствует в стандартной библиотеке, начиная с версии 2.6.)

Первые шаги

Для начала приведем краткий пример того, как пользоваться библиотекой Tweepy для выполнения поиска в Twitter:

```
# tweepy-example.py
import tweepy
results = tweepy.api.search(q='twython3k')
for tweet in results:
    print ' User: %s' % tweet.from_user
    print ' Date: %s' % tweet.created_at
    print ' Tweet: %s' % tweet.text
```

Если вы будете выполнять этот фрагмент кода на языке Python 2 — на момент написания этой книги библиотека Tweepy еще не реализована для Python 3, — задав запрос именно так, как показано в примере, то заметите, что данное условие поиска было выбрано именно для получения лишь нескольких результатов. Это означает, что вы увидите буквально пару твитов (на данный момент главным образом от вашего покорного слуги), возвращенных сетью Twitter (это относится к версии библиотеки Twython для Python 3):

```
$ python twython-example.py
User: @wespcy
Date: Tue, 04 Oct 2011 21:09:41 +0000
Tweet: Testing posting to Twitter using Twython3k (another story of
life on the bleeding edge)
```

```
User: @wespcy
Date: Tue, 04 Oct 2011 17:18:38 +0000
Tweet: @ryanmcgrath cool... thx! i also have a &quot;real&quot;
twython3k bug i need to file... will do it officially on github. just
giving you a heads-up!
```

```
User: @wespcy
Date: Tue, 04 Oct 2011 08:01:09 +0000
Tweet: @ryanmcgrath Hey ryan, good work on Twython thus far!
Can you pls drop twitter_endpoints.py into twython3k? It's out-ofdate.
thx! :-)
```

Вызов `search()` библиотеки Tweepy отыскивает соответствующие результаты в списке. Приведенный выше код просматривает твиты и отображает атрибуты, представляющие для вас интерес. Twython является аналогичной Python-библиотекой для интерфейса API Twitter.

Библиотека Twython похожа на Tweepy, хотя имеются некоторые отличия. Ее можно использовать как в Python 2, так и в Python 3, но для хранения результирующих данных она также использует чисто “питоновские” словари вместо объектов. Сравните `tweepy-example.py` с этим сценарием, `twython-example.py`, который также совместим с Python 2 и Python 3:

```
# twython-example.py
from distutils.log import warn as printf
try:
    import twython
except ImportError:
    import twython3k as twython
```



```
TMPL = '''\
    User: @%(from_user)s
    Date: %(created_at)s
    Tweet: %(text)s
'''

twitter = twython.Twython()
data = twitter.searchTwitter(q='twython3k')
for tweet in data['results']:
    printf(TMPL % tweet)
```

Функция `distutils.log.warn()` исполняет роль нашего прокси для оператора `print` (в языке Python 2) и функции (в языке Python 3). Мы также пытаемся импортировать обе библиотеки `Twython` (для Python 2 и Python 3) в надежде, что это удастся.

Что же касается вывода, то результатом `Twython.searchTwitter()` является словарь, а объектом, размещенным в ключе `results`, — список словарей, каждый из которых представляет результирующий твит. Это позволяет нам упростить отображение, поскольку результаты являются словарями — т.е. речь идет о более простом вызове. (Платой за это упрощение будет шаблон строки, где нам придется расширить ключевые переменные.)

Еще одним изменением, внесенным здесь, является то, что вместо лишь одиночных строк вывода мы поместили все строки вместе в один большой шаблон строк, а затем передали в него выходной словарь. Причиной этого является то, что на практике вы скорее всего будете в любом случае использовать ту или иную разновидность шаблона (либо строчной, либо веб-шаблон).

Как нетрудно догадаться, этот вывод окажется идентичным выводу, полученному при использовании библиотеки `Твееру`, поэтому мы решили не дублировать его здесь.

13.3.3. Пример API с более длинной комбинацией

Эти простые маленькие фрагменты кода хороши для того, чтобы быстро ввести читателей в курс дела и показать, как пользоваться библиотеками `Твееру` и `Twython`. Однако на практике вам наверняка встретится сценарий, в котором потребуется объединить и использовать несколько подобных API. Сейчас мы рассмотрим расширенный пример. Мы создадим совместимую библиотеку, которая поддерживает набор базовых команд Twitter, воспользовавшись для этого как `Твееру`, так и `Twython`. Это упражнение поможет вам изучить обе указанные библиотеки, а также лучше познакомиться с интерфейсом API Twitter.

Идентификация пользователя

Для того чтобы приступить к выполнению этого упражнения, вам понадобится учетная запись в сети Twitter. Обратитесь на сайт <http://twitter.com> и зарегистрируйтесь на нем, если вы не успели сделать это раньше. Для идентификации пользователя необходимо указать соответствующее имя пользователя и пароль. (Более современные решения включают также биометрическую идентификацию путем снятия отпечатков пальцев или сканирования сетчатки глаза.) Указав свое имя пользователя и пароль, вы подтверждаете свое право пользоваться данной системой. Для доступа к данным необходимо авторизоваться.

Авторизация

То обстоятельство, что вы надлежащим образом идентифицировали себя в системе, еще не означает, что вы получили доступ к данным (своим собственным или чьим-либо еще). Для этого необходимо пройти процедуру *авторизации*. Авторизация нужна для того, чтобы вы могли получить доступ к данным (своим собственным или чьим-либо еще); впрочем, вы можете авторизовать какого-то другого пользователя, чтобы он получил доступ к данным от вашего имени (например, разрешить какому-либо внешнему приложению загрузить ваш Twitter-поток или передать на вашу учетную запись в сети Twitter обновление статуса).

Для того чтобы получить свой мандат (полномочия) на авторизацию в сети Twitter, вам нужно создать приложение. Это можно сделать на сайте <http://dev.twitter.com>. После того как у вас появится хотя бы одно приложение, щелкните на том из них, мандатом которого вы хотели бы воспользоваться. Соответствующий указатель URL будет иметь примерно такой вид: <http://dev.twitter.com/apps/APP-ID/show>. Здесь вы найдете четыре важных элемента, которые понадобятся вам для доступа к данным в сети Twitter: ваши параметры OAuth включают ваш *ключ потребителя* и *секрет потребителя*. Здесь вы также получите свой *маркер доступа* и *секрет маркера доступа*, которые обеспечивают вам доступ к вашим данным в сети Twitter.

Скопируйте эти четыре важных элемента данных и сохраните их в каком-нибудь надежном месте (разумеется, не в своем исходном коде!). В данном примере я сохранил их как четыре глобальные переменные в модуле под названием `tweet_auth.py`, который я буду импортировать из нашего будущего приложения. На практике вы, наверное, либо только разошлете файл `.рус`, скомпилированный в виде байт-кода (но не простого текста!), либо сделаете его доступным посредством какой-либо базы данных или в каком-то другом месте сети (возможно, в зашифрованном виде). После этой подготовки я сначала опишу приложение, а затем продемонстрирую соответствующий код.

Гибридное приложение для интерфейса API Twitter

Это приложение выполняет четыре операции: сначала выполняет поиск в сети Twitter и распечатывает результаты этого поиска; затем выясняет некоторые подробности о текущем пользователе и распечатывает эту информацию; потом извлекает временную последовательность сообщений о статусе текущего пользователя и распечатывает их; и, наконец, помещает твит от имени текущего пользователя. Все эти четыре операции выполняются дважды: один раз с использованием библиотеки Твееру и еще раз с использованием библиотеки Twython. Для этого нам нужно обеспечить поддержку четырех команд API Twitter, как показано в табл. 13.2.

Таблица 13.2. Четыре команды гибридного приложения интерфейса API Twitter

Команда	Описание
<code>Search</code>	Получает поисковый запрос и выполняет поиск в сети Twitter самых последних твитов, которые удовлетворяют вашему условию поиска. Представляет собой неаутентифицированный вызов (единственный вызов такого рода в нашем приложении); это означает, что его может выполнить любой пользователь в любой момент времени
<code>verify_credentials</code>	Запрашивает в сети Twitter текущую информацию, касающуюся аутентифицируемого пользователя
<code>user_timeline</code>	Получает самые последние твиты от аутентифицируемого пользователя
<code>update_status</code>	Обновляет статус от аутентифицируемого пользователя — да, создается новый твит от вашего имени

Это приложение будет поддерживать как библиотеку Tweepy, так и библиотеку Twython и будет выполняться и в среде Python 2, и в Python 3. Соответствующий код будет состоять из поддерживаемых команд, шагов по инициализации обеих библиотек, а затем включать код, который поддерживает каждую из команд. Готовы? Рассмотрим файл `twapi.py` в примере 13.2.

Пример 13.2. Сочетание библиотек API Twitter (`twapi.py`)

Демонстрация взаимодействия с сетью Twitter путем использования библиотек Tweepy и Twython.

```

1  #!/usr/bin/env python
2
3  from distutils.log import warn as printf
4  from unittest import TestCase, main
5  from tweet_auth import *
6
7  # set up supported APIs
8  CMDs = {
9      'twython': {
10         'search': 'searchTwitter',
11         'verify_credentials': None,
12         'user_timeline': 'getUserTimeline',
13         'update_status': None,
14     },
15     'tweepy': dict.fromkeys((
16         'search',
17         'verify_credentials',
18         'user_timeline',
19         'update_status',
20     )),
21 }
22 APIs = set(CMDs)
23
24 # удаляем недоступные интерфейсы API
25 remove = set()
26 for api in APIs:
27     try:
28         __import__(api)
29     except ImportError:
30         try:
31             __import__('%s3k' % api)
32         except ImportError:
33             remove.add(api)
34
35 APIs.difference_update(remove)
36 if not APIs:
37     raise NotImplementedError(
38         'No Twitter API found; install one & add to CMDs!')
39
40 class Twitter(object):
41     'Twitter -- Use available APIs to talk to Twitter'
42     def __init__(self, api, auth=True):
43         if api not in APIs:
44             raise NotImplementedError(
45                 '%r unsupported; try one of: %r' % (api, APIs))
46
47     self.api = api

```

```

48     if api == 'twython':
49         try:
50             import twython
51         except ImportError:
52             import twython3k as twython
53         if auth:
54             self.twitter = twython.Twython(
55                 twitter_token=consumer_key,
56                 twitter_secret=consumer_secret,
57                 oauth_token=access_token,
58                 oauth_token_secret=access_token_secret,
59             )
60         else:
61             self.twitter = twython.Twython()
62     elif api == 'tweepy':
63         import tweepy
64         if auth:
65             auth = tweepy.OAuthHandler(consumer_key,
66                                         consumer_secret)
67             auth.set_access_token(access_token,
68                                   access_token_secret)
69             self.twitter = tweepy.API(auth)
70         else:
71             self.twitter = tweepy.api
72
73     def _get_meth(self, cmd):
74         api = self.api
75         meth_name = CMDs[api][cmd]
76         if not meth_name:
77             meth_name = cmd
78             if api == 'twython' and '_' in meth_name:
79                 cmds = cmd.split('_')
80                 meth_name = '%s%s' % (cmds[0], cmds[1].title())
81         return getattr(self.twitter, meth_name)
82
83     def search(self, q):
84         api = self.api
85         if api == 'twython':
86             res = self._get_meth('search')(q=q)['results']
87             return (ResultsWrapper(tweet) for tweet in res)
88         elif api == 'tweepy':
89             return (ResultsWrapper(tweet)
90                     for tweet in self._get_meth('search')(q=q))
91
92     def verify_credentials(self):
93         return ResultsWrapper(
94             self._get_meth('verify_credentials')())
95
96     def user_timeline(self):
97         return (ResultsWrapper(tweet)
98                 for tweet in self._get_meth('user_timeline')())
99
100    def update_status(self, s):
101        return ResultsWrapper(
102            self._get_meth('update_status')(status=s))
103
104    class ResultsWrapper(object):
105        "ResultsWrapper -- makes foo.bar the same as foo['bar']"

```

```
106     def __init__(self, obj):
107         self.obj = obj
108
109     def __str__(self):
110         return str(self.obj)
111
112     def __repr__(self):
113         return repr(self.obj)
114
115     def __getattr__(self, attr):
116         if hasattr(self.obj, attr):
117             return getattr(self.obj, attr)
118         elif hasattr(self.obj, '__contains__') and attr in self.obj:
119             return self.obj[attr]
120         else:
121             raise AttributeError(
122                 '%r has no attribute %r' % (self.obj, attr))
123
124     __getitem__ = __getattr__
125
126 def _demo_search():
127     for api in APIs:
128         printf(api.upper())
129         t = Twitter(api, auth=False)
130         tweets = t.search('twython3k')
131         for tweet in tweets:
132             printf('----' * 10)
133             printf('@%s' % tweet.from_user)
134             printf('Status: %s' % tweet.text)
135             printf('Posted at: %s' % tweet.created_at)
136         printf('----' * 10)
137
138 def _demo_ver_creds():
139     for api in APIs:
140         t = Twitter(api)
141         res = t.verify_credentials()
142         status = ResultsWrapper(res.status)
143         printf('@%s' % res.screen_name)
144         printf('Status: %s' % status.text)
145         printf('Posted at: %s' % status.created_at)
146         printf('----' * 10)
147
148 def _demo_user_timeline():
149     for api in APIs:
150         printf(api.upper())
151         t = Twitter(api)
152         tweets = t.user_timeline()
153         for tweet in tweets:
154             printf('----' * 10)
155             printf('Status: %s' % tweet.text)
156             printf('Posted at: %s' % tweet.created_at)
157             printf('----' * 10)
158
159 def _demo_update_status():
160     for api in APIs:
161         t = Twitter(api)
162         res = t.update_status(
163             'Test tweet posted to Twitter using %s' % api.title())
```

```

164         printf('Posted at: %s' % res.created_at)
165         printf('----' * 10)
166
167     # object wrapper unit tests
168     def _unit_dict_wrap():
169         d = {'foo': 'bar'}
170         wrapped = ResultsWrapper(d)
171         return wrapped['foo'], wrapped.foo
172
173     def _unit_attr_wrap():
174         class C(object):
175             foo = 'bar'
176             wrapped = ResultsWrapper(C)
177             return wrapped['foo'], wrapped.foo
178
179     class TestSequenceFunctions(TestCase):
180         def test_dict_wrap(self):
181             self.assertEqual(_unit_dict_wrap(), ('bar', 'bar'))
182
183         def test_attr_wrap(self):
184             self.assertEqual(_unit_attr_wrap(), ('bar', 'bar'))
185
186     if __name__ == '__main__':
187         printf('\n*** SEARCH')
188         _demo_search()
189         printf('\n*** VERIFY CREDENTIALS')
190         _demo_ver_creds()
191         printf('\n*** USER TIMELINE')
192         _demo_user_timeline()
193         printf('\n*** UPDATE STATUS')
194         _demo_update_status()
195         printf('\n*** RESULTS WRAPPER')
196         main()

```

Прежде чем мы приступим к разбору этого сценария, выполним его и посмотрим, что у нас получится на выходе. Обязательно создайте сначала файл `tweet_auth.py` с этими переменными (и исправьте соответствующие значения для вашего Twitter-приложения):

```

# tweet_auth.py
consumer_key = 'SOME_CONSUMER_KEY'
consumer_secret = 'SOME_CONSUMER_SECRET'
access_token = 'SOME_ACCESS_TOKEN'
access_token_secret = 'SOME_ACCESS_TOKEN_SECRET'

```

Теперь мы готовы к дальнейшим действиям. Естественно, приведенный ниже результат на выходе был получен в ходе этого одного прогона, выполненного в момент написания данной книги. Результат, полученный вами, несомненно, будет отличаться от моего. Вот что получилось, когда мы выполнили этот код (“...” означает, что мы, в целях большей краткости, урезали распечатку результата, полученного на выходе):

```

$ twapi.py

*** SEARCH
TWYTHON
-----

```

```
@ryanmcgrath
Status: #twython is now version 1.4.4; should fix some utf-8 decoding issues, twython3k
should be caught up, etc: http://t.co/s6fTVh0P /cc
@wespcy
Posted at: Thu, 06 Oct 2011 20:25:17 +0000
-----
@wespcy
Status: Testing posting to Twitter using Twython3k (another story of life on the
bleeding edge)
Posted at: Tue, 04 Oct 2011 21:09:41 +0000
-----
@wespcy
Status: @ryanmcgrath cool... thx! i also have a &quot;real&quot;
twython3k bug i need to file... will do it officially on github. just giving you a
heads-up!
Posted at: Tue, 04 Oct 2011 17:18:38 +0000
-----
@wespcy
Status: @ryanmcgrath Hey ryan, good work on Twython thus far! Can you pls drop twitter_
endpoints.py into twython3k? It's out-of-date. ..
thx! :-)
Posted at: Tue, 04 Oct 2011 08:01:09 +0000
-----
TWEETPY
-----
@ryanmcgrath
Status: #twython is now version 1.4.4; should fix some utf-8 decoding issues, twython3k
should be caught up, etc: http://t.co/s6fTVh0P /cc
@wespcy
Posted at: 2011-10-06 20:25:17
. . .
-----
*** VERIFY CREDENTIALS
@wespcy
Status: .@imusicmash That's great that you're enjoying corepython.com!
Note: there will be lots of cookies at #SVCC: yfrog.com/khlazqznj
Posted at: Fri Oct 07 22:37:37 +0000 2011
-----
@wespcy
Status: .@imusicmash That's great that you're enjoying corepython.com!
Note: there will be lots of cookies at #SVCC: yfrog.com/khlazqznj
Posted at: 2011-10-07 22:37:37
-----
*** USER TIMELINE
TWYTHON
-----
Status: .@imusicmash That's great that you're enjoying corepython.com!
Note: there will be lots of cookies at #SVCC: yfrog.com/khlazqznj
Posted at: Fri Oct 07 22:37:37 +0000 2011
-----
Status: SFBayArea: free technical conference w/free
food+drinks+parking this wknd! I'm doing #Python & @App_Engine http://
t.co/spvVjYUA
Posted at: Fri Oct 07 15:20:46 +0000 2011
-----
Status: RT @GoogleCode @Google Cloud SQL: your database in the cloud
http://t.co/4wt2cjpH @app_engine #mysql
Posted at: Thu Oct 06 20:12:26 +0000 2011
-----
```

```

. . .
Status: Watch this: http://t.co/pm2QCLtW Read this: http://t.co/
Om5TtLZP Note the 2 paragraphs that start w/"No one wants to die"
Posted at: Thu Oct 06 00:36:27 +0000 2011
-----
Status: I'm wondering: will future Apple products visually be designed
as well & have as much impact on the market? What do you think?
Posted at: Thu Oct 06 00:02:16 +0000 2011
-----
. . .
-----
TWEETPY
-----
Status: .@imusicmash That's great that you're enjoying corepython.com!
Note: there will be lots of cookies at #SVCC: yfrog.com/khlazqznj
Posted at: 2011-10-07 22:37:37
. . .
-----
*** UPDATE STATUS
Posted at: Sat Oct 08 05:18:51 +0000 2011
-----
Posted at: 2011-10-08 05:18:51
-----
*** RESULTS WRAPPER
..
-----
-
Ran 2 tests in 0.000s
OK
$

```

Нетрудно заметить, что мы прошлись по всем четырем функциям, выполнив их с каждой библиотекой. Выполнение этого сценария на компьютере под управлением операционной системы Windows приводит к получению такого же результата (конечно, если все установлено правильно). Поскольку наш код также совместим с Python 3, вы также должны получить аналогичный результат на выходе; однако вы должны увидеть результат только от Twython, поскольку (на момент написания данной книги) библиотека Твеэру еще не адаптирована к Python 3. Итак, приступим к подробному рассмотрению этого исходного кода.

Построчное объяснение

Строки 1–5

Данная совокупность операторов импортирования включает пару операторов из стандартной библиотеки (использующих `distutils.log.warn()` в качестве модуля доступа для оператора или функции `print`, в зависимости от того, используете ли вы Python 2 или 3, плюс базовые атрибуты для выполнения поблочного тестирования в Python) и наш мандат для авторизации в сети Twitter.

Мы также хотим напомнить вам о том, что использование `from module import *` (строка 5), вообще говоря, не рекомендуется, поскольку стандартная библиотека или пакеты сторонних разработчиков могут использовать такие же имена переменных, что и ваши модули, что может создать вам определенные проблемы. В данном случае мы имеем полный контроль над файлом `tweet_auth.py` и знаем обо всех (четырех) его переменных. Его единственное назначение заключается в том, чтобы скрыть

мандат (полномочия) пользователя. На практике такой файл устанавливался бы лишь в производственной среде как скомпилированный в виде байт-кода (.рус) или оптимизированного (.руо) файла, однако и тот и другой нечитабельны; их источником могла бы быть какая-либо база данных или сетевой вызов.

Строки 7–38

Первое реальное тело кода выполняет единственную функцию: определяет, какие клиентские библиотеки Twitter доступны для интерпретатора Python, который выполняет ваш код.

Объект `CMD` — это словарь, в котором есть по одному входу на каждую поддерживаемую библиотеку. В нашем случае имеются библиотеки `Twython` (`twython`) и `Tweepy` (`tweepy`). В одном из упражнений, приведенных в конце этой главы, вам предстоит добавить поддержку третьей библиотеки.

Для каждой библиотеки мы обеспечиваем имена методов, которые представляют соответствующие команды интерфейса API Twitter. Эти команды вызываются для четырех упоминавшихся ранее функций, которые мы хотим поддерживать. Если соответствующее значение равно `None`, то это означает, что имя метода в точности совпадает; таким образом, любое “реальное” значение (т.е. значение, которое *не* равняется `None`) представляет исключение из данного правила.

Библиотека `Tweepy` облегчает нашу задачу, поскольку имена ее методов совпадают с командами. Именно поэтому мы используем `dict.fromkeys()` для создания словаря со всеми значениями `None` для его ключей. С библиотекой `Twython` ситуация несколько сложнее, поскольку она использует имена, которые задействуют применение прописных букв (camel capitalization) и являются исключениями из данного правила. Описание того, как получают эти имена и выбираются методы, можно найти в строках 71–80.

В строке 22 мы упорядочиваем все поддерживаемые интерфейсы API в виде одного набора. Эта структура данных является самой быстрой проверкой членства в этом языке. С помощью этой переменной мы также выполняем цикл по каждому из интерфейсов API. К этому моменту мы создали лишь возможные интерфейсы API; сейчас мы должны оставить только то, что у нас реально имеется в наличии.

Код в строках 25–33 пытается импортировать каждую из библиотек и удаляет те, которые невозможно найти, собирая несуществующие интерфейсы API в другую совокупность под названием `remove`. После завершения этого цикла мы знаем точно, чего у нас нет, и убираем все это (в строке 35) из полной совокупности API. Если оказывается, что библиотек, к которым можно было бы обратиться, в нашем распоряжении нет, мы вызываем `NotImplementedError` в строках 36–38.

Строки 40–71

Класс `Twitter` является первичным объектом в этом приложении. Мы определяем инициализатор, который берет `api` (в данном случае либо “`twython`”, либо “`tweepy`”) и необязательный флажок `auth`. По умолчанию значением этого флажка является `True`, поскольку большую часть времени нам требуется (аутентифицированный и) авторизованный доступ к данным пользователя. (Поиск является единственной функцией, которая не требует аутентификации.) Выбранный интерфейс API мы кешируем в `self.api`.

Оставшийся код этого раздела (строки 48–71) создает экземпляр конечной точки Twitter, основываясь на выбранном интерфейсе API и параметрах аутентификации. Созданный экземпляр затем присваивается `self.twitter`. Этот объект является инструментом, с помощью которого мы выполняем команды API Twitter.

Строки 73–81

Метод `_get_meth()` занимается составлением правильного имени метода для вызова каждого интерфейса API. Обратите внимание: перед именем этой функции мы добавляем одиночный символ подчеркивания (`_`), который указывает на то, что данная функция не должна вызываться пользователями. Эту функцию следует рассматривать как внутренний метод, который должен вызываться каким-либо из других методов в нашем классе.

В этом методе мы могли бы использовать конструкцию `self.api`, однако для обеспечения быстрого доступа было бы предпочтительнее присвоить атрибуты часто используемого экземпляра какой-либо локальной переменной. Использование `self.api` требует двух просмотров, тогда как `api` — только одного просмотра. С точки зрения затрат времени центрального процессора один дополнительный просмотр обойдется нам не так уж дорого, но при использовании цикла и/или в случае частого выполнения затраты времени центрального процессора могут существенно возрасти. Именно этим соображением обусловлено присвоение локальной переменной в строке 74.

В следующей строке извлекается подходящая команда из запрашиваемого интерфейса API и присваивается `meth_name`. Если ее значение равно `None`, то по умолчанию предполагается, что имя метода совпадает с именем команды. В случае библиотеки Твееру проблем не возникает: мы уже указывали, что имена ее методов совпадают с именами команд. Следующая совокупность строк отвечает за обработку особых ситуаций, для которых нам придется получить правильное имя.

Как указывалось выше, библиотека `Twython` использует для своих слов прописные буквы (camel capitalization) вместо применения в качестве разделителя символа подчеркивания (`_`). Это означает, что мы должны заканчивать каждое слово на символе подчеркивания, отображать второе слово прописными буквами, а затем присоединять его к первому слову (строки 79–80). Последним действием является использование этого имени и извлечение соответствующего объекта метода из запрашиваемого интерфейса API; речь идет об объекте первого класса, который возвращается непосредственно на вызывающую функцию.

Строки 83–102

В этих четырех функциях реализуются четыре поддерживаемые команды интерфейса API Twitter: `search()`, `verify_credentials()`, `user_timeline()`, `update_status()`. Помимо функции `search()`, другие команды достаточно просты и идентичны в обеих библиотеках. Рассмотрим сначала эти, более простые команды, а затем займемся более сложной командой `search()`.

Проверка мандата (полномочий) аутентифицируемого пользователя — единственное, что вы можете сделать с помощью команды `verify_credentials()`. Для вас это также самый быстрый способ оценить программистским путем свой самый последний твит. Предоставленная вами информация о пользователе приходит в ответ и переносится на новую строку с помощью команды `ResultsWrapper` (подробнее об этом мы расскажем ниже), а затем возвращается на вызывающую функцию. Подробнее об использовании этой команды можно прочитать на сайте http://dev.twitter.com/docs/api/1/get/account/verify_credentials.

История твитов пользователя включает самые последние из его твитов (и ретвитов). Команда интерфейса API Twitter `user_timeline` по умолчанию возвращает 20 самых последних твитов пользователя, но вы можете запросить до 200 твитов, воспользовавшись параметром `count`, который мы не используем в своем примере

(впрочем, вам придется воспользоваться им в другом упражнении, которое помещено в конце этой главы). Подробнее об использовании этой функции можно прочитать на сайте http://dev.twitter.com/docs/api/1/get/statuses/user_timeline. В отличие от команды `verify_credentials()`, мы переносим на новую строку каждый отдельный возвращаемый твит, а не весь результат, приходящий от сети Twitter, и возвращаем выражение-генератор, обеспечивающее циклический просмотр твитов.

Самой основной функцией сети Twitter является обновление пользователями своих статусов; по-другому это называется твитингом. Можно утверждать, что без этой функции сеть Twitter теряет смысл. Вы можете видеть, что команда `update_status()` принимает дополнительный параметр `s`, который представляет собой текст соответствующего твита. Возвращаемое значение — это сам твит (он также переносится на новую строку с помощью `ResultsWrapper`), причем самой важной характеристикой является поле `created_at`, которое подтверждает, что соответствующий твит действительно был создан и опубликован. Вы увидите `created_at` в следующем коде, который демонстрирует эту функцию в действии. Подробнее об использовании этой функции можно прочитать на сайте <http://dev.twitter.com/docs/api/1/post/statuses/update>.

Теперь вернемся к поиску. Именно здесь наблюдается различие между двумя указанными интерфейсами API, и именно поэтому мы видим здесь больше кода, чем обычно. Twython пытается быть честной и интерпретировать результаты, полученные от сети Twitter, практически дословно, преобразуя JavaScript Object Notation (JSON) в словарь Python (которые являются “близкими родственниками”). Эта структура данных содержит разные метаданные, а также “реальные блага”, интересующие вас под ключом `results` (именно поэтому нам нужно выполнить просмотр в строке 86).

Библиотека Tweepy, с другой стороны, демонстрирует более реалистичный подход и возвращает эти результаты поиска непосредственно в объекте, имеющем вид списка, — фактически `ResultsSet` представляет собой подкласс списка, — поскольку его разработчик понимает, что это именно то, что вам нужно. Это делает его более удобным и избавляет вас от необходимости дальнейших просмотров. Что можно сказать обо всех этих дополнительных метаданных? Они являются не чем иным, как атрибутами этого возвращаемого `ResultsSet`.

Строки 104–124

Этот очередной фрагмент кода представляет собой класс общего назначения, который вы можете использовать где угодно за пределами этого приложения. Он не имеет никакого отношения к библиотекам Twitter или чему-либо еще, поскольку он используется лишь как удобный инструмент для наших пользователей в том смысле, что обеспечивает единый интерфейс с объектами, возвращаемыми из этих библиотек.

Не приходилось ли вам когда-либо испытывать разочарование объектами, подобными либо словарям, либо объектам? Я имею в виду, что в первом случае вам приходится делать нечто, эквивалентное вызову `_getitem()`, чтобы получить определенное значение, т.е. `foo['bar']`, а во втором случае приходится иметь дело с объектами, которые имеют интерфейс для атрибутов, т.е. `foo.bar`. Наверное, было бы просто замечательно, если бы у вас была возможность использовать какой-либо из этих двух вариантов для всех объектов и не задумываться над этим снова и снова. Именно эту задачу решает класс `ResultsWrapper`.

Эта идея пришла мне в голову во время написания данной книги. Возможно, это еще нельзя считать идеальным вариантом, но моя идея заключается в том, чтобы взять любой объект Python и завернуть его в объект, который делегирует выполнение

просмотра (посредством `_getitem_()` или `_getattr_()`) завернутому объекту. (Обзор делегирования можно найти в главе, посвященной объектно-ориентированному программированию, в книге *Core Python Programming* или *Core Python Language Fundamentals*.)

В инициализаторе (строки 106–107) мы заворачиваем объект. Затем идут все его строковые представления (строки 109–113). Многое происходит внутри `_getattr_()`. Когда на какой-либо атрибут поступает запрос, который невозможно распознать, `_getattr_()` проверяет, существует ли этот атрибут в завернутом объекте (строки 116–117). Если там его нет, то, возможно, речь идет об объекте, подобном словарю, поэтому нужно проверить, является ли он “ключом” (строки 118–119). Естественно, прежде чем использовать оператор `in`, нам нужно проверить, поддерживает ли данный объект этот тип проверки или доступа; для этого нужно сначала выяснить, есть ли у этого объекта атрибут `_contains_`. Если все эти проверки не приводят к положительному результату, то вам придется сообщить пользователю неутешительные новости (строки 120–122).

Последняя строка (124) в этом фрагменте кода, которая нуждается в разъяснении, предназначена на тот случай, когда пользователь пытается получить доступ к какому-либо из атрибутов так, как это обычно делается при использовании словаря; при этом в качестве ключа используется имя соответствующего атрибута. Именно поэтому нам требуется точно такое же поведение, как у `_getattr_()`. В таком случае объект какого бы типа ни завертывался, мы можем извлечь и вернуть именно то, что требуется пользователю.

Строки 126–165

Цель функций `_demo_*()` заключается в том, чтобы делать именно то, на что указывает их название: `_demo_search()` демонстрирует поиск термина “twython3k” с помощью всех имеющихся в наличии API, а затем отображает данные результирующих твитов; `_demo_ver_creds()` выполняет команду `verify_credentials` и отображает самый последний твит аутентифицируемого пользователя; `_demo_user_timeline()` выдает 20 самых последних твитов, отображая текст и временную метку каждого из этих твитов. Наконец, `_demo_update_status()` передает новые твиты, описывая API, который использовался для этой цели.

Строки 167–184

Этот фрагмент кода предназначен для тестирования класса `ResultsWrapper`. Каждая из функций `_unit_*_wrap()` тестирует словари завертывания (или объекты типа словарей), а также объекты с интерфейсом атрибутов. Результат применения обеих форм доступа к атрибутам, либо с помощью `obj['foo']`, либо `obj.foo`, должен быть один и тот же: “bar”. Эта проверка достоверности выполняется классом `unittest`, `TestSequenceFunctions` (строки 179–184).

Строки 186–196

Эти строки `main()` отображают, какая именно функция тестируется, и вызывают конкретные функции `_demo_*()`, которые отображают их собственный выходной результат. Последним вызовом является обращение к функции `unittest.main()`, которая выполняет тесты отдельных блоков.

13.3.4. Резюме

Надеемся, что, ознакомившись с материалом этого раздела, вы составили определенное представление о взаимодействии пользователя с парой служб, реализованных к настоящему времени в Интернете: сервером биржевых котировок Yahoo! и сетью Twitter. Важно понимать, что интерфейс Yahoo! полностью определяется URL и не предполагает аутентификации, тогда как сеть Twitter обеспечивает полноценный интерфейс API REST и авторизацию OAuth для гарантирования безопасного доступа к данным. Нам удалось воспользоваться сильными сторонами того и другого, используя для решения нашей задачи вполне читабельный и эффективный код Python.

Мы рассмотрели лишь пару веб-служб; в действительности таких служб существует гораздо больше. Мы еще вернемся к этим двум веб-службам в главе 14.

13.3.5. Дополнительные интернет-ресурсы

Yahoo! Finance

- <http://gummy-stuff.org/Yahoo-data.htm>
- <http://gummy-stuff.org/forex.htm>

Twitter

- <http://dev.twitter.com/docs/twitter-libraries#python>.
- <http://github.com/ryanmcgrath/twython>
- <http://tweepy.github.com>

13.4. Упражнения

Веб-службы

- 13.1. Веб-службы.** Опишите своими словами, что собой представляют веб-службы. Найдите некоторые из этих служб в Интернете и опишите, как они работают. Что собой представляет их интерфейс API и как вы осуществляете доступ к таким данным? Требуется ли аутентификация или авторизация?
- 13.2. REST и веб-службы.** Выясните, как используются стиль REST, язык XML и формат JSON в приложениях и интерфейсах API более современных веб-служб. Опишите дополнительные возможности, которые они предоставляют по сравнению с более старыми системами, такими как сервер биржевых котировок Yahoo!, которые используют параметры URL.
- 13.3. REST и веб-службы.** Разработайте платформу приложений, воспользовавшись поддержкой Python для REST и XML, которая позволит вам поделиться этим кодом и повторно использовать его при написании приложений, которые используют какие-либо из более новых веб-служб и интерфейсов API, имеющихся на данный момент. Отобразите свой код, воспользовавшись интерфейсами API компаний Yahoo!, Google, eBay и Amazon.

Упражнения 13.4–13.11 предусматривают внесение изменений в пример с биржевыми котировками Yahoo! (`stock.py`), представленный ранее в этой главе.

- 13.4. Веб-службы.** Внесите изменения в `stock.py`, которые позволяли бы загружать другие данные биржевых котировок, учитывая дополнительные параметры, перечисленные в табл. 13.1. Вы можете просто взять `stock.py` в том виде, как было показано ранее в этой главе, и добавить новые функции.
- 13.5. Обработка строк.** Вы, наверное, обратили внимание, что некоторые из возвращаемых полей содержат котировки, загромождая таким образом выводимую информацию. Удалите эти котировки. Существует ли какой-либо другой способ избавиться от этих котировок, помимо решения, выбранного вами?
- 13.6. Обработка строк.** Не все биржевые тикеры являются четырехбуквенными. Аналогично не все цены акций (в расчете на одну акцию) находятся в диапазоне от 10 до 99,99 долл. То же самое можно сказать о суточном изменении цены и изменении цены, выраженном в процентах. Внесите необходимые изменения в свой сценарий с тем, чтобы, даже если бы поля вывода представляли собой строки разной длины, выходная информация по-прежнему форматировалась правильно, с надлежащим выравниванием и по одинаковому принципу для всех акций. Ниже приведен соответствующий пример:

```
C:\py>python stock.py
```

Prices quoted as of: Sat Oct 29 02:38:53 2011

Тикер	Цена в долларах	Изменение	Изменение в процентах
YHO0	16,56	-0,07	-0,42
GOOG	600,14	+1,47	+0,25
T	29,74	+0,60	+2,06
AMZN	217,32	+10,54	+5,10
BAC	7,35	+0,30	+4,26
BRK-B	79,96	+0,065	+0,08

- 13.7. Файлы.** Измените это приложение таким образом, чтобы данные биржевых котировок сохранялись в файле, а не отображались на экране. **Дополнительное задание:** можете изменить этот сценарий таким образом, чтобы пользователи могли выбирать между отображением данных биржевых котировок на экране и их сохранением в файле.
- 13.8. Веб-службы и модуль csv.** Преобразуйте `stock.py` таким образом, чтобы перейти от использования обычного цикла `for` и синтаксического анализа данных вручную к использованию модуля `csv` для автоматического выполнения синтаксического анализа поступающих данных, подобно тому, как мы делали это в соответствующем примере фрагмента кода.
- 13.9. Устойчивость.** Обычно Yahoo! время от времени изменяет имя хоста загрузки. Сегодня это может быть, например, `quote.yahoo.com`, а завтра `finance.yahoo.com`. Сегодня модным именем хоста загрузки является `download.finance.yahoo.com`. Иногда мы являемся свидетелями возвращения к старым именам хоста загрузки. Сделайте свое приложение

(приложения) устойчивым, поддерживая целый список таких имен хоста загрузки, и проверьте доступность веб-серверов в этих хостах (с помощью команды ping), чтобы выяснить, правильны ли они, прежде чем пытаться получить котировки. Вы можете также периодически анализировать ту или иную страницу с биржевыми котировками Yahoo! и получать имя хоста из ссылки Download Data в разделе Toolbox в нижней части страницы.

- 13.10. *Расширение интерфейса API.* На сервере биржевых котировок Yahoo! есть гораздо больше команд. Для того чтобы увидеть более полный их перечень, обратитесь на сайт <http://gummy-stuff.org/Yahoo-data.htm>. Выберите несколько новых пунктов данных и включите их в свой сценарий `stock.py`.
- 13.11. *Python 3.* Перенесите файл `stock.py` в среду Python 3 и назовите его `stock3.py`. Плюс один балл: предложите решение, которое работало бы как на версиях 2.x, так и на версиях 3.x, и опишите любые специальные приемы и методы, которыми вы пользовались для этого.
- 13.12. *Курсы зарубежных валют.* Сервер биржевых котировок Yahoo! может также предоставлять информацию о курсах обмена зарубежных валют. Обратитесь на сайт <http://gummy-stuff.org/forex.htm> и разработайте новый сценарий `forex.py`, который предоставлял бы такую информацию.
- 13.13. *График изменения цен на бирже.* Yahoo! также позволяет автоматически формировать графики изменения цен на бирже. Ниже приведен перечень URL, обратившись на которые вы можете составить представление об этой службе:

Малый график:

1 сутки: <http://chart.yahoo.com/t?s=GOOG>

5 суток: <http://chart.yahoo.com/v?s=GOOG>

1 год: <http://chart.yahoo.com/c/bb/m/GOOG>

Большой график:

1 сутки: <http://chart.yahoo.com/b?s=GOOG>

5 суток: <http://chart.yahoo.com/w?s=GOOG>

3 месяца: <http://chart.yahoo.com/c/3m/GOOG>

6 месяцев: <http://chart.yahoo.com/c/6m/GOOG>

1 год: <http://chart.yahoo.com/c/1y/GOOG>

2 года: <http://chart.yahoo.com/c/2y/GOOG>

5 лет: <http://chart.yahoo.com/c/5y/GOOG>

Максимум: <http://chart.yahoo.com/c/my/GOOG>

Подобно нашему примеру обеспечения устойчивости (упражнение 13.9), chart.yahoo.com, ichart.yahoo.com и ichart.finance.yahoo.com сейчас равноценны, поэтому любым из них можно пользоваться для получения требуемых данных. Напишите приложение, которое позволяло бы пользователям генерировать графики для портфелей их акций. Кроме того, обеспечьте возможность запуска веб-браузера непосредственно на страницу, которая отображает соответствующий график. Подсказка: см. модуль `webbrowser`.

- 13.14. *Исторические данные.* Оказывается, ichart.finance.yahoo.com также обеспечивает "исторические" поисковые операции. Воспользуйтесь этим образцом URL, чтобы уяснить, как действует этот механизм, и напишите приложение, которое запрашивает цены акций за прошлые периоды: <http://chart.yahoo.com/table.csv?s=GOOG&a=06&b=12&c=2006&d=10&e=2&f=2007>.

Twitter

- 13.15.** *Служба Twitter.* Опишите службу Twitter своими словами. Сформулируйте определение твита и укажите некоторые из его ограничений.
- 13.16.** *Библиотека Twitter.* Обсудите сходства и различия между Python-библиотеками Twython и Tweepy.
- 13.17.** *Библиотеки Twitter.* Среди библиотек Python есть и другие, которые вы могли бы использовать для доступа к API Twitter. В чем их достоинства и недостатки по сравнению с библиотеками Python, которые рассматривались в этой главе?
- 13.18.** *Библиотеки Twitter.* Допустим, вам не нравятся Python-библиотеки Twython и Tweepy. Напишите собственный вариант библиотеки Python, которая обеспечивала бы безопасное и надежное взаимодействие с Twitter в стиле REST. Для начала обратитесь на сайт <https://dev.twitter.com/docs>.

Приведенные ниже упражнения требуют от вас дополнить и усовершенствовать пример `twapi.py` из этой главы.

- 13.19.** *Пользовательские запросы.* Нарастите функциональные возможности таким образом, чтобы можно было запрашивать экранное имя пользователя в Twitter и возвращать соответствующий ID этого пользователя. Обратите внимание: экранные имена некоторых пользователей, по сути, представляют собой целочисленные значения; позаботьтесь о том, чтобы ваши пользователи имели возможность вводить такие числа как потенциальные экранные имена. Воспользуйтесь таким ID для получения самого последнего твита соответствующего пользователя.
- 13.20.** *Выполнение ретвита.* Дополните поисковые возможности таким образом, чтобы пользователи могли не только запрашивать твиты, но и выполнять ретвит избранных твитов. Для поддержки этой возможности можете применить либо командную строку, либо веб-, либо GUI-интерфейс.
- 13.21.** *Удаление твитов.* Подобно упражнению 13.20, предоставьте возможность пользователю удалять его собственные “посты”. Учтите, что в этом случае соответствующие твиты будут удаляться лишь из Twitter. Содержимое этих твитов, возможно, уже разошлось по Интернету.
- 13.22.** *Отслеживание.* Добавьте функцию, которая позволяла бы вам отыскивать последователей того или иного пользователя (их ID), а также ID пользователей, которых отслеживает какой-то определенный пользователь.
- 13.23.** *Библиотеки Twitter.* Добавьте в `twapi.py` поддержку какой-то другой клиентской библиотеки Python/Twitter. Например, попытайтесь выполнить это упражнение с помощью `python-twitter`, обратившись на сайт <http://code.google.com/p/python-twitter>. Другие библиотеки можно найти на сайте <http://dev.twitter.com/docs/twitter-libraries#python>.
- 13.24.** *Редактирование профиля.* Создайте такую конфигурацию, чтобы пользователь мог обновлять свой профиль и/или выгружать картинку нового профиля. Плюс один балл: предоставьте возможность пользователям обновлять цвета или фоновое изображение своего профиля.

13.25. Подсчет. Функция интерфейса `API Twitter user_timeline()` также поддерживает переменную `count`. По умолчанию сеть Twitter возвращает во временной последовательности пользователя 20 самых последних твитов, но пользователи могут запросить до 200 твитов. Добавьте в `twapi.py` поддержку переменной `count` и других дополнительных параметров.

13.26. Непосредственные сообщения. Обеспечьте поддержку непосредственных сообщений (`direct messages — DM`), отправляя их какому-то определенному пользователю, получая список отправленных DM, формируя список текущих DM и удаляя DM.

В нашем примере `twapi.py` нам удалось проанализировать и модифицировать наш Twitter-поток, поскольку у нас были все разрешения, которые требовал Twitter от нашего приложения. Однако возникает иная ситуация, когда вы хотите написать приложение, которое отправляет твиты от вашего имени или ваших зарегистрированных пользователей. В этих случаях вам необходимо пройти через весь поток OAuth, чтобы получить маркер доступа и секрет, чтобы ваше приложение могло действовать от имени соответствующего пользователя.

Для выполнения этого последнего упражнения понадобится немало времени, поскольку вам нужно будет узнать все об OAuth. Начните с прочтения следующих двух документов: <https://dev.twitter.com/docs/auth/oauth> и <https://dev.twitter.com/docs/auth/moving-from-basic-auth-to-oauth>.

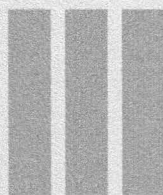
13.27. Архив твитов. Создайте архивную службу сети Twitter для себя (или других пользователей). Поскольку Twitter хранит не более 200 самых последних твитов, вы довольно быстро начинаете терять историю своих твитов. Создайте архивную службу сети Twitter, которая сохраняет твиты зарегистрированных пользователей. Если выполнить поиск в веб по ключевым словам “twitter archive” или “twitter research tools”, то можно узнать принципы построения архивов в сети Twitter. Надеюсь, что, выполнив это упражнение, вы сможете создать следующее поколение инструментов анализа Twitter!

13.28. Создание коротких ссылок и опрос “поступлений”. Усовершенствуйте программу обработки сообщений, поступающих в блог. Разработайте программу периодического сканирования (RSS или другую) для вашего личного или рабочего блога. Когда в блоге появляется новое сообщение, выдавайте автоматический твит в виде короткой ссылки и первые N слов из заголовка этого сообщения.

13.29. Другие веб-службы. Прочитайте об интерфейсе `API Prediction (Google)` на сайте <http://code.google.com/apis/predict> и поработайте с его учебником “HelloWorld”. Ознакомившись с этими материалами, разработайте свою собственную модель, которая сканирует разные твиты (ваши собственные или твиты из публичной последовательности). Разработайте и обучите модель прогнозирования, которая определяет характер содержимого того или иного твита (позитивный, негативный или нейтральный). Завершив обучение, воспользуйтесь этим инструментом для определения “настроений” новых твитов, используя один и тот же запрос. Для того чтобы выполнить

это упражнение, вам понадобится создать проект на консоли API Google (<http://code.google.com/apis/console>) и задействовать как интерфейс Google Prediction, так и интерфейс Google Storage. Если не хотите создавать учетную запись на сервере Google, воспользуйтесь любым аналогичным интерфейсом API.

ЧАСТЬ



**Дополнительная
и экспериментальная**

Обработка текста

В этой главе...

- Значения, разделяемые запятыми
- Нотация объектов JavaScript
- Расширяемый язык разметки гипертекста
- Литература
- Модули, связанные с обработкой текстов

*Как разработчик я предпочитаю редактировать
обычный текст. XML не в счет.
Уэсли Чан, июль 2009 г.
(высказывание на конференции OSCON)*

Какого бы типа приложения вы ни создавали, вам неизбежно понадобится обрабатывать данные, которые способен воспринимать человек; такие данные обычно называют *текстом*. Стандартная библиотека Python содержит три модуля обработки текста и пакеты, помогающие вам решить эту задачу: `csv`, `json` и `xml`. В данной главе мы кратко рассмотрим каждый из этих пакетов в указанной последовательности.

В конце мы объединим XML с познаниями в области клиент-серверных технологий, приобретенными вами в главе 2, и покажем, как с помощью Python создавать службы, поддерживающие протокол XML-RPC. Поскольку этот стиль программирования не считается обработкой текстов, а самим XML (который представляет собой лишь формат транспортировки данных) вы не манипулируете, этот последний раздел можно рассматривать лишь как дополнительный полезный материал.

14.1. Значения, разделяемые запятыми

В первом разделе этой главы мы рассмотрим значения, разделяемые запятыми (comma-separated values — CSV). Рассмотрение значений, разделяемых запятыми, мы начнем с краткого введения, затем перейдем к изучению того, как с помощью Python читать и записывать CSV-файлы, воспользовавшись для этого соответствующим примером кода. Наконец, мы наведем одного своего старого друга.

14.1.1. Введение в значения, разделяемые запятыми

Использование CSV является распространенным способом перемещения данных в приложения типа электронных таблиц и из таких приложений с использованием обычного текста (в отличие от специализированного формата двоичных файлов). Вообще говоря, CSV даже не является истинно структурированными данными: содержимое файлов CSV представляет собой лишь ряды строковых значений, разделенных запятыми. Форматам CSV присущи определенные тонкости, но в целом они весьма несущественны. Во многих случаях вам, по сути, вовсе не нужны богатые возможности CSV-ориентированного модуля.

На первый взгляд, синтаксический анализ в этом случае должен оказаться весьма простым. Достаточно было бы, к примеру, регулярно выполнять `str.split(',')`. Однако мы не можем поступить так, поскольку значения в отдельных полях могут содержать встроенные запятые, а это обуславливало бы необходимость синтаксического анализа CSV и генерирования библиотеки, подобной CSV-модулю Python.

Рассмотрим короткий пример получения данных, записи CSV в какой-либо файл и последующего считывания тех же данных. Кроме того, отдельные поля у нас будут включать запятые (это сделано специально для того, чтобы несколько усложнить ситуацию). Пример 14.1 представляет `csvex.py`, сценарий, который принимает тройные кавычки и фиксирует каждую соответствующую запись на диске как CSV-файл. Затем выполняется считывание и синтаксический анализ этих записанных на диск CSV-данных.

Пример 14.1. Демонстрация модуля CSV, совместимого с версиями Python 2 и Python 3 (csvex.py)

Этот простой сценарий демонстрирует запись на диск CSV-данных и последующее их считывание.

```

1  #!/usr/bin/env python
2
3  import csv
4  from distutils.log import warn as printf
5
6  DATA = (
7      9, 'Web Clients and Servers', 'base64, urllib'),
8      (10, 'Web Programming: CGI & WSGI', 'cgi, time, wsgiref'),
9      (13, 'Web Services', 'urllib, twython'),
10 )
11
12 printf('*** WRITING CSV DATA')
13 f = open('bookdata.csv', 'w')
14 writer = csv.writer(f)
15 for record in DATA:
16     writer.writerow(record)
17 f.close()
18
19 printf('*** REVIEW OF SAVED DATA')
20 f = open('bookdata.csv', 'r')
21 reader = csv.reader(f)
22 for chap, title, modpkgs in reader:
23     printf('Chapter %s: %r (featuring %s)' % (
24         chap, title, modpkgs))
25 f.close()

```

Ниже приведен еще один пример сценариев записи, которые совместимы как с Python 2, так и с Python 3. Какую бы из версий Python вы ни использовали, вы получите одинаковый результат, представленный ниже.

```

$ python csvex.py
*** WRITING CSV DATA
*** REVIEW OF SAVED DATA
Chapter 9: 'Web Clients and Servers' (featuring base64, urllib)
Chapter 10: 'Web Programming: CGI & WSGI' (featuring cgi, time, wsgiref)
Chapter 13: 'Web Services' (featuring urllib, twython)

```

Построчное объяснение

Строки 1–10

Сначала мы импортируем модуль `csv`, а также `distutils.log.warn()` как модуль доступа для оператора или функции `print`. (В действительности совместимость здесь не обеспечивается, за исключением отдельно взятой строки, но этот фрагмент кода выполняет поставленную задачу при условии, что вы можете работать с его ограничением.) Вслед за операторами импорта расположен наш набор данных. Он состоит из тройных кортежей, столбцы которых представляют номера и названия глав, а также модули и пакеты, которые используются в примерах кода из соответствующих глав.

Строки 12–17

Эти шесть строк кода в особых пояснениях не нуждаются. `csv.writer()` является функцией, которая получает на входе объект типа “открытый файл” (или файлоподобный объект) и возвращает объект, выполняющий запись (устройство записи). Устройство записи использует метод `writerow()`, который позволяет вам записывать в определенный файл строки данных, разделяемых запятыми. После выполнения записи этот файл закрывается.

Строки 19–25

В этом фрагменте кода `csv.reader()` является функцией, противоположной `csv.writer()`; эта функция возвращает итерируемый объект, который вы можете использовать для считывания и выполнения синтаксического анализа каждой строки CSV-данных. Подобно `csv.writer()`, функция `csv.reader()` также получает на входе объект типа “открытый файл” и возвращает объект, выполняющий чтение (устройство чтения). Когда вы просматриваете поочередно каждую строку данных, автоматически выполняется синтаксический анализ этих CSV-данных, после чего они возвращаются вам (строка 22). Выводимая информация отображается, а когда все строки будут обработаны, файл закрывается.

Помимо функций `csv.reader()` и `csv.writer()`, модуль `csv` содержит также классы `csv.DictReader` и `csv.DictWriter`, которые считывают CSV-данные в словарь (с заданными именами полей или, в противном случае, первой строкой) и записывают поля словаря в CSV-файл.

14.1.2. Повторение примера с портфелем акций

Прежде чем перейти к другому формату обработки текста, рассмотрим еще один пример. Вернемся к нашему сценарию портфеля акций, `stock.py`, описанному в главе 13. Вместо того чтобы выполнить `str.split(',')`, мы преобразуем это приложение таким образом, чтобы оно использовало модуль `csv`.

Кроме того, вместо того чтобы показывать вам *весь* соответствующий код, большая часть которого идентична сценарию `stock.py`, мы сосредоточимся лишь на различиях. Ниже приведен краткий обзор всего сценария `stock.py` (Python 2). (Подробное построчное разъяснение этого сценария см. в главе 13.)

```
#!/usr/bin/env python

from time import ctime
from urllib2 import urlopen
TICKS = ('yhoo', 'dell', 'cost', 'adbe', 'intc')
URL = 'http://quote.yahoo.com/d/quotes.csv?s=%s&f=sllc1p2'

print '\nPrices quoted as of: %s PDT\n' % ctime()
print 'TICKER', 'PRICE', 'CHANGE', '%AGE'
print '-----', '-----', '-----', '-----'
u = urlopen(URL % ','.join(TICKS))

for row in u:
    tick, price, chg, per = row.split(',')
    print tick, '%.2f' % float(price), chg, per,

u.close()
```

Выходной результат как оригинальной версии, так и нашей модифицированной версии будет одинаковым. В качестве напоминания приводим пример результатов, полученных после одного выполнения сценария `stock.py`:

Prices quoted as of: Sat Oct 29 02:06:24 2011 PDT

Тикер	Цена в долларах	Изменение	Изменение в процентах
"YHOO"	16,56	-0,07	"-0,42%"
"DELL"	16,31	-0,01	"-0,06%"
"COST"	84,93	-0,29	"-0,34%"
"ADBE"	29,02	+0,68	"+2,40%"
"INTC"	24,98	-0,15	"-0,60%"

Все, что мы собираемся сделать, — это скопировать код из файла `stock.py` в новый сценарий `stockcsv.py` и внести изменения, необходимые для использования модуля `csv`. Проанализируем различия, сосредоточившись на коде, который следует после обращения к `urlopen()`. Как только мы получим этот открытый файл, мы назначим его `csv.reader()`, как показано ниже.

```
reader = csv.reader(u)
for tick, price, chg, pct in reader:
    print tick.ljust(7), ('%.2f' % round(float(price), 2)).rjust(6), \
          chg.rjust(6), pct.rstrip().rjust(6)
u.close()
```

Цикл `for` в основном остался таким же, как был, за исключением того, что сейчас мы не считываем всю строку и разделяем ее по запятой. Вместо этого модуль `csv` выполняет за нас естественным образом синтаксический анализ и предоставляет возможность пользователям указать имена целевых полей как переменных цикла. Обратите внимание: результат на выходе подобен прежнему, но не полностью совпадает с ним. Можете ли вы указать разницу между ними (разумеется, помимо отметки времени)? Взгляните:

Prices quoted as of: Sun Oct 30 23:19:04 2011 PDT

Тикер	Цена в долларах	Изменение	Изменение в %
YHOO	16,56	-0,07	-0,42%
DELL	16,31	-0,01	-0,06%
COST	84,93	-0,29	-0,34%
ADBE	29,02	+0,68	+2,40%
INTC	24,98	-0,15	-0,60%

Разница, на первый взгляд, несущественная. Некоторые поля в версии `str.split()` взяты в кавычки, тогда как в версии, обработанной модулем `csv`, этих кавычек нет. Чем это объясняется? Вспомните из главы 13, что некоторые значения возвращаются в кавычках и что в конце главы 13 помещено упражнение, где вы должны вручную удалить лишние кавычки.

Здесь это не является проблемой, поскольку модуль `csv` помогает нам обрабатывать CSV-данные, в том числе находить и убирать лишние кавычки, которые приходят

нам с сервера Yahoo!. Ниже приведен фрагмент кода и выходная информация в подтверждение этих лишних кавычек.

```
>>> from urllib2 import urlopen
>>> URL = 'http://quote.yahoo.com/d/quotes.csv?s=goog&f=sllc1p2'
>>> u = urlopen(URL, 'r')
>>> line = u.read()
>>> u.close()
>>> line
'"GOOG",598.67,+12.36,"+2.11%"\r\n'
```

Эти кавычки являются для разработчиков лишней головной болью, от которой желательно их избавить; модуль `csv` берет на себя решение указанной проблемы, несколько облегчая таким образом чтение кода (в этом случае от разработчиков не требуется создавать код для дополнительной обработки строк).

Для того чтобы еще больше улучшить управление данными, желательно, чтобы данные были организованы в более иерархическую структуру. Было бы, например, неплохо, если бы каждая возвращаемая строка была частью какого-то одного объекта, причем атрибутами этого объекта были бы цена, абсолютное и процентное изменение цены. В случае CSV-строки с четырьмя значениями невозможно сказать, какое из них является первичным ключом, если в качестве такового вы не используете первое значение или если не используется какое-либо другое, аналогичное соглашение. Именно в этом случае JSON может оказаться более подходящим инструментом для ваших приложений.

14.2. Нотация объектов JavaScript

Как нетрудно догадаться по названию, нотация объектов JavaScript (JavaScript Object Notation — JSON) является принадлежностью языка JavaScript. Речь идет о подмножестве этого языка, используемом для обработки структурированных данных. Нотация объектов JavaScript базируется на стандарте ECMA-262 и рассматривается как упрощенная альтернатива обмена данными расширяемому языку разметки гипертекста (Extensible Markup Language — XML), который будет рассматриваться в последнем разделе этой главы. Формат JSON считается более читабельным способом транспортировки структурированных данных. Подробнее о формате JSON можно прочитать на сайте <http://json.org>.

2.5-2.6

Поддержка JSON была официально добавлена в стандартную библиотеку Python 2.6 посредством модуля `json`. По сути, он является интегрированной в настоящее время версией внешней библиотеки `simplejson`, разработчики которой обеспечили обратную совместимость с версией 2.5. Более подробную информацию можно получить на сайте <http://github.com/simplejson/simplejson>.

Кроме того, модуль `json` (и, следовательно, `simplejson`) обеспечивает интерфейс, подобный интерфейсам в модулях `pickle` и `marshal`, т.е. функции `dump()` / `load()` и `dumps()` / `loads()`. Помимо базовых параметров, эти функции включают также всевозможные параметры, относящиеся лишь к формату JSON. Этот модуль включает также классы кодировщика и декодировщика, из которых вы можете выводить и которые можете использовать напрямую.

Объект *JSON* очень похож на словарь *Python*, подтверждением чего могут служить приведенные ниже фрагменты кода, в которых мы используем *dict* для переноса данных в объект *JSON*, а затем обратно.

```
>>> dict(zip('abcde', range(5)))
{'a': 0, 'c': 2, 'b': 1, 'e': 4, 'd': 3}
>>> json.dumps(dict(zip('abcde', range(5))))
'{"a": 0, "c": 2, "b": 1, "e": 4, "d": 3}'
>>> json.loads(json.dumps(dict(zip('abcde', range(5)))))
{'a': 0, 'c': 2, 'b': 1, 'e': 4, 'd': 3}
```

Обратите внимание: объект *JSON* понимает только строки Юникода, поэтому при выполнении обратной трансляции в код *Python* последний из предыдущих примеров (все *Python 2*) преобразует ключи в строки Юникода. Выполнение точно такой же строки кода в *Python 3* оказывается более нормальным без оператора строки Юникода (обозначение *u* в нем предшествует открывающей кавычке):

```
>>> json.loads(json.dumps(dict(zip('abcde', range(5)))))
{'a': 0, 'c': 2, 'b': 1, 'e': 4, 'd': 3}
```

Словари *Python* (*dict*) преобразуются в объекты *JSON*. Аналогично списки (*list*) или кортежи (*tuple*) *Python* считаются *массивами JSON*:

```
>>> list('abcde')
['a', 'b', 'c', 'd', 'e']
>>> json.dumps(list('abcde'))
'["a", "b", "c", "d", "e"]'
>>> json.loads(json.dumps(list('abcde')))
[u'a', u'b', u'c', u'd', u'e']
>>> # ['a', 'b', 'c', 'd', 'e'] in Python 3
>>> json.loads(json.dumps(range(5)))
[0, 1, 2, 3, 4]
```

Какие еще существуют различия между типами данных и значениями *Python* и *JSON*? Некоторые из ключевых различий перечислены в табл. 14.1.

Таблица 14.1. Различия между типами *JSON* и *Python*

JSON	Python 2	Python 3
object	dict	dict
array	list, tuple	list, tuple
string	Unicode	str
number (int)	int, long	int
number (real)	float	float
True	True	True
false	False	False
null	None	None

Еще одним малозаметным различием, не представленным в табл. 14.1, является то, что *JSON* не использует одиночные кавычки/апострофы; каждая строка ограничивается путем использования двойных кавычек. Кроме того, не используются дополнительные конечные запятые, которые *Python*-программисты иногда ставят для удобства в конце каждой последовательности или отображающего элемента.

Для того чтобы читатели могли более отчетливо уяснить некоторые из этих различий, пример 14.2 представляет `dict2json.py`. Это совместимый с Python 2 и Python 3 сценарий, который выполняет дамп содержимого словаря четырьмя способами: дважды как словаря Python (`dict`) и дважды как объекта JSON.

Пример 14.2. Преобразование словаря Python (`dict`) в JSON (`dict2json.py`)

Этот сценарий преобразует словарь Python в JSON и отображает его в разных форматах.

```

1  #!/usr/bin/env python
2
3  from distutils.log import warn as printf
4  from json import dumps
5  from pprint import pprint
6
7  BOOKS = {
8      '0132269937': {
9          'title': 'Core Python Programming',
10         'edition': 2,
11         'year': 2007,
12     },
13     '0132356139': {
14         'title': 'Python Web Development with Django',
15         'authors': ['Jeff Forcier', 'Paul Bissex', 'Wesley Chun'],
16         'year': 2009,
17     },
18     '0137143419': {
19         'title': 'Python Fundamentals',
20         'year': 2009,
21     },
22 }
23
24 printf('*** RAW DICT ***')
25 printf(BOOKS)
26
27 printf('\n*** PRETTY_PRINTED DICT ***')
28 pprint(BOOKS)
29
30 printf('\n*** RAW JSON ***')
31 printf(dumps(BOOKS))
32
33 printf('\n*** PRETTY_PRINTED JSON ***')
34 printf(dumps(BOOKS, indent=4))

```

Построчное объяснение

Строки 1–5

Мы импортируем три функции, которые будут использованы в этом сценарии: 1) `distutils.log.warn()` как замену оператора `print` в Python 2 и функции `print()` в языке Python 3; 2) `json.dumps()`, чтобы возвращать строковое представление в формате JSON того или иного объекта Python; и 3) `pprint.pprint()`, которая выполняет простую структурную распечатку объектов Python.

Строки 7–22

Структура данных BOOKs является словарем Python, представляющим книги, идентифицируемые по их номерам в системе стандартных международных номеров книг (International Standard Book Number — ISBN). Каждая книга может характеризоваться дополнительной информацией, такой как название, автор, год издания и т.п. Вместо того чтобы использовать более простую структуру данных (например, список), мы выбрали dict, поскольку он позволяет выстроить структурированную иерархию атрибутов. Обратите внимание на все лишние запятые, которые будут удалены в эквивалентном JSON-представлении.

Строки 24–34

Оставшиеся строки этого сценария выполняют весь вывод информации. Первая выполняет дамп словаря Python; каких-либо дополнительных разъяснений в данном случае не требуется. Обратите внимание: лишние запятые здесь также удаляются. В исходном коде мы используем их главным образом для удобства чтения. Второй пример является тем же самым словарем Python, но с точки зрения программы структурной распечатки.

Два последних вывода осуществляются в формате JSON. Первый из них представляет собой простой JSON-дамп после преобразования, второй — дополнительную функцию структурной распечатки, встроенную в json.dumps(). Для того чтобы задействовать эту функцию, вам необходимо лишь передать в нее уровень структурирования выводимой информации.

Выполнение этого сценария в версиях Python 2 либо Python 3 позволяет получить на выходе следующую информацию:

```
$ python dict2json.py
*** RAW DICT ***
{'0132269937': {'edition': 2, 'year': 2007, 'title': 'Core Python
Programming'}, '0137143419': {'year': 2009, 'title': 'Python
Fundamentals'}, '0132356139': {'authors': ['Jeff Forcier',
'Paul Bissex', 'Wesley Chun'], 'year': 2009, 'title': 'Python
Web Development with Django'}}

*** PRETTY_PRINTED DICT ***
{'0132269937': {'edition': 2,
    'title': 'Core Python Programming',
    'year': 2007},
'0132356139': {'authors': ['Jeff Forcier', 'Paul Bissex', 'Wesley
    Chun'],
    'title': 'Python Web Development with Django',
    'year': 2009},
'0137143419': {'title': 'Python Fundamentals', 'year': 2009}}

*** RAW JSON ***
{"0132269937": {"edition": 2, "year": 2007, "title": "Core Python
Programming"}, "0137143419": {"year": 2009, "title": "Python
Fundamentals"}, "0132356139": {"authors": ["Jeff Forcier",
"Paul Bissex", "Wesley Chun"], "year": 2009, "title": "Python
Web Development with Django"}}

*** PRETTY_PRINTED JSON ***
{
  "0132269937": {
```

```
"edition": 2,  
"year": 2007,  
"title": "Core Python Programming"  
},  
"0137143419": {  
    "year": 2009,  
    "title": "Python Fundamentals"  
},  
"0132356139": {  
    "authors": [  
        "Jeff Forcier",  
        "Paul Bissex",  
        "Wesley Chun"  
    ],  
    "year": 2009,  
    "title": "Python Web Development with Django"  
}  
}
```

Этот пример демонстрирует переход от словаря `dicts` к представлению в формате JSON. Вы можете также перемещать данные между `lists` или кортежами и JSON-массивами. Модуль `json` также предоставляет классы для кодирования и декодирования других типов данных Python в JSON и из JSON. Несмотря на то что здесь у нас нет возможности обсудить все эти типы данных, нетрудно заметить, что поле деятельности для исследований с помощью формата JSON довольно обширное и, безусловно, не ограничивается лишь представленным здесь кратким введением.

Сейчас мы приступаем к рассмотрению такого внушительного средства форматирования текста, как XML.

14.3. Расширяемый язык разметки гипертекста

Третьей темой при рассмотрении обработки данных, которую мы обсуждаем в этой главе, является расширяемый язык разметки гипертекста (Extensible Markup Language — XML). Подобно приведенному выше обсуждению CSV, мы сделаем небольшое вступление, после чего рассмотрим способы обработки данных XML с помощью Python. После рассмотрения краткого примера соответствующего кода мы выполним синтаксический анализ реальных данных, поступающих от службы Google News.

14.3.1. Введение в язык XML

В последнем разделе этой главы мы рассмотрим XML, более старый формат структурированных данных, который, по утверждению его разработчиков, является обычным текстовым форматом, используемым для представления структурированных данных. Несмотря на то что данные XML представляют собой обычный текст, многие утверждают, что XML является нечитабельным форматом, и это утверждение не лишено оснований. Такой формат может быть практически нечитабельным, и без помощи синтаксического анализатора здесь не обойтись. Однако формат XML используется уже достаточно давно и до сих пор имеет более широкое распространение, чем формат JSON. В наши дни практически в каждом языке программирования предусмотрены синтаксические анализаторы XML.

Язык XML представляет собой ограниченную форму стандартного обобщенного языка разметки гипертекста (Standard Generalized Markup Language — SGML), который сам по себе является одним из стандартов ISO (ISO 8879). Он появился в 1996 г., когда консорциум World Wide Web (W3C) создал рабочую группу для разработки этого языка. Первая спецификация XML была опубликована в 1998 г.; самое последнее обновление было выпущено в 2008 г. Язык XML можно представлять себе как подмножество SGML. Язык HTML можно также рассматривать как даже еще меньшее подмножество SGML.

14.3.2. Языки Python и XML

Первоначально поддержка XML в языке Python была обеспечена сразу же после появления версии 1.5 и модуля `xmllib`. С тех пор модуль `xmllib` превратился в пакет `xml`, обеспечивающий множество способов выполнения синтаксического анализа, а также конструирования XML-документов.

2.0

Язык Python поддерживает структурированную в виде дерева объектную модель документов (Document Object Model — DOM), а также базирующийся на событиях обработки XML-документов интерфейс API Simple для языка XML (SAX). Текущей версией спецификации SAX является 2.0.1, поэтому, когда речь идет о поддержке в языке Python, эту спецификацию, вообще говоря, принято обозначать как SAX2. Стандарт DOM является достаточно старым и находится в обращении примерно столько же, сколько сам XML. Поддержка SAX и DOM была добавлена в Python в версии 2.0.

2.3

SAX — это потоковый интерфейс; это означает, что синтаксический анализ и обработка документов выполняются построчно посредством непрерывного потока байтов. Это означает, что в рамках какого-либо XML-документа вы не можете ни выполнять просмотр в обратном порядке, ни выполнять произвольный доступ к данным. Нетрудно предположить, что возможным компромиссом в данном случае являются процессоры, базирующиеся на событиях; такие процессоры работают быстрее и более эффективно используют память, тогда как синтаксические анализаторы, базирующиеся на деревьях, в любой момент обеспечивают вам полный доступ ко всему документу в памяти.

Обратите внимание на то, что пакет `xml` зависит от наличия по крайней мере одного SAX-совместимого синтаксического анализатора XML. В то время это означало, что пользователям нужно было найти и загрузить модули или пакеты сторонних разработчиков, которые помогли бы им выполнить это требование. К счастью, в версии 2.3 потоковый синтаксический анализатор Expat был включен в стандартную библиотеку под именем `xml.parsers.expat`.

Анализатор Expat появился до SAX и несовместим с ним. Однако вы можете использовать Expat для создания синтаксических анализаторов SAX или DOM. Обратите внимание и на то, что главным преимуществом Expat является быстродействие. Его высокое быстродействие объясняется тем, что он *не выполняет проверку достоверности*, т.е. он не выполняет проверку на полную совместимость разметки. Как нетрудно догадаться, синтаксические анализаторы, *выполняющие проверку достоверности*, работают медленнее, из-за того что выполняют дополнительную обработку.

2.5

Поддержка XML в версии Python 2.5 стала более всесторонней в результате добавления `ElementTree` — чрезвычайно популярного, быстродействующего синтаксического анализатора и генератора документов XML, — включенного в стандартную библиотеку под именем `xml.etree.ElementTree`. Мы будем использовать `ElementTree` для всех наших сырых примеров XML (с привлечением некоторой помощи от `xml.dom.minidom`), а затем продемонстрируем несколько примеров записи клиент-серверных приложений с использованием поддержки XML-RPC, предусмотренной в Python.

В примере 14.3 (`dict2xml.py`) мы берем структурированные данные в словаре Python, используем `ElementTree` для формирования допустимого документа XML, представляющего такую структуру данных, используем `xml.dom.minidom` для структурной распечатки этих данных и, наконец, используем разные итераторы `ElementTree` для выполнения синтаксического анализа и отображения в этих данных контента, представляющего для нас интерес.

Пример 14.3. Преобразование dict Python в XML (`dict2xml.py`)

Этот сценарий на языке Python 2 преобразует dict в XML и отображает его в нескольких форматах.

```

1  #!/usr/bin/env python
2
3  from xml.etree.ElementTree import Element, SubElement, tostring
4  from xml.dom.minidom import parseString
5
6  BOOKS = {
7      '0132269937': {
8          'title': 'Core Python Programming',
9          'edition': 2,
10         'year': 2006,
11     },
12     '0132356139': {
13         'title': 'Python Web Development with Django',
14         'authors': 'Jeff Forcier:Paul Bissex:Wesley Chun',
15         'year': 2009,
16     },
17     '0137143419': {
18         'title': 'Python Fundamentals',
19         'year': 2009,
20     },
21 }
22
23 books = Element('books')
24 for isbn, info in BOOKS.iteritems():
25     book = SubElement(books, 'book')
26     info.setdefault('authors', 'Wesley Chun')
27     info.setdefault('edition', 1)
28     for key, val in info.iteritems():
29         SubElement(book, key).text = ', '.join(str(val) .split(':'))
30
31 xml = tostring(books)
32 print '*** RAW XML ***'
33 print xml
34
```

```

35 print '\n*** PRETTY-PRINTED XML ***'
36 dom = parseString(xml)
37 print dom.toprettyxml(' ')
38
39 print '*** FLAT STRUCTURE ***'
40 for elmt in books.getiterator():
41     print elmt.tag, '-', elmt.text
42
43 print '\n*** TITLES ONLY ***'
44 for book in books.findall('.//title'):
45     print book.text

```

Выполнение этого сценария, который легко преобразовать для версии Python 3, приводит к следующим результатам:

```

$ dict2xml.py
*** RAW XML ***
<books><book><edition>2</edition><authors>Wesley Chun</
authors><year>2006</year><title>Core Python Programming</
book><book><edition>1</edition><authors>Wesley Chun</
authors><year>2009</year><title>Python Fundamentals</title></
book><book><edition>1</edition><authors>Jeff Forcier, Paul Bissex,
Wesley Chun</authors><year>2009</year><title>Python Web Development
with Django</title></book></books>

*** PRETTY-PRINTED XML ***
<?xml version="1.0" ?>
<books>
  <book>
    <edition>
      2
    </edition>
    <authors>
      Wesley Chun
    </authors>
    <year>
      2006
    </year>
    <title>
      Core Python Programming
    </title>
  </book>
  <book>
    <edition>
      1
    </edition>
    <authors>
      Wesley Chun
    </authors>
    <year>
      2009
    </year>
    <title>
      Python Fundamentals
    </title>
  </book>
</books>

```



```

<edition>
  1
</edition>
<authors>
  Jeff Forcier, Paul Bissex, Wesley Chun
</authors>
<year>
  2009
</year>
<title>
  Python Web Development with Django
</title>
</book>
</books>

```

```

*** FLAT STRUCTURE ***
books - None
book - None
edition - 2
authors - Wesley Chun
year - 2006
title - Core Python Programming
book - None
edition - 1
authors - Wesley Chun
year - 2009
title - Python Fundamentals
book - None
edition - 1
authors - Jeff Forcier, Paul Bissex, Wesley Chun
year - 2009
title - Python Web Development with Django

```

```

*** TITLES ONLY ***
Core Python Programming
Python Fundamentals
Python Web Development with Django

```

Построчное объяснение

Строки 1–21

Первая половина этого сценария очень похожа на первую половину сценария `dict2json.py`, который мы обсуждали в предыдущем разделе. К числу очевидных изменений относится импортирование `ElementTree` и `minidom`. Мы исходим из того, что вам известно, что именно нужно сделать, чтобы ваш код был пригоден как для версии Python 2, так и для Python 3, поэтому не будем отвлекаться на эти тонкости и сосредоточимся исключительно на решении, касающемся версии Python 2.

Наконец, самое малозаметное различие заключается в том, что поле `'authors'` представляет собой не список, как это было в `dict2json.py`, а строку, ограниченную двоеточием (:). Однако это изменение является необязательным, а поле `'authors'` может по-прежнему оставаться списком, если вы предпочитаете именно такой вариант.

Причиной внесения такого изменения является стремление упростить обработку данных. Одним из ключевых мест, где это можно сделать, очевидно, является

строка 29. Еще одно различие заключается в том, что в примере JSON мы не задали фамилию автора по умолчанию, если она не указана (в данном случае мы указали фамилию автора). Проще выполнять проверку на наличие двоеточия и не выполнять дополнительную проверку, если значением наших данных является строка или список.

Строки 23–29

Именно в этих строках выполняется реальная работа данного сценария. Мы создаем объект верхнего уровня, `books`, а затем под этим узлом присоединяем все остальное. Для каждой книги добавляется вложенный узел `book`, принимающий значения `authors` и `edition`, предусмотренные по умолчанию, если они не заданы в явном виде выше в исходном определении словаря. Это сопровождается итерацией по всем парам ключевых значений и добавлением их как дополнительных вложенных узлов каждой книги.

Строки 31–45

Этот последний блок кода распечатывает наши данные в разных форматах: сырой XML, структурированная распечатка XML (с помощью MiniDOM), итерация по всем узлам как одной крупной простой структуры и, наконец, демонстрация просмотра всего документа XML.

14.3.3. Применение формата XML на практике

В то время как предыдущий пример показывает разные приемы, которыми вы можете пользоваться для создания и синтаксического анализа документов XML, не приходится сомневаться в том, что большинство приложений пытаются воспользоваться именно последним вариантом, а не первым. Рассмотрим еще одно небольшое приложение, которое выполняет синтаксический анализ данных для получения полезной информации.

В примере 14.4 `goognewsrss.py` принимает сообщения “Top Stories” от службы Google News и извлекает заголовки пяти (по умолчанию) важнейших новостей, а также ссылки на публикации с развернутым изложением соответствующих новостей. Данное решение, `goognewsrss.topnews()`, представляет собой генератор, легко идентифицируемый выражением `yield`. Это означает, что индивидуальные пары (заголовков, ссылка) вырабатываются данным генератором итеративным образом. Взгляните на этот код и оцените, можете ли вы понять, что здесь происходит и каким может оказаться выходной результат (мы намеренно не приводим распечатку результатов работы этого приложения по причине, которая будет разъяснена ниже).

Пример 14.4. Выполнение синтаксического анализа фактического XML (`goognewsrss.py`)

Этот сценарий, совместимый с Python 2 и Python 3, отображает важнейшие новости (по умолчанию отображается пять таких новостей) службы Google News, а также ссылки на публикации с развернутым изложением соответствующих новостей.

```
1  #!/usr/bin/env python
2
3  try:
4      from io import BytesIO as StringIO
5  except ImportError:
6      try:
```

```
7         from cStringIO import StringIO
8     except ImportError:
9         from StringIO import StringIO
10
11     try:
12         from itertools import izip as zip
13     except ImportError:
14         pass
15
16     try:
17         from urllib2 import urlopen
18     except ImportError:
19         from urllib.request import urlopen
20
21     from pprint import pprint
22     from xml.etree import ElementTree
23
24     g = urlopen('http://news.google.com/news?topic=h&output=rss')
25     f = StringIO(g.read())
26     g.close()
27     tree = ElementTree.parse(f)
28     f.close()
29
30     def topnews(count=5):
31         pair = [None, None]
32         for elmt in tree.getiterator():
33             if elmt.tag == 'title':
34                 skip = elmt.text.startswith('Top Stories')
35                 if skip:
36                     continue
37                 pair[0] = elmt.text
38             if elmt.tag == 'link':
39                 if skip:
40                     continue
41                 pair[1] = elmt.text
42                 if pair[0] and pair[1]:
43                     count -= 1
44                     yield(tuple(pair))
45                 if not count:
46                     return
47                 pair = [None, None]
48
49     for news in topnews():
50         pprint(news)
```

Прежде чем выполнить этот код, обязательно ознакомьтесь с условиями пользования службой (Terms of Service — ToS), которые можно найти на следующей странице: http://news.google.com/intl/en_us/terms_google_news.html. Здесь описаны условия, при которых вы можете пользоваться этой службой Google. Ключевой фразой здесь является следующая: “Вы можете отображать содержимое данной службы лишь для своего личного пользования (т.е. для некоммерческих целей) и не имеете права копировать, воспроизводить, вносить изменения, модифицировать, создавать производные работы или публично отображать любое содержимое данной службы”.

В данном случае это означает следующее: поскольку настоящая книга предназначена для широкого круга читателей, я не могу вставить в нее пример выполнения

рассматриваемого нами кода, равно как и не могу попытаться замаскировать фактический результат его выполнения, поскольку это означало бы модификацию содержимого, однако вы можете сделать это — для своего личного пользования.

В результате выполнения этого кода вы увидите совокупность из заголовков пяти важнейших новостей и соответствующих ссылок (в виде пар). Обратите внимание: поскольку в данном случае речь идет о динамической службе, содержимое которой непрерывно изменяется, повторное выполнение этого сценария в другое время наверняка принесет другие результаты.

Построчное объяснение

Строки 1–22

Да, мы отдаем себе отчет в том, что перфекционистам этот код покажется довольно безобразным из-за наличия в нем операций импортирования, которые затрудняют чтение кода, — и мне придется согласиться с такой оценкой. Однако на практике, когда у вас есть разные версии языка для выполнения производственного кода, особенно когда на сцене появляется версия Python 3, приходится иметь дело с операторами типа “`ifdef`” — и не только с ними. Давайте разделим их на составные части, чтобы вы могли по крайней мере увидеть, что же происходит на самом деле.

Нам понадобится буфер, в котором могла бы поместиться большая строка, с интерфейсом файла. Иными словами, это одна большая строка в памяти, которая поддерживает файловый интерфейс; т.е. она содержит файловые методы наподобие `write()`. Это был бы класс `StringIO`. Данные, которые поступают из сети, обычно имеют кодировку ASCII или являются чисто байтовыми (не Юникод). Поэтому, если мы работаем с версией Python 3, нам нужно использовать класс `io.BytesIO` как `StringIO`.

Если же мы используем версию Python 2, то Юникод не является частью этой картины, поэтому мы могли бы попытаться воспользоваться более быстродействующим, скомпилированным на языке C классом `cStringIO.StringIO`, если таковой имеется в нашем распоряжении. Если же его нет, то у нас в запасе есть исходный класс `StringIO.StringIO`.

Далее, нам желательно эффективно использовать оперативную память; следовательно, мы предпочли бы итераторную версию встроенной функции `zip()`, а именно `itertools.izip()`. Если в модуле `itertools` имеется `izip()`, то мы понимаем, что работаем с Python 2; следовательно, эту функцию нужно импортировать как `zip()`. В противном случае мы понимаем, что работаем с Python 3, поскольку `izip()` заменена и переименована в `zip()`; это означает, что нам следует просто игнорировать сообщение об ошибке `ImportError`, если искомая функция не будет найдена. Обратите внимание: в этом коде не используется ни `zip()`, ни `izip()`; подробнее об этом см. ниже, во врезке “Уголок хакера”.

Последний особый случай относится к модулю Python 2 `urllib2`, который объединен с несколькими другими и помещен во вложенный модуль Python 3 `urllib.request`. Тот из них, который будет возвращен, предоставляет нам для использования свою функцию `urlopen()`.

Наконец, мы будем использовать `ElementTree`, а также функцию `pprint.pprint()`, которая выполняет простую структурную распечатку. Выходная информация, вообще говоря, охватывается этим примером, поэтому мы предпочитаем такой вариант вывода результирующей информации как альтернативу `distutils.log.warn()`.

Строки 24–28

Сбор данных в этом приложении выполняется именно в этих строках кода. Мы начинаем с открытия соединения с сервером Google News и запроса вывода RSS, который представлен в формате XML. Мы считываем всю входную информацию и записываем ее непосредственно в свой StringIO-эквивалентный файл в оперативной памяти.

Запрашиваемой темой являются заголовки важнейших новостей, которая указывается посредством пары ключевых значений `topic=h`. Другими параметрами являются: `lr` — для новостей, приковывающих всеобщее внимание; `w` — для мировых новостей; `n` — для новостей США; `b` — для новостей бизнеса; `tc` — для новостей из мира технологий; `e` — для новостей из мира развлечений; `s` — для новостей из мира спорта; `spc` — для новостей из мира науки; и `m` — для новостей, касающихся здоровья.

Файл с веб-соединением закрывается, и мы передаем этот файлоподобный объект на функцию `ElementTree.parse(f)`, которая выполняет синтаксический анализ соответствующего документа XML и возвращает экземпляр класса `ElementTree`. Обратите внимание: вы можете создать экземпляр ее самой, поскольку вызов `ElementTree.parse(f)` в этом примере эквивалентен `ElementTree.ElementTree(file=f)`. Наконец, закрываем файл в оперативной памяти.

Строки 30–50

Функция `topnews()` выполняет всю работу по упорядочиванию выходной информации для вызывающей программы. Мы лишь хотим вернуть надлежащим образом форматированные новостные материалы, поэтому создаем пару в форме списка пар, причем первым элементом каждой пары является заголовок, а вторым элементом — соответствующая ссылка. Лишь когда у нас есть и то и другое, мы формируем соответствующий элемент данных; в этой точке мы либо выходим из программы, если к этому времени уже вернули требуемое количество заголовков новостей (напоминаем, что по умолчанию это количество равняется пяти), либо просто сбрасываем эту пару.

Для первого заголовка (который в действительности не является заголовком новостей, каким обычно должен быть заголовок новостного типа) нам требуется специальный код. В нашем случае, поскольку мы запрашивали заголовки, мы получаем в поле “заголовок” нечто такое, что не является заголовком к новостям, а скорее некой “категорией” заголовков, содержимое которой представляет собой точную копию строки “Top Stories” (“Важнейшие новости”). Мы игнорируем все это.

Последняя пара строк в этом сценарии выводит “двойки”, генерируемые функцией `topnews()`.



Уголок хакера: сведение `topnews()` к одной (длинной) строке Python

Функцию `topnews()` можно свести к одной строке, которая выглядит весьма непривлекательно:

```
topnews = lambda count=5: [(x.text, y.text) for x, y in zip(
    (tree.getiterator('title'), tree.getiterator('link')) if not
    x.text.startswith('Top Stories')][:count]
```

Надеюсь, эта строка не перенапрягла ваше зрение? Тайный способ приготовления этого “блюда” заключается в применении функции `ElementTree.getiterator()` и предположении, что все новостные данные форматированы надлежащим образом. В стандартной версии `topnews()` не используются ни `zip()`, ни `itertools.izip()`, но в данном случае они используются для формирования пар заголовков и соответствующих им ссылок.

Обработка текста — не единственное, что может делать XML. Несмотря на то что следующий раздел посвящен именно XML, там очень мало говорится о XML как таковом. XML — это строительный блок, с помощью которого разработчики, создающие онлайн-службы, могут кодировать на верхнем уровне клиент-серверные приложения. Попросту говоря, вы не столько создаете какую-либо онлайн-службу, сколько предоставляете клиентам возможность вызывать функции или, говоря конкретнее, предоставляете им возможность выполнять удаленные вызовы процедур (Remote Procedure Call — RPC).

14.3.4. *Клиент-серверные службы с использованием протокола XML-RPC

Протокол XML-RPC был создан в конце 1990-х годов как способ, позволяющий разработчикам создать службу удаленного вызова процедур (RPC) с помощью протокола передачи гипертекста (HyperText Transfer Protocol — HTTP) как средства транспортировки, причем роль “полезной нагрузки” выполняет документ XML.

Этот документ содержит имя RPC и параметры, передаваемые соответствующей процедуре для выполнения. Впоследствии протокол XML-RPC привел к созданию протокола SOAP, но XML-RPC не так сложен, как SOAP. Поскольку формат JSON является более читабельным, чем формат XML, неудивительно, что существует и протокол JSON-RPC, в том числе и SOAP-версия, которая носит название SOAPjr.

Язык Python поддерживает протокол XML-RPC в трех пакетах: `xmlrpclib` на клиентской стороне, а также `SimpleXMLRPCServer` и `DocXMLRPCServer` на серверной. Вполне логично, что эти три пакета были реорганизованы в пакеты `xmlrpc.client` и `xmlrpc.server` в версии Python 3.x.

В примере 14.5 представлен `xmlrpcsrvr.py`, который является сценарием на языке Python 2, содержащим единую службу XML-RPC с широким разнообразием вызовов RPC. Сначала мы представим вам соответствующий код, а затем опишем каждую из служб, обеспечиваемых вызовами RPC.

Пример 14.5. Код сервера XML-RPC (`xmlrpcsrvr.py`)

Это пример сервера XML-RPC, который содержит множество функций RPC.

```

1  #!/usr/bin/env python
2
3  import SimpleXMLRPCServer
4  import csv
5  import operator
6  import time
7  import urllib2
8  import twapi # twapi.py from the "Web Services" chapter
9
10 server = SimpleXMLRPCServer.SimpleXMLRPCServer(("localhost", 8888))
11 server.register_introspection_functions()
12
13 FUNCS = ('add', 'sub', 'mul', 'div', 'mod')
14 for f in FUNCS:
15     server.register_function(getattr(operator, f))
16 server.register_function(pow)
17
18 class SpecialServices(object):
19     def now_int(self):

```

```
20         return time.time()
21
22     def now_str(self):
23         return time.ctime()
24
25     def timestamp(self, s):
26         return '[%s] %s' % (time.ctime(), s)
27
28     def stock(self, s):
29         url = 'http://quote.yahoo.com/d/quotes.csv?s=%s&f=llclp2dlt1'
30         u = urllib2.urlopen(url % s)
31         res = csv.reader(u).next()
32         u.close()
33         return res
34
35     def forex(self, s='usd', t='eur'):
36         url = 'http://quote.yahoo.com/d/quotes.csv?s=%s&f=nlldlt1'
37         u = urllib2.urlopen(url % (s, t))
38         res = csv.reader(u).next()
39         u.close()
40         return res
41
42     def status(self):
43         t = twapi.Twitter('twython')
44         res = t.verify_credentials()
45         status = twapi.ResultsWrapper(res.status)
46         return status.text
47
48     def tweet(self, s):
49         t = twapi.Twitter('twython')
50         res = t.update_status(s)
51         return res.created_at
52
53 server.register_instance(SpecialServices())
54
55 try:
56     print 'Welcome to PotpourriServ v0.1\n(Use ^C to exit)'
57     server.serve_forever()
58 except KeyboardInterrupt:
59     print 'Exiting'
```

Построчное объяснение

Строки 1–8

Разные операторы `import` включают прежде всего самый важный оператор `SimpleXMLRPCServer`, а также вспомогательные операторы, которые используются для предоставляемых служб. Эти службы даже включают использование кода сервера биржевых котировок Yahoo! и социальной сети Twitter (см. главу 13).

Сначала мы импортируем все модули/пакеты стандартной библиотеки, а затем модуль пользовательского уровня, `twapi`, который мы разработали для взаимодействия со службой Twitter. Порядок следования операторов импорта соответствует правилам “передового опыта”: стандартная библиотека, сторонние разработчики, а затем определенные пользователем.

Строки 10-11

После того как все операторы импорта будут выполнены, оператор SimpleXMLRPCServer учреждает нашу службу с заданным именем хоста или IP-адресом и номером порта. В этом случае мы просто используем *localhost* или *127.0.0.1*. Затем следует регистрация общепринятых функций интроспекции (самоанализа) протокола XML-RPC.

Эти функции позволяют клиентам выдавать запросы на сервер, чтобы определить его возможности. Они помогают клиенту определить, какие методы поддерживает данный сервер, как он может вызвать конкретную RPC и есть ли какая-либо документация для конкретной RPC. Вызовы, которые позволяют получить ответы на эти вопросы, называются `system.listMethods`, `system.methodSignature` и `system.methodHelp`.

Спецификации для этих функций интроспекции вы можете найти на сайте <http://scripts.incutio.com/xmlrpc/introspection.html>. Подробный пример того, как осуществить это на практике, предоставлен на сайте <http://www.doughellmann.com/PyMOTW/SimpleXMLRPCServer/#introspection-api>.

Строки 13-16

Эти четыре строки кода представляют стандартные арифметические функции, доступ к которым мы хотим обеспечить посредством RPC. Мы используем встроенную функцию (built-in function — BIF) `pow()` и извлекаем другие из модуля `operator`. Функция `server.register_function()` просто делает их доступными для клиентских запросов RPC.

Строки 18-26

Следующая совокупность функций, которые мы хотим добавить в нашу службу, касаются времени. Они также принимают форму созданного нами класса `SpecialServices()`. Нет принципиальной разницы между тем, присутствует ли какой-либо класс в коде или вне его, и мы хотели продемонстрировать это с помощью арифметических функций, а также следующих трех функций: `now_int()`, которая возвращает текущее время в секундах после момента начала отсчета времени; `now_str()`, которая возвращает “дружественную” к Unix метку времени, представляющую текущее время в местном временном поясе; а также вспомогательной функции `timestamp()`, которая использует некую строку в качестве входного параметра и возвращает метку времени, присоединенную к ней спереди.

Строки 28-40

Здесь мы заимствуем внушительный фрагмент кода из главы 13, начиная с кода, который обеспечивает взаимодействие с сервером биржевых котировок Yahoo!. Функция `stock()` берет тикер-символ какой-либо компании, а затем выбирает самую последнюю цену, последнее изменение, изменение в процентном выражении, а также дату и время последних торгов. Функция `forex()` делает нечто подобное, но для валютных курсов.

Использование кода из главы 13 необязательно, поэтому, если вы еще не успели ознакомиться с главой 13, не обращайте внимания на реализацию любой из этих функций, поскольку ни одна из них не нужна для усвоения концепций протокола XML-RPC.

Строки 42–53

Последние RPC, которые мы регистрируем, используют код для сети Twitter, который мы разработали в главе 13 с помощью библиотеки Twython. Функция `status()` извлекает текущий статус текущего пользователя, а `tweet()` выкладывает обновление статуса от имени данного пользователя. В последней строке этого блока мы регистрируем все функции в классе `SpecialServices()`, используя функцию `register_instance()`.

Строки 55–59

Эти последние пять строк запускают нашу службу (посредством бесконечного цикла), а также определяют, когда пользователь хочет выйти из этого процесса (путем нажатия комбинации клавиш <Ctrl+C>).

Теперь, когда у нас есть сервер, возникает вопрос: какая нам от него польза, если отсутствует клиентский код, который позволял бы воспользоваться всеми этими функциями? В примере 14.6 мы рассмотрим одно возможное клиентское приложение, `xmlrpcclnt.py`. Естественно, вы можете выполнить это на любом компьютере, который может обратиться к данному серверу с помощью соответствующей пары “хост-адрес порта”.

Пример 14.6. Клиентский код XML-RPC для Python 2 (`xmlrpcclnt.py`)

Это один из возможных клиентов, который выполняет обращения к нашему серверу XML-RPC.

```
1  #!/usr/bin/env python
2
3  from math import pi
4  import xmlrpclib
5
6  server = xmlrpclib.ServerProxy('http://localhost:8888')
7  print 'Current time in seconds after epoch:', server.now_int()
8  print 'Current time as a string:', server.now_str()
9  print 'Area of circle of radius 5:', server.mul(pi, server.pow(5, 2))
10 stock = server.stock('goog')
11 print 'Latest Google stock price: %s (%s / %s) as of %s at %s' %
    tuple(stock)
12 forex = server.forex()
13 print 'Latest foreign exchange rate from %s: %s as of %s at %s' %
    tuple(forex)
14 forex = server.forex('eur', 'usd')
15 print 'Latest foreign exchange rate from %s: %s as of %s at %s' %
    tuple(forex)
16 print 'Latest Twitter status:', server.status()
```

Клиентский фрагмент кода в данном случае не требует особых пояснений, тем не менее все же дадим некоторые пояснения.

Построчное объяснение

Строки 1–6

Для того чтобы обратиться к серверу XML-RPC, вам требуется модуль `xmlrpclib` в версии Python 2. Как указывалось выше, в Python 3 вам следовало бы воспользоваться модулем `xmlrpc.client`. Мы также извлекаем из модуля `math` константу `pi`. В первой строке реального кода мы подключаемся к серверу XML-RPC, передавая в него нашу пару “хост-адрес порта” как URL.

Строки 7–16

Каждая из оставшихся строк кода выполняет один RPC-запрос к серверу XML-RPC, который возвращает желаемые результаты. Единственной функцией, не тестируемой этим клиентом, является функция `tweet()`, которую мы оставляем в качестве упражнения для читателя. Выполнение столь большого количества обращений к серверу может показаться чрезмерным — и это действительно так. Вот почему в конце данной главы вы найдете упражнение, которое позволяет решить эту проблему.

Когда сервер функционирует, мы можем запустить на выполнение этот клиент и увидеть результат его выполнения (полученный вами результат наверняка будет другим):

```
$ python xmlrpclnt.py
Current time in seconds after epoch: 1322167988.29
Current time as a string: Thu Nov 24 12:53:08 2011
Area of circle of radius 5: 78.5398163397
Latest Google stock price: 570.11 (-9.89 / -1.71%) as of 11/23/2011 at
4:00pm
Latest foreign exchange rate from USD to EUR: 0.7491 as of 11/24/2011
at 3:51pm
Latest foreign exchange rate from EUR to USD: 1.3349 as of 11/24/2011
at 3:51pm
Latest Twitter status: @KatEller same to you!!! :-) we need a
celebration meal... this coming monday or friday? have a great
thanksgiving!!
```

Несмотря на то что мы достигли конца главы, мы лишь скользнули по самой поверхности программирования протоколов JSON-RPC и XML-RPC. Если хотите углубить свои познания в этой области, рекомендуем вам обратить внимание на документацию серверов XML-RPC посредством класса `DocXMLRPCServer`, всевозможные типы структур данных, которые вы можете возвращать от сервера XML-RPC (см. `xmlrpclib/ xmlrpc.client`), и т.п.

14.4. Литература

14.4.1. Дополнительные ресурсы

В Интернете можно найти множество документов, касающихся всего материала, о котором шла речь в этой главе. Приведенный ниже перечень, хоть и не является полным, содержит значительное количество ресурсов, которыми вы можете воспользоваться для углубления знаний, полученных в этой главе.

- <http://docs.python.org/library/csv>
- <http://json.org/>
- <http://simplejson.readthedocs.org/en/latest/>
- <http://pypi.python.org/pypi/simplejson>
- <http://github.com/simplejson/simplejson>
- <http://docs.python.org/library/json>
- <http://en.wikipedia.org/wiki/JSON>
- <http://en.wikipedia.org/wiki/XML>
- <http://docs.python.org/library/xmlrpclib>
- <http://docs.python.org/library/>
- `simplexmlrpcserver`
- <http://docs.python.org/library/docxmlrpcserver>
- <http://www.saxproject.org>
- [http://en.wikipedia.org/wiki/Expat_\(XML\)](http://en.wikipedia.org/wiki/Expat_(XML))
- <http://en.wikipedia.org/wiki/XML-rpc>
- <http://scripts.incutio.com/xmlrpc/introspection.html>
- <http://en.wikipedia.org/wiki/JSON-RPC>
- <http://json-rpc.org/>
- <http://www.doughellmann.com/PyMOTW/SimpleXMLRPCServer/#introspection-api>

Для более глубокого уяснения этого предмета рекомендуем обратиться к *Text Processing in Python* (Addison-Wesley, 2003) — классическому учебнику по этой теме. Есть еще одна книга по обработке текстов — *Python 2.6 Text Processing* (Pact, 2010). Название этой книги не должно вводить вас в заблуждение: информацию, которую вы найдете в этой книге, можно применить к большинству нынешних выпусков Python.

14.5. Модули, связанные с обработкой текстов

Таблица 14.2. Модули, связанные с обработкой текстов

Модуль/пакет	Описание
<code>csv</code> ^a	Обработка значений, разделяемых запятыми
<code>SimpleXMLRPCServer</code>	Сервер XML-RPC (включенный в <code>xmlrpc.server</code> в Python 3)
<code>DocXMLRPCServer</code>	Сервер самодокументирования XML-RPC (включенный в <code>xmlrpc.server</code> в Python 3)
<code>xmlrpclib</code>	Клиент XML-RPC (переименованный в <code>xmlrpc.client</code> в Python 3)
<code>json</code> ^b	Кодирование и декодирование JSON (известен широкому кругу пользователей как <code>simplejson</code> , обычно до версии 2.6)
<code>xml.parsers.expat</code> ^c	Быстродействующий синтаксический анализатор XML (без проверки на достоверность)
<code>xml.dom</code> ^b	Выполнение синтаксического анализа XML, базирующееся на дереве/DOM

Модуль/пакет	Описание
<code>xml.sax^b</code>	Выполнение синтаксического анализа XML, базирующееся на событиях/потоке
<code>xml.etree.ElementTree^c</code>	Синтаксический анализатор ElementTree XML и формирователь дерева

^a Новый в Python 2.3.

^b Новый в Python 2.6.

^c Новый в Python 2.0.

^d Новый в Python 2.5.

14.6. Упражнения

CSV

- 14.1. CSV. Каков формат CSV и для каких типов приложений он обычно подходит?
- 14.2. CSV *против* `str.split()`. Приведите несколько примеров данных, для которых было бы недостаточно вызова `str.split(',')` и где использование модуля `csv` является единственным способом решения проблемы.
- 14.3. CSV *или* `str.split()`. В упражнении 13.16 (см. главу 13) вам было предложено сделать вывод `stock.py` более гибким, обеспечив как можно более точное выравнивание всех столбцов, несмотря на переменную длину символов биржевого тикера, разные цены акций и изменения цен. Внесите изменения в этот сценарий, перейдя от вызова `str.split(',')` к методу `csv.reader`.
- 14.4. *Альтернативные форматы CSV*. Помимо запятой, существует еще несколько альтернативных разделителей. Например, файлы паролей, совместимые со стандартом POSIX, разделяются двоеточием (:), тогда как адреса в системе электронной почты Outlook разделяются точкой с запятой (;). Напишите функции, которые могут считывать или записывать документы, используя эти альтернативные разделители.

JSON

- 14.5. JSON. Каковы различия в синтаксисе между форматом JSON, с одной стороны, и словарями и списками Python, с другой?
- 14.6. *Массивы JSON*. Пример `dict2json.py` демонстрирует лишь переход от словарей (`dict`) Python к объектам JSON. Напишите аналогичный сценарий под именем `lort2json.py`, чтобы продемонстрировать переход от списков или кортежей к массивам JSON.
- 14.7. *Обратная совместимость*. Попытка выполнить пример 14.2, `dict2json.py` с помощью Python 2.5 и более старых версий не удастся:

```
$ python2.5 dict2json.py
Traceback (most recent call last):
  File "dict2json.py", line 12, in <module>
    from json import dumps
ImportError: No module named json
```

- а) Что нужно сделать, чтобы этот пример можно было выполнять с более старыми версиями Python?
 - б) Внесите изменения в ту часть кода `dict2json.py`, которая импортирует функциональность JSON, чтобы можно было работать с более старыми версиями Python (например, с версиями 2.4 и 2.5), а также с версией 2.6 и более новыми.
- 14.8. *JSON*. Добавьте в свой пример `stock.py` из главы 13 новый код, который извлекал бы биржевые котировки из службы Yahoo! Finance таким образом, чтобы возвращалась JSON-строка, представляющая все биржевые данные в формате иерархической структуры данных, в отличие от простого вывода (дампа) результатов на экран.
- 14.9. *JSON и типы/классы*. Разработайте сценарий, который кодировал бы и декодировал любой тип объектов Python, такой как числа, классы, экземпляры и т.п.

XML и XML-RPC

- 14.10. *Веб-программирование*. Усовершенствуйте сценарий `goognewsrss.py` таким образом, чтобы можно было выводить форматированный HTML, представляющий якоря/ссылки, которые можно было бы канализировать непосредственно на “плоский” файл `.html` для визуализации браузером. Ссылки должны быть правильными/достоверными, чтобы, щелкнув на любой такой ссылке, пользователь мог сразу же запустить соответствующую веб-страницу.
- 14.11. *Устойчивость и надежность*. Добавьте в сценарий `xmlrpcsrvr.py` поддержку операций `>`, `>=`, `<`, `<=`, `=`, `!=`, а также обычное и целочисленное деление.
- 14.12. *Twitter*. В сценарии `xmlrpcclnt.py` мы не тестировали метод `SpecialServices.tweet()`. Добавьте эту функциональность в свой сценарий.
- 14.13. *CGIXMLRPCRequestHandler*. По умолчанию `SimpleXMLRPCServer` использует класс обработчика `SimpleXMLRPCRequestHandler`. Какова разница между этим обработчиком и `CGIXMLRPCRequestHandler`? Разработайте новый сервер, который использовал бы `CGIXMLRPCRequestHandler`.
- 14.14. *DocXMLRPCServer*. Изучите серверы самодокументирования протокола XML-RPC, а затем ответьте на следующие вопросы:
- а) В чем заключаются различия между объектами `SimpleXMLRPCServer` и `DocXMLRPCServer`? В чем заключаются различия нижнего уровня (по сети), помимо различий, указанных вами при ответе на первый вопрос?
 - б) Преобразуйте свой стандартный клиент и сервер XML-RPC так, чтобы обеспечивалась функция самодокументирования.
 - в) Также преобразуйте свою версию CGI из предыдущей задачи так, чтобы использовался класс `DocCGIXMLRPCRequestHandler`.
- 14.15. *Мультивызовы XML-RPC*. В `xmlrpcclnt.py` мы выполняли индивидуальные запросы к серверу. Более производительными являются клиенты, выполняющие множественные обращения к серверу, поскольку способны выполнять так называемый *мультивызов*, т.е. несколько RPC-вызовов с помощью одного запроса на обслуживание к серверу. Изучите функцию `register_multicall_functions()`, а затем добавьте эту функциональность в свой

сервер. Наконец, измените свой клиент таким образом, чтобы можно было использовать мультивызов.

- 14.16.** *XML и XML-RPC.* Каким образом какой-либо из материалов о протоколе XML-RPC, освещавшихся в этой главе, связан с языком XML? Материал в последнем разделе совершенно отличается от остального материала этой главы; каким образом они связаны между собой и что между ними общего?
- 14.17.** *JSON-RPC или XML-RPC.* Что представляет собой протокол JSON-RPC и как он связан с протоколом XML-RPC?
- 14.18.** *JSON-RPC.* Перенесите код своего клиента и сервера XML-RPC в эквивалентные им `jsonrpcsrvr.py` и `jsonrpcclnt.py`.

ГЛАВА

15

Разное

В этой главе...

- Интерпретатор Jython
- Служба Google+

В компании Google язык Python является одним из трех "официальных" языков наряду с C++ и Java.

Грег Стейн, март 2005 г.
(из выступления на собрании SDForum)

Как и в главе 14, в этой главе вы ознакомитесь с кратким введением в области программирования на языке Python, на более полное рассмотрение которых у нас не было времени. Надеемся, что со временем, в будущих изданиях этой книги, мы сможем посвятить каждой из этих областей отдельную главу. Начнем с программирования на языках Java и Jython, а затем перейдем к обсуждению интерфейса API Google+.

15.1. Интерпретатор Jython

В первой части этой главы мы покажем, как реализовать язык Python на виртуальной машине JVM с помощью интерпретатора Jython. Сначала мы кратко рассмотрим, что такое интерпретатор Jython, и продемонстрируем сходство его работы со средой Python (а также посмотрим, в чем его отличие). Это обсуждение будет сопровождаться анализом примера кода графического пользовательского интерфейса с использованием библиотеки Swing. Затем мы продемонстрируем Java-код, сопровождаемый его эквивалентом на языке Python и выполняемый с помощью интерпретатора Jython (несмотря на то, что такой вариант использования Java не является характерным для этого языка, он может служить весьма поучительным примером). Надеемся, что для будущих изданий этой книги мы разработаем больше примеров на языке Java.

15.1.1. Введение в интерпретатор Jython

Интерпретатор Jython является одним из тех инструментов, которые способны объединить сторонников двух принципиально разных направлений в программировании. Во-первых, он удовлетворяет потребностям программистов, работающих на языке Python, но в среде разработки Java, и позволяет им быстро находить прототипы решений, которые идеально вписываются в любую существующую Java-платформу. Еще одной причиной является то, что интерпретатор Jython помогает упростить жизнь миллионов программистов, работающих на языке Java, предоставляя для Java среду сценарного языка. Благодаря этому им уже не придется разрабатывать тест или приложение-драйвер лишь для того, чтобы протестировать созданный ими класс.

Среда Jython предоставляет в ваше распоряжении большую часть того, что может предложить вам среда Python, наряду со способностью создавать экземпляры и взаимодействовать с классами Java! Код на языке Jython динамически компилируется в байт-код Java, к тому же в среде Jython вы можете расширять классы Java. С помощью языка Java вы можете также расширять язык Python. Вы можете достаточно легко написать какой-либо класс на языке Python, а затем использовать его как класс языка Java. Вы всегда можете статически скомпилировать какой-либо сценарий среды Jython в байт-код Java.

Интерпретатор Jython можно загрузить с веб-сайта этой книги или с сайта <http://jython.org>. Когда вы в первый раз выполняете интерактивный интерпретатор Jython, он отображает уведомления, извещающие вас о том, что обрабатываются новые файлы с расширением .jar, как показано в приведенном ниже примере.


```
$ jython
*sys-package-mgr*: processing new jar, '/usr/local/jython2.5.2/
jython.jar'
*sys-package-mgr*: processing new jar, '/System/Library/Java/
JavaVirtualMachines/1.6.0.jdk/Contents/Classes/classes.jar'
. . .
*sys-package-mgr*: processing new jar, '/System/Library/Java/
JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/sunpkcs11.jar'
Jython 2.5.2 (Release_2_5_2:7206, Mar 2 2011, 23:12:06)
[Java HotSpot(TM) 64-Bit Server VM (Apple Inc.)] on java1.6.0_26
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Каждый последующий вызов выглядит так, словно вы используете среду Python. К тому же вы по-прежнему можете выполнить хорошо знакомое приложение “Hello World!” на языке Python:

```
$ jython
Jython 2.5.2 (Release_2_5_2:7206, Mar 2 2011, 23:12:06)
[Java HotSpot(TM) 64-Bit Server VM (Apple Inc.)] on java1.6.0_26
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Hello World!'
Hello World!
```

Еще более интересным в интерактивном интерпретаторе Jython является то, что теперь вы можете выполнить приложение “Hello World!” с помощью языка Java:

```
>>> from java.lang import System
>>> System.out.write('Hello World!\n')
Hello World!
```

Язык Java предоставляет пользователям языка Python дополнительные преимущества естественной обработки исключений (отсутствующие в стандартном языке Python, или *CPython*, как его называют, упоминая о нем в числе других реализаций) и использования собственного механизма сбора мусора в языке Java (поэтому разработчикам не приходится заниматься переделкой механизма сбора мусора в языке Python для языка Java).

15.1.2. Пример графического пользовательского интерфейса с использованием библиотеки Swing

Располагая доступом ко всем классам языка Java, мы значительно расширяем круг своих возможностей. Одним из примеров может служить разработка графического пользовательского интерфейса. В среде Python по умолчанию используется интерфейс GUI Tk посредством модуля Tkinter, но Tk не является для Python “родным” инструментарием. Однако в языке Java есть библиотека Swing, которая является “родной” для него. При использовании интерпретатора Jython мы можем фактически разработать приложение с графическим пользовательским интерфейсом, используя компоненты Swing, — не с помощью Java, а Python.

Пример 15.1 представляет собой простое приложение “Hello World!” с графическим пользовательским интерфейсом, написанное на языке Java. Затем мы приводим пример 15.2, который показывает его эквивалент на языке Python. Оба этих примера являются подражанием примерам `tkhello3.py`, приведенным в главе 5. Эти программы называются `swhello.java` и `swhello.py` соответственно.

Пример 15.1. Программа "Hello World!" на языке Java (swhello.java)

Эта программа создает графический пользовательский интерфейс точно так же, как сценарий tkhello3.py, но вместо инструментария Tk используется библиотека Swing.

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.lang.*;
5
6  public class swhello extends JFrame {
7      JPanel box;
8      JLabel hello;
9      JButton quit;
10
11     public swhello() {
12         super("JSwing");
13         JPanel box = new JPanel(new BorderLayout());
14         JLabel hello = new JLabel("Hello World!");
15         JButton quit = new JButton("QUIT");
16
17         ActionListener quitAction = new ActionListener() {
18             public void actionPerformed(ActionEvent e) {
19                 System.exit(0);
20             }
21         };
22         quit.setBackground(Color.red);
23         quit.setForeground(Color.white);
24         quit.addActionListener(quitAction);
25         box.add(hello, BorderLayout.NORTH);
26         box.add(quit, BorderLayout.SOUTH);
27
28         addWindowListener(new WindowAdapter() {
29             public void windowClosing(WindowEvent e) {
30                 System.exit(0);
31             }
32         });
33         getContentPane().add(box);
34         pack();
35         setVisible(true);
36     }
37
38     public static void main(String args[]) {
39         swhello app = new swhello();
40     }
41 }

```

Пример 15.2. Приложение "Hello World!" на языке Python (swhello.py)

Это сценарий на языке Python, эквивалентный приведенной выше программе на языке Java; он выполняется с использованием интерпретатора Jython.

```

1  #!/usr/bin/env jython
2
3  from pawt import swing

```

```
4 import sys
5 from java.awt import Color, BorderLayout
6
7 def quit(e):
8     sys.exit()
9
10 top = swing.JFrame("PySwing")
11 box = swing.JPanel()
12 hello = swing.JLabel("Hello World!")
13 quit = swing.JButton("QUIT", actionPerformed=quit,
14                     background=Color.red, foreground=Color.white)
15
16 box.add("North", hello)
17 box.add("South", quit)
18 top.contentPane.add(box)
19 top.pack()
20 top.visible = 1 # or True for Jython 2.2+
```

Коды обоих примеров соответствуют коду `tkhello3.py`, за исключением того, что вместо инструментария Tk они используют библиотеку Swing. Оба эти сценария мы опишем параллельно.

Поблочное совместное объяснение кода

Как программа `swhello.java`, так и программа `swhello.py` начинаются с импортирования требуемых модулей, библиотек и пакетов. Следующие блоки кода в каждом из двух сценариев используют примитивы библиотеки Swing. Обратный вызов процедуры выхода из программы выполняется в блоке кода Java, тогда как код Python определяет эту функцию до вхождения в базовую часть приложения.

После того как будут определены графические элементы управления, следующие блоки кода помещают их в предназначенные для них места в соответствующем пользовательском интерфейсе. Последним действием является размещение всего на панели контента, упаковка всех графических элементов управления и визуализация всего интерфейса на экране.

Особенностью версии на языке Python является существенное сокращение количества строк кода, необходимого для выполнения тех же функций на языке Java. Код на языке Python более выразителен, а каждая строка этого кода несет в себе большую смысловую нагрузку. Коротко говоря, в коде на языке Python присутствует меньше “белого шума”. Код на языке Java, как правило, оказывается более “многословным”, и для выполнения необходимого объема работы требуется гораздо больше стандартных операторов; при использовании языка Python вы можете сосредоточиться на важных частях своего приложения, т.е. на решении поставленной перед вами задачи.

В то время как оба эти приложения скомпилированы в байт-код Java, неудивительно, что они практически неотличимы друг от друга при выполнении на одной и той же платформе (рис. 15.1).

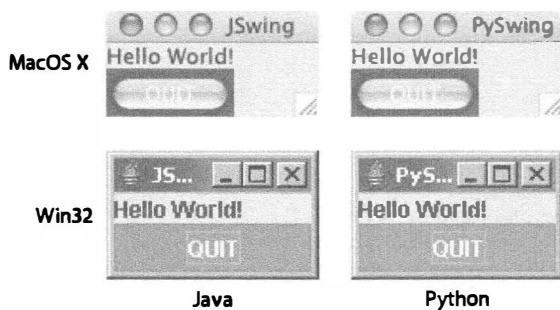


Рис. 15.1. Демонстрационные сценарии "Hello World!" с использованием библиотеки Swing. (swhello.{java, py})

Интерпретатор Jython считается великолепным инструментом для разработки, поскольку он обеспечивает выразительность языка Python плюс богатство интерфейсов API в библиотеках языка Java. Если вы являетесь разработчиком программ на языке Java, то мы надеемся, что вы уже осознали, какие возможности открываются перед вами, учитывая богатый потенциал языка Python. Если же ваш опыт использования языка Java пока невелик, то интерпретатор Jython может облегчить вашу задачу. Вы можете создать в среде Jython прототип, а затем, по мере необходимости, легко перенести свою работу на язык Java. Кроме того, Jython является великолепной средой для разработки сценариев, которая может удачно дополнять повседневную разработку с помощью языка Java.

15.2. Служба Google+

Вторая половина этой экспериментальной главы относится к социальной платформе компании Google — службе Google+. Сначала мы кратко разъясним, что такое служба Google+, а затем обсудим, как подключиться к ней из среды Python. Наконец, мы предложим вашему вниманию краткий пример кода, чтобы продемонстрировать несколько возможностей, которые предоставляет служба Google+. Поскольку этот продукт всегда добавляет новые возможности, надеемся, что в будущем продемонстрируем вам еще кое-что из них.

15.2.1. Введение в платформу Google+

Службу Google+ можно рассматривать как еще одну социальную платформу с интерфейсом прикладного программирования (API). Службу Google+ можно рассматривать (с корпоративной точки зрения) и так: она включает интерфейс Google+ в большинство своих продуктов; при этом интерфейс Google+ служит дополнением к уже существующему набору возможностей (именно этим, наверное, объясняется его название — Google+).

Как бы вы ни рассматривали эту службу, в ней, несомненно, есть определенный социальный аспект, и, как и в большинстве продуктов компании Google, для него предусмотрен свой интерфейс API. С помощью службы Google+ пользователи могут размещать сообщения и фотографии, отслеживать действия других пользователей и показывать, что им нравится то или иное сообщение, щелкнув на кнопке +1, ассоциирующейся с этим сообщением. Пользователи могут прокомментировать заинтересовавшее их сообщение и/или поделиться им с одним или несколькими *кругами* (так

в службе Google+ принято обозначать подгруппы в ваших сетях) или обсудить это сообщение со сколь угодно широким кругом пользователей.

Как уже упоминалось, у службы Google+ есть интерфейс прикладного программирования (API). На момент написания этой книги разработчики могут использовать этот интерфейс для доступа к пользователям службы Google+ и их поиска, а также для доступа к потокам действий пользователей службы Google+, включая комментарии к этим действиям. Разработчики могут также писать приложения, интегрирующие программное обеспечение Google+ Hangouts, поскольку у него также имеется интерфейс API. Такие интерфейсы позволяют разработчикам создавать приложения, которые могут осуществлять поиск публичных постов и извлекать пользовательские профили. Рассмотрим пример.

15.2.2. Среда Python и интерфейс API Google+

Доступ к этой функциональности из среды Python достаточно прост. Однако в своем кратком обзоре мы даже не пытаемся затронуть вопрос доступа к материалам, использование которых предусматривает аутентификацию пользователя (если ваше приложение имеет право доступа к данным Google+, то ваши возможности существенно расширяются). Поэтому необходимо иметь в виду, что в действительности этот пример наверняка окажется даже проще.

2.4

Прежде чем приступить к рассмотрению примера, вы должны сначала установить библиотеку Client Library для среды Python, если до сих пор не сделали этого. Вы без труда можете сделать это с помощью такого инструмента, как `pip` или `easy_install`. (Вам потребуется версия Python 2.4 или какая-либо из более поздних версий, поскольку для версии 3.x этой библиотеки еще нет.) При использовании инструмента `easy_install` команда обновления/инсталляции будет выглядеть примерно так:

```
$ sudo easy_install --upgrade google-api-python-client
```

Эту библиотеку можно использовать для доступа ко многим службам компании Google, а не только для интерфейса API Google+. Полный перечень поддерживаемых интерфейсов API можно найти на сайте <http://code.google.com/p/google-api-python-client/wiki/SupportedApis>.

Далее, нам нужен ключ доступа. Обратитесь на сайт <http://code.google.com/apis/console> и создайте новый проект. Выберите во вновь созданном проекте вкладку Services и установите флажок Google+ API. Затем выберите вкладку API Access, скопируйте ключ API и вставьте его в следующий код (или, еще лучше, выполните рефакторинг привилегированных данных, таких как полномочия, в отдельный файл). Выполнив указанные действия, можно приступать к решению поставленной задачи.

15.2.3. Простой инструмент анализа социальных средств коммуникации

В примере 15.3 представлен созданный нами простой инструмент анализа социальных средств коммуникации. С помощью этого инструмента вы можете узнать, что говорят люди в службе Google+, обсуждая ту или иную тему. Однако не все создаваемые посты равнозначны. Некоторые из них видит лишь очень небольшое число людей, тогда как другие активно обсуждаются и многократно передаются от одного пользователя к другому.

В нашем приложении мы сосредоточимся на этих популярных постах. Оно ранжирует посты по их популярности и создает список пяти самых популярных постов за прошлую неделю, посвященных либо языку Python, либо условию поиска (в зависимости от того, что интересует соответствующего пользователя). Еще одной, хоть и не очень существенной функцией в этом сценарии является способность находить и отображать профиль интересующего нас пользователя службы Google+.

Наш сценарий называется `plus_top_posts.py`, но прежде чем приступить к рассмотрению его кода, рассмотрим приведенный ниже пример результатов выполнения этой программы, работающей под управлением меню. Это позволит вам составить представление о том, как она работает:

```
$ python plus_top_posts.py
-----
Google+ Command-Line Tool v0.3
-----
(p) Top 5 Python posts in past 7 days
(u) Fetch user profile (by ID)
(t) Top 5 posts in past 7 days query
Enter choice [QUIT]: p
*** Searching for the top 5 posts matching 'python' over the past 7
days...
From: Gretta Bartels (110414482530984269464)
Date: Fri Nov 25 02:01:16 2011
Chatter score: 19
Post: Seven years old. Time to learn python. And maybe spelling.
Link: https://plus.google.com/110414482530984269464/posts/MHSDkdxEyE7
-----
From: Steven Van Bael (106898588952511738977)
Date: Fri Nov 25 11:00:50 2011
Chatter score: 14
Post: Everytime I open a file in python I realize how awesome the
language actually is for doing utility scripts. f =
open('test.txt','w') f.write('hello world') f.close() Try doing that
in java
Link: https://plus.google.com/106898588952511738977/posts/cBRko8luYX2
-----
From: Estevan Carlos Benson (115832511083802586044)
Date: Fri Nov 25 20:02:11 2011
Chatter score: 11
Post: Can anyone recommend some online Python resources for a
beginner. Also, for any python developers, your thoughts on the
language?
Link: https://plus.google.com/115832511083802586044/posts/9GNwa9TXHzt
-----
From: Michael Dorsey Jr (103222958721998092839)
Date: Tue Nov 22 11:31:56 2011
Chatter score: 11
Post: I slowly but surely see python becoming my language of choice.
Programming language talk at the gym. Must be cardio time.
Link: https://plus.google.com/103222958721998092839/posts/jRuPPDpfndv
-----
From: Gabor Szabo (102810219707784087582)
Date: Fri Nov 25 17:59:14 2011
Chatter score: 9
Post: In http://learnpythonthehardway.org/ Zed A. Shaw suggest to read
code backwards. Any idea why would that help? Anyone practicing
```

anything like that?

Link: <https://plus.google.com/102810219707784087582/posts/QEC5TQlqoQU>

Google+ Command-Line Tool v0.3

(p) Top 5 Python posts in past 7 days

(u) Fetch user profile (by ID)

(t) Top 5 posts in past 7 days query

Enter choice [QUIT]: u

Enter user ID [102108625619739868700]:

Name: wesley chun

URL: <https://plus.google.com/102108625619739868700>

Pic: https://lh3.googleusercontent.com/-T_wVWLmg7w/AAAAAAAAAAI/AAAAAAAAAA/zeVf2azgGYI/photo.jpg?sz=50

About: WESLEY J. CHUN, MSCS, is the author of Prentice Hall's bestseller, [<i>Core Python Programming</i>](http://corepython.com), its video

. . .

Google+ Command-Line Tool v0.3

(p) Top 5 Python posts in past 7 days

(u) Fetch user profile (by ID)

(t) Top 5 posts in past 7 days query

Enter choice [QUIT]:

\$

Теперь проанализируем исходный код.

Пример 15.3. Простой инструмент анализа социальных средств коммуникации (`plus_top_posts.py`)

Этот сценарий на языке Python 2 выполняет в службе Google+ поиск определенных запросов и пользовательских профилей.

```
1  #!/usr/bin/env python
2
3  import datetime as dt
4  from apiclient.discovery import build
5
6  WIDTH = 40
7  MAX_DEF = 5
8  MAX_RES = 20
9  MAX_TOT = 60
10  UID = '102108625619739868700'
11  HR = '\n%s' % ('-' * WIDTH)
12  API_KEY = 'YOUR_KEY_FROM_CONSOLE_API_ACCESS_PAGE'
13
14  class PlusService(object):
15      def __init__(self):
16          self.service = build("plus", "v1",
17                              developerKey=API_KEY)
18
19      def get_posts(self, q, oldest, maxp=MAX_TOT):
20          posts = []
21          cap = min(maxp, MAX_RES)
```

```

22     cxn = self.service.activities()
23     handle = cxn.search(maxResults=cap, query=q)
24     while handle:
25         feed = handle.execute()
26         if 'items' in feed:
27             for activity in feed['items']:
28                 if oldest > activity['published']:
29                     return posts
30                 if q not in activity['title']:
31                     continue
32                 posts.append(PlusPost(activity))
33                 if len(posts) >= maxp:
34                     return posts
35             handle = cxn.search_next(handle, feed)
36         else:
37             return posts
38     else:
39         return posts
40
41     def get_user(self, uid):
42         return self.service.people().get(userId=uid).execute()
43
44     scrub = lambda x: ' '.join(x.strip().split())
45
46     class PlusPost(object):
47         def __init__(self, record):
48             self.title = scrub(record['title'])
49             self.post = scrub(record['object'].get(
50                 'originalContent', ''))
51             self.link = record['url']
52             self.when = dt.datetime.strptime(
53                 record['published'],
54                 "%Y-%m-%dT%H:%M:%S.%fZ")
55             actor = record['actor']
56             self.author = '%s (%s)' % (
57                 actor['displayName'], actor['id'])
58             obj = record['object']
59             cols = ('replies', 'plusoners', 'resharers')
60             self.chatter = \
61                 sum(obj[col]['totalItems'] for col in cols)
62
63     def top_posts(query, maxp=MAX_DEF, ndays=7):
64         print '''
65         *** Searching for the top %d posts matching \
66         %r over the past %d days...''' % (maxp, query, ndays)
67         oldest = (dt.datetime.now()-dt.timedelta(ndays)).isoformat()
68         posts = service.get_posts(query, oldest, maxp)
69         if not posts:
70             print '*** no results found... try again ***'
71             return
72         sorted_posts = sorted(posts, reverse=True,
73             key=lambda post: post.chatter)
74         for i, post in enumerate(sorted_posts):
75             print '\n%d)' % (i+1)
76             print 'From:', post.author
77             print 'Date:', post.when.ctime()
78             print 'Chatter score:', post.chatter
79             print 'Post:', post.post if len(post.post) > \

```



```

80         len(post.title) else post.title
81     print 'Link:', post.link
82     print HR
83
84     def find_top_posts(query=None, maxp=MAX_DEF):
85         if not query:
86             query = raw_input('Enter search term [python]: ')
87         if not query:
88             query = 'python'
89     top_posts(query, maxp)
90     py_top_posts = lambda: find_top_posts('python')
91
92     def find_user():
93         uid = raw_input('Enter user ID [%s]: ' % UID).strip()
94         if not uid:
95             uid = UID
96         if not uid.isdigit():
97             print '*** ERROR: Must enter a numeric user ID'
98             return
99         user = service.get_user(uid)
100        print 'Name:', user['displayName']
101        print 'URL:', user['url']
102        print 'Pic:', user['image']['url']
103        print 'About:', user.get('aboutMe', '')
104
105    def _main():
106        menu = {
107            't': ('Top 5 posts in past 7 days query', find_top_posts),
108            'p': ('Top 5 Python posts in past 7 days', py_top_posts),
109            'u': ('Fetch user profile (by ID)', find_user),
110        }
111        prompt = ['(%s) %s' % (item, menu[item][0]) for item in menu]
112        prompt.insert(0, '%s\n%s' % (HR,
113            'Google+ Command-Line Tool v0.3'.center(WIDTH), HR))
114        prompt.append('\nEnter choice [QUIT]: ')
115        prompt = '\n'.join(prompt)
116        while True:
117            ch = raw_input(prompt).strip().lower()
118            if not ch or ch not in menu:
119                break
120            menu[ch][1]()
121
122    if __name__ == '__main__':
123        service = PlusService()
124        _main()

```

Построчное объяснение

Строки 1–12

Интересное обстоятельство: несмотря на то, что этот сценарий — один из самых длинных в нашей книге, в нем присутствует лишь два импорта. Одним является пакет стандартной библиотеки `datetime`; другой заимствован из библиотеки `Client Library API Google` для среды Python. Что касается последней библиотеки, то нас интересует лишь функция `build()`. По сути, одной из причин того, почему эта часть оказалась столь простой, является то, что мы проигнорировали проблему обеспечения

безопасности (авторизации). Впрочем, у вас будет возможность восполнить это упущение в одном из упражнений, которые вы найдете в конце этой главы.

В последней части этого блока кода размещены константы, которыми мы намерены воспользоваться. Переменные `WIDTH` и `HR` касаются лишь отображения пользователя. `API_KEY` решает вопрос вашей аутентификации на сервере Google и получения доступа к службе API Google+ (данные общего пользования). Настоятельно рекомендуем вам вывести это значение за пределы своей логики, в какой-либо другой файл, например `secret.py`, в целях повышения его безопасности; вам следует предоставить файл `secret.py` лишь в случае, если к вашим файлам обращаются другие пользователи. (Расширение `.рус` не является “защитой от дурака”, но требует от потенциальных злоумышленников знания внутреннего устройства виртуальной машины языка Python, чтобы попытаться самостоятельно воспроизвести ее.) Если вы потеряли его, то выше в этой главе можете найти инструкции, как получить ключ API.

Параметр `MAX_DEF` — это предусмотренное по умолчанию количество отображаемых результатов; `MAX_RES` — это максимальное количество результатов поиска, которые вы можете (на данный момент) запросить от интерфейса API Google+; `MAX_TOT` — это текущий максимум, который мы разрешаем запрашивать пользователям этого сценария; а `UID` — это идентификатор пользователя, предусмотренный по умолчанию (уточню: это пользовательский идентификатор вашего покорного слуги).

Строки 14–17

Класс `PlusService` обеспечивает главный интерфейс вашего инструмента с интерфейсом API Google+. В инициализаторе `_init_()` мы подключаемся к этому интерфейсу прикладного программирования, вызывая функцию `apiclient.discovery.build()` и передавая ей желаемый интерфейс API (Google+ представлен с помощью “plus” и номера его версии) и ваш ключ интерфейса API.

Строки 19–39

Метод `get_posts()` выполняет большой объем работы. Он осуществляет установку и фильтрацию; к тому же он выполняет первичное обращение для получения данных от Google. Он начинается с инициализации списка результатов (`posts`), задания максимального количества результатов, которые можно запросить от Google (это количество не должно превышать `MAX_RES`), кеширования соединения с API и выполнения начального обращения к API Google+, возвращая значение `handle`, если запрос оказался успешным. Цикл `while` обеспечивает “бесконечное” выполнение (точнее говоря, до того момента, когда от интерфейса API перестанут поступать результаты). Как будет показано ниже, есть и другие способы выхода из этого цикла.

Используя параметр `handle` этого соединения, мы выполняем запрос и принимаем объект `feed` от службы Google+. Он приходит по проводу в виде некой JSON-структуры и преобразуется в словарь Python. Если в принятых данных есть элементы (т.е. имеется ключ с именем ‘items’), мы циклически обрабатываем их, принимая и сохраняя соответствующие данные; в противном случае данных больше нет, поэтому мы выходим из цикла и возвращаем любые промежуточные результаты. Находясь внутри этого цикла, мы также можем выйти из него, руководствуясь фактором времени. Поскольку интерфейс API возвращает результаты в обратной хронологической последовательности, как только мы получим посты за одну неделю, мы знаем, что все оставшиеся посты появились еще раньше; следовательно, можем выйти из цикла и вернуть принятую совокупность данных.

Сравнение по времени создания осуществляется путем непосредственного сравнения временных меток ISO 8601/RFC 3339. Все посты службы Google+ отправляются нам, естественно, в этом формате, тогда как нашу временную метку необходимо преобразовать из объекта `datetime.datetime` в соответствующий ISO-эквивалент. (См. это преобразование в приведенном ниже описании `top_posts()`.)

Следующий фильтр игнорирует все посты, в заголовках которых не указано поисковое условие. Это, наверное, не самый точный способ поиска, поскольку поисковое условие может появиться, например, во вложении или в теле поста (в этом случае необходимо просматривать все содержимое поста). У вас будет возможность исправить эту ситуацию в упражнениях, приведенных в конце данной главы. Это является последним тестом или фильтром, предусмотренным в нашем решении; сюда можно добавить любые другие, придуманные вами.

После того как мы передали эти тесты, создаем новый объект `PlusPost`, передавая его инициализатор всему объекту поста и заставляя его отфильтровать лишь те поля, которые актуальны для нас.

Далее, проверяем, достигли ли мы максимального количества результатов. Если достигли, выходим из цикла. В противном случае вызываем метод `search_next()` из интерфейса API Google+, передавая в него параметры `handle` и `feed` текущего соединения, чтобы ему было известно, где мы остановились последний раз (не правда ли, это похоже на то, как действует курсор?).

Последнее выражение `else` выполняет возврат, если поиск в целом оказался безрезультатным.

Строки 41–44

Последним методом в нашем классе является метод `get_user()`. Поскольку он считается функцией, которую должен выполнить пользователь, а не действием компьютера, мы используем метод `self.service.people()`, а не `self.service.activity()`. Метод `get()` должен найти информацию об определенном пользователе Google+ по его идентификатору.

Последняя строка кода в этом разделе содержит служебную операцию `scrub()`, которая принимает многострочное тело текста и сокращает его до единственной строки, заменяя все пробелы (даже следующие друг за другом) одним символом пробела. Это помогает управлять выводом в сценарии командной строки, но не так уж необходимо для эквивалентных веб-приложений.

Строки 46–61

Назначение объекта `PlusPost` заключается в том, чтобы создать избавленный от всего лишнего и обгороженный эквивалент данных того или иного поста, который содержал бы лишь интересующий нас материал. Этот объект представляет отдельно взятый пост службы Google+, созданный каким-либо пользователем. Интерфейс API Google+ возвращает хорошо структурированные данные объектов JSON (JavaScript Object Notation) и массивы, преобразованные в словари (`dict`) и списки (`list`) языка Python, ориентироваться в которых бывает довольно сложно. Этот класс преобразует такую структуру данных в плоский объект, самые важные свойства которого представлены как переменные экземпляров.

Заголовок и содержимое поста вычищаются операцией `scrub()`, указатель URL принимается таким, какой он есть, и используется более читабельная временная метка. Сохраняемая информация об исходном отправителе поста включает имя и

идентификатор отображения, причем идентификатор можно использовать для получения дополнительной информации о соответствующем пользователе. Выполняется синтаксический анализ временной метки поста, которая преобразуется из ISO 8601/RFC 3339 в естественный для среды Python объект `datetime.datetime`, а затем присваивается.

Рейтинг упоминаемости (*chatter score*) помогает оценить влияние и релевантность соответствующего поста. *Рейтинг упоминаемости* в данном случае определяется как сумма отметок "1" и перепостов данного поста. Чем выше рейтинг упоминаемости, тем важнее соответствующий пост для нашего (и чьего-либо еще) инструмента анализа социальных средств коммуникации. Все эти сигналы подаются в структуре данных 'object' и сохраняются в поле `totalItems` для каждого из этих показателей. В данном коде `sum()` используется для подсчета всех этих значений столбцов посредством соответствующего генераторного выражения и вычисления рейтинга упоминаемости.

Более очевидный способ написания этого кода суммирования, если вы еще недостаточно хорошо освоили язык Python, таков:

```
self.chatter = api_record['object']['replies']['totalItems']\
    + api_record['object']['plusoners']['totalItems']\
    + api_record['object']['resharers']['totalItems']
```

Мы внесли в этот код ряд изменений, чтобы получить приведенный выше исходный код. Какие изменения мы внесли и почему?

1. Выполняется повторяющийся просмотр `api_record['object']`. Если этот код выполнить лишь несколько раз, то это практически не скажется на производительности системы. Однако если мы выполняем на сервере миллионы просмотров в сутки, то со временем это существенно повлияет на производительность. В языке Python эта проблема обычно решается путем назначения некой локальной переменной для кеширования таких ссылок, например `obj = record['object']`.
2. По-видимому, мы выбираем столбец с одним и тем же именем (`totalItems`), так почему бы и это не использовать повторно.
3. Вместо того чтобы добавлять значения вручную, используя плюс (+), я зачастую предпочитаю полагаться на встроенные функции, такие как `sum()`, поскольку они обычно бывают написаны на языке C и работают быстрее, чем при использовании только языка Python.
4. Если мы можем повысить наш рейтинг упоминаемости с помощью дополнительных показателей, то это влечет за собой добавление еще одной длинной строки кода, например `" + api_record['object'][SOMETHING_ELSE]['totalItems']"`, в то время как мы могли бы добавить к нашему полю столбцов лишь единственное слово, например `cols = ('replies', 'plusoners', 'resharers', SOMETHING_ELSE)`.

Учитывая, что одной из самых важных целей при создании кода на языке Python является обеспечение хорошей читабельности, простоты и элегантности, в данном случае любое из предложенных решений окажется эффективным. Другое дело, если бы рейтинг упоминаемости требовал суммирования, например, десяти значений.

Еще раз возвращаясь к нашей теме, отметим: несмотря на то, что рейтинг упоминаемости похож на показатель Ripples Google+, это не одно и то же, поскольку

Ripples — это в большей степени инструмент, который обеспечивает *визуальное представление* рейтинга упоминаемости того или иного поста. (Подробнее об инструменте Ripples можно прочитать на сайтах <http://googleblog.blogspot.com/2011/10/google-popular-posts-eye-catching.html> и <http://google.com/support/plus/bin/answer.py?answer=1713320>.)

Строки 63–82

Функция `top_posts()` представляет пользовательский интерфейс для поиска постов. Она отображает сообщение, указывающее начало запроса, объединяет и сортирует результаты, а затем отображает их, один за другим, пользователю. Эта функция выполняет сортировку в порядке убывания рейтинга упоминаемости, согласно вызову встроенной функции `sorted()`.

По умолчанию приложение отображает верхние пять отвечающих условию поиска и самых релевантных постов. Это можно изменить с помощью последнего оператора `if`. Вы можете изменить также величину совокупности постов, которые составляют весь набор данных.

Строки 84–90

Функция `find_top_posts()` представляет пользовательский интерфейс, который напоминает пользователю о необходимости указать условие поиска, а затем с помощью этого запроса вызывает функцию `top_posts()`. По умолчанию условие поиска представляет собой строку `'python'`, если пользователь не задал его в явном виде. Функция `py_top_posts()` представляет собой пользовательскую модификацию, которая вызывает функцию `find_top_posts()` непосредственно с помощью поискового условия `python`.

Строки 92–103

Функция `find_user()` работает аналогично функции `find_top_posts()` за исключением того, что ее задача состоит в получении пользовательского идентификатора и вызове функции `get_user()`, которая выполняет свою работу. Она также проверяет, ввел ли пользователь числовой идентификатор, и отображает результаты на экране.

Строки 105–124

Последний блок этого кода содержит функцию `_main()`, которая отображает меню опций для пользователя. Сначала она выполняет синтаксический анализ меню, а затем помещает вокруг него шаблонный текст. Пользователь получает напоминание о сделанном им выборе, который реализуется после его проверки на достоверность. В противном случае выполняется выход из сценария (по умолчанию).

На момент написания этой книги служба Google+ все еще остается сравнительно новой системой, и к тому времени, когда эта книга окажется в руках читателя, нужно быть готовым к появлению изменений. Подобно предыдущему разделу этой главы, мы лишь в самых общих чертах обрисовали потенциал платформы Google+ в целом, а также ее интерфейса прикладного программирования. Как и в случае среды Jython, надеемся, что этот фрагмент диалога, а также приведенные нами примеры кода дадут вам правильное представление об этих технологиях и их истинных возможностях. В будущих изданиях этой книги я надеюсь существенно дополнить оба эти раздела.

15.3. Упражнения

Java, Python, Jython

- 15.1. *Jython*. В чем разница между средами Jython и CPython?
- 15.2. *Java и Python*. Возьмите какое-либо уже существующее приложение на языке Java и перенесите его в среду Python. Опишите свой опыт в каком-либо журнале. Выполнив эту часть работы, составьте пояснительную записку о том, какой результат должен быть достигнут, в чем заключаются некоторые из важнейших шагов и какие типичные операции вы должны выполнить, чтобы достигнуть поставленной цели.
- 15.3. *Java и Python*. Изучите исходный код интерпретатора Jython. Опишите, как некоторые стандартные типы языка Python реализованы в языке Java.
- 15.4. *Java и Python*. Попытайтесь расширить язык Python, написав это расширение на языке Java. Какие шаги необходимо выполнить для этого? Продемонстрируйте свое рабочее решение, показав, как оно работает, воспользовавшись интерактивным интерпретатором Jython.
- 15.5. *Jython и базы данных*. Найдите какое-либо интересное упражнение в главе 6 и перенесите его в среду Jython. Одно из лучших качеств среды Jython заключается в том, что с версии 2.1 она комплектуется модулем баз данных JDBC, который называется `zxJDBC`. Этот модуль почти совместим с версией 2.0 DB-API Python.
- 15.6. *Python и Jython*. Найдите какой-либо модуль среды Python, еще не реализованный на языке Jython, и перенесите его. Рассмотрите возможность применения его как “заплаты” к дистрибутиву среды Jython.

Google+

- 15.7. *Количество результатов*. В сценарии `plus_top_posts.py` мы ограничили количество отображаемых результатов до “top 5”, хотя очевидно, что данный код поддерживает гораздо большее их количество. Добавьте в меню еще один пункт, который позволял бы пользователям выбирать возвращаемое количество результатов (до какого-либо разумного числа).
- 15.8. *Рассматриваемый отрезок времени*. В функции `top_posts()` используется переменная `ndays`, которая задает отрезок времени, на котором отыскиваются самые популярные посты (по умолчанию этот отрезок времени равняется семи последним дням). Попытайтесь повысить гибкость данного сценария, обеспечив поддержку переменной величины этого отрезка времени (чтобы можно было указать любое количество дней).
- 15.9. *Объектно-ориентированное программирование и глобальные переменные*. В сценарии `plus_top_posts.py` мы сосредоточили всю базовую функциональность в классе `PlusService`. Весь код, который обеспечивает взаимодействие с пользователем (вызовы функции `raw_input()` и выполнение оператора `print`), сосредоточен в функциях за пределами этого класса.
 - а) Это заставило нас сделать `service` глобальной переменной. Почему это по-прежнему плохо?

- б) Измените этот код таким образом, чтобы `service` больше не использовалась в качестве доступа глобальной переменной из этих внешних функций. Можно рассмотреть следующий вариант: объедините внешние функции как методы `PlusService` (включая `_main()`), передавайте его как локальную переменную и т.д.

15.10. *Python 3.* Библиотека `Client Library` интерфейса `API Google` для среды `Python` еще не реализована в версии `Python 3`, но с учетом того, что раньше или позже это обязательно будет сделано, перенесите сценарий `plus_top_posts.py` в некий эквивалент для версии `Python 3` или создайте гибрид, который выполнялся бы как в среде `Python 2`, так и в `Python 3`. Вы сможете протестировать его, как только клиентская библиотека интерфейса `API Google` для среды `Python` будет реализована для версии `Python 3`, но в любом случае это полезно сделать уже сейчас.

15.11. *Индикатор выполнения хода задачи.* Несмотря на очевидную полезность сообщения `"Searching for..."` ("Выполняется поиск..."), призывающего пользователей немного потерпеть, пока будут готовы результаты, было бы еще полезнее, если бы пользователи получали более подробную информацию о ходе этого процесса. С этой целью добавьте импорт модуля `sys` в сценарий `plus_top_posts.py`. Затем добавьте в метод `PlusService.get_posts()` вызов, который записывает на экран одну точку, — сразу же после того, как новый объект класса `PlusPost` присоединяется к полученным результатам.

Мы рекомендуем воспользоваться потоком `stderr` вместо `stdout`, поскольку первый не буферизуется и сразу же появится на экране. Когда вы будете выполнять еще один поиск, вы получите более полное представление о промежуточной работе, которая требуется для того, чтобы скомпоновать результаты еще до того, как вы увидите их на экране.

15.12. *Google+.* Пользователи могут отвечать на посты собственными комментариями. В сценарии `plus_top_posts.py` мы лишь перечислили эти посты. Дополните эту функциональную возможность таким образом, чтобы можно было также показывать комментарии к каждому посту. Более подробную информацию можно получить на сайте <https://developers.google.com/+/api/latest/comments>.

15.13. *Авторизация.* Ни один из вариантов поиска, выполненных в сценарии `plus_top_posts.py`, не был авторизованным; это означает, что данное приложение может выполнять поиск только в общедоступных данных. Добавьте поддержку службы `OAuth2` таким образом, чтобы можно было получать доступ к пользовательским и частным данным. Более подробную информацию можно получить на сайтах <https://developers.google.com/+/api/oauth> и <http://code.google.com/p/google-api-python-client/wiki/OAuth2Client>.

15.14. *Точность.* В коде поиска наш сценарий `plus_top_posts.py`, точнее говоря, функция `get_posts()`, отфильтровывал содержимое, не представляющее для нас интереса. При этом мы гарантировали, что условие поиска появится в заголовке соответствующего поста. Это был несколько примитивный способ, поскольку заголовок является лишь частью всего сообщения. Попробуйте повысить точность нашей фильтрации, проверяя также, содержится ли указанное нами условие поиска и в самом *содержимом*, и если какой-либо

конкретный пост отвечает этим двум требованиям, то мы должны сохранить его. Если хотите узнать, как обратиться к содержимому, обратите внимание на инициализатор для объектов `PlusPost`. Попробуйте минимизировать дублирование кода, где это возможно. Плюс один балл: проверяйте на условие поиска также содержимое присоединений.

15.15. *Время против релевантности.* Порядок выполнения поиска, предусмотренный в службе Google+ по умолчанию, таков, что первыми проверяются самые последние по времени посты. В вызове метода `self.service.activities().search()` присутствует подразумеваемый параметр `orderBy = 'recent'`. Более подробные сведения о параметре `orderBy` можно найти в документации разработчика на сайте <https://developers.google.com/+api/latest/activities/search>.

a) Этот порядок выполнения поиска можно изменить с самых последних на самые релевантные посты, указав `'best'` вместо `'recent'`. Внесите такое изменение.

б) Как это повлияет на код в методе `get_posts()`, из которого мы выходим, как только достигнем поста, более старого, чем заданный нами порог?

в) Что нужно сделать, чтобы решить указанную проблему (если вообще нужно что-либо делать)? В случае необходимости внесите требуемое изменение.

15.16. *Поиск людей.* Добавьте поддержку поиска людей посредством интерфейса API Google+, создав функцию `find_people()`. Текущий поиск действий использует метод `self.service.activities().search()`. Для поиска пользователей вы должны воспользоваться соответствующим эквивалентом, т.е. `self.service.people().search()`. Более подробную информацию можно получить на сайте <http://developers.google.com/+api/latest/people/search>.

15.17. *Присоединения.* Наш сценарий `plus_top_posts.py` не выполняет каких-либо действий в отношении присоединений, в которых подчас бывает заключена основная причина постов. В примере выполнения этого сценария, показывающем пять верхних постов, касающихся Python (на момент написания этой книги), по меньшей мере в одном имелась ссылка на соответствующую веб-страницу и еще в одном — фотография. Добавьте в свою версию этого сценария поддержку для присоединений.

15.18. *Аргументы командной строки.* Версия сценария `plus_top_posts.py`, приведенная в этой главе, является программой командной строки; управление этой программой осуществляется с помощью меню. Однако на практике вы могли бы отдать предпочтение неинтерактивному интерфейсу, особенно для написания сценариев, крон-заданий и т.п. Усовершенствуйте это приложение, включив в него интерфейс синтаксического анализа аргументов командной строки (и эквивалентную функциональность). Рекомендуем воспользоваться `argparse`¹; впрочем, будет достаточно `optparse`² или даже старого модуля `getopt`.

¹ Новый в Python 2.7.

² Новый в Python 2.3.

- 15.19.** *Веб-программирование.* Сценарий `plus_top_posts.py` хорошо функционирует как инструмент командной строки, однако время от времени вы можете отлучаться от терминала; возможно, вы окажетесь в месте, где доступ онлайн может быть единственным вариантом в вашем распоряжении. Разработайте версию этого инструмента в виде совершенно отдельного веб-приложения. Можете использовать любые инструменты, имеющиеся в вашем распоряжении.

Ответы на некоторые упражнения

*Ответ на Великий вопрос... Жизни, Вселенной и Всего остального...
Это... "Сорок два", — изрек глубокомысленный компьютер
с бесконечным величием и спокойствием.
Дуглас Адамс, октябрь 1979 г.
(из книги *The Hitchhiker's Guide to the Galaxy*, 1979, Pan Books)*

Глава 1

Регулярные выражения

1.1. Распознаны строки

bat, hat, bit и т.д.

[bh] [aiu]t

1.2. Имя и фамилия

A-Za-z-]+ [A-Za-z-]+

(Любая пара слов, разделенных одним пробелом, т.е. имя и фамилия; допускается использование дефисов.)

1.3. Фамилия и первая буква имени

[A-Za-z-]+, [A-Za-z-]

(Любое слово и буква, разделенные запятой и одиночным пробелом, как в сочетании фамилии и первая буква имени.)

$[A-Za-z-]^+$, $[A-Za-z-]^+$

(Любая пара слов, разделенная запятой и одиночным пробелом, как в сочетании фамилии и имени; допускается использование дефисов.)

1.8. Длинные целые числа, поддерживаемые языком Python

`\d+[1L]`

(Только десятичные [основание 10] целые числа)

1.9. Числа с плавающей точкой, поддерживаемые языком Python

`[0-9]+(\.[0-9]*)?`

(Описывает простое число с плавающей точкой, т.е. любое количество цифр, за которыми следует [необязательно] одна десятичная точка и нуль или другие десятичные цифры, например "0.004", "2", "75." и т.п.)

Глава 2

2.3. Сокеты

TCP

2.6. Служба дневного времени

```
>>> import socket
>>> socket.getservbyname('daytime', 'udp')
13
```

Глава 3

3.20. Идентификаторы

pass — это ключевое слово, поэтому его нельзя использовать в качестве идентификатора. Типичный прием во всех подобных случаях заключается в том, чтобы присоединить к имени недопустимой переменной символ подчеркивания (`_`).

Глава 4

4.2. Поток Python

Ограничиваемые пропускной способностью ввода-вывода... Почему?

Глава 5

5.1. Архитектура "клиент-сервер"

Клиенты системы окон — это события GUI, обычно генерируемые пользователями; эти события должны обрабатываться системой окон, которая действует как сервер. Она отвечает за своевременное внесение обновлений в отображение, чтобы они были очевидны для пользователя.

Глава 6

6.1. Расширение Python

- Повышение производительности
- Защита исходного кода
- Новое или желаемое изменение функциональности
- И другое!

Глава 7

7.16. Улучшение обозначения для слайдов с названием

Основной проблемой в нашем коде является то, что он обращается к `str.title()` как для получения названия диалога, так и для названий отдельных слайдов. Строку 43, действительно, необходимо улучшить:

```
s.Shapes[0].TextFrame.TextRange.Text = line.title()
```

Мы можем внести простое изменение, чтобы наш код применял регистр названия ко всему названию (в данном случае это означает, что все название будет отображаться прописными буквами), оставляя неизменными заголовки в слайдах без названий:

```
s.Shapes[0].TextFrame.TextRange.Text = title и  
line.title() или line
```

Однако можно поступить лучше. В этом упражнении спрашивается, что делать с ТСП/ІР и как избежать его преобразования в строку “Тсп/Ір”. Допустим, мы создали новую переменную `eachWord`. Я предлагаю в этом случае проверить, выполняется ли условие `eachWord == eachWord.upper()`. Если это аббревиатура, то она остается неизменной; в противном случае мы можем применить регистр названия. Конечно, возможны исключения, но если наш код учитывает хотя бы 80% возможных случаев, то для начала это не так уж плохо.

Глава 8

8.1. DB-API

DB-API — это общепринятая спецификация интерфейса для всех адаптеров баз данных Python. Это хорошо в том отношении, что заставляет всех разработчиков адаптеров кодировать в соответствии с одной и той же спецификацией, чтобы программисты конечного пользователя писали совместимый код, который можно было бы без особого труда переносить на другие базы данных.

Глава 10

10.6. Ошибки CGI

Веб-сервер либо не возвращает вообще никаких данных, либо возвращает текст ошибки, что приводит к HTTP 500 или внутренней ошибке сервера (Internal Server Error) в вашем браузере, поскольку это (возвращаемые данные) не является допустимыми данными HTTP или HTML. Модуль `cgitb` отслеживает последовательность действий Python и возвращает ее посредством CGI как допустимые данные. Эти данные отображаются пользователю, что может служить великолепным инструментом отладки.

Глава 13

13.8. Веб-службы и модуль csv

Замените цикл `for` в `stock.py` следующим фрагментом кода:

```
import csv  
for tick, price, chg, per in csv.reader(f):  
    print tick.ljust(7), ('%.2f' % round(float(price),  
        2)).rjust(6), chg.rjust(6), per.rjust(6)
```

Глава 14

14.2. CSV против `str.split()`

Очевидно, запятая, действительно, является не лучшим вариантом разделителя при выполнении синтаксического анализа данных, в которых используется какой-либо другой разделитель (это не нуждается в доказательствах). Помимо этого, использование запятых также включает в себе опасность, если соответствующие поля (значения индивидуальных столбцов) могут содержать кавычки. Дополнительные осложнения могут также возникнуть, если поля могут содержать кавычки. Это может случиться не только потому, что запятые могут попасть в строковые значения, но и при выполнении синтаксического анализа текста, в котором запятые могут появляться либо слева, либо справа от кавычки.

Сами по себе кавычки также могут создавать проблемы: как вы будете выполнять синтаксический анализ строки, которая содержит строки, заключенные в кавычки, если хотите, чтобы все слова этой строки с кавычками рассматривались как одно целое? (Подсказка: см. сайт <http://docs.python.org/library/shlex>.)

14.11. Устойчивость и надежность

В сценарии `xmlrpcsrvr.py` измените строку 13:

```
FUNCS = ('add', 'sub', 'mul', 'div', 'mod')
```

добавив все требуемые функции:

```
FUNCS = ('add', 'sub', 'mul', 'div', 'mod',
        'gt', 'ge', 'lt', 'le', 'eq', 'ne',
        'truediv', 'floordiv',
        )
```

Все они присутствуют в модуле `operator`, поэтому никакой дополнительной работы, помимо указанного изменения, не потребуется. Сейчас вы располагаете достаточными знаниями, чтобы суметь добавить унарные `-`, `**` и побитовые операторы `&`, `|`, `^` и `~`.

Глава 15

15.1. Jython

Jython — это Java-реализация стандартного интерпретатора Python (большей его части), которая написана на языке C; отсюда ее название — CPython. Она является байт-компилированной, что позволяет выполнять ее на виртуальной машине Java (Java Virtual Machine — JVM). Вместо того чтобы прибегнуть к непосредственному переносу, создатель Jython признал наличие у Java собственного управления памятью и инфраструктур обработки исключений (а такие особенности языка не требуют переноса). Версии Jython нумеруются в соответствии с конкретной совместимостью, т.е. версия Jython 2.5 совместима с CPython 2.5. Оригинальная версия Jython получила название JPython, но впоследствии ее переименовали в Jython. Более подробную информацию о Jython можно получить на сайте <http://wiki.python.org/jython/JythonFaq/GeneralInfo>, в разделе онлайн-ответов на часто задаваемые вопросы (FAQ) по Jython.

Справочные таблицы

У кого-нибудь еще из этого списка есть свое мнение?

Должен ли я перейти на другой язык?

Гвидо ван Россум, декабрь 1991 г.

Ключевые слова Python

В табл. Б.1 перечислены ключевые слова Python.

Таблица Б.1. Зарезервированные слова Pythona

and	as ^b	assert ^c	break
class	continue	def	del
elif	else	except	exec ^d
finally	for	from	global
if	import	in	is
lambda	nonlocal ^e	not	or
pass	printd	raise	return
try	while	withb	yieldf

^a Ключевое слово **None** стало константой в Python 2.4; **None**, **True**, **False** стали ключевыми словами в версии 3.0.

^b Новое в Python 2.6.

^c Новое в Python 1.5.

^d Стало встроенной функцией и перестало быть ключевым словом в Python 3.0.

^e Новое в Python 3.0.

^f Новое в Python 2.3.

Стандартные операторы и функции Python

В табл. Б.2 представлены операторы и функции (встроенные и “заводские”), которые могут быть использованы с большинством стандартных объектов Python, а также объектов, определенных пользователем, в которых вы реализовали соответствующие им специальные методы.

Таблица Б.2. Операторы и функции стандартного типа

Оператор/ функция	Описание	Результат ^a
Представление строк		
<code>r b</code>	Строковое представление, которое можно оценить	str
Встроенные и “заводские” функции		
<code>cmp(obj1, obj2)</code>	Сравнивает два объекта	int
<code>repr(obj)</code>	Строковое представление, которое можно оценить	str
<code>str(obj)</code>	Строковое представление, которое можно распечатать	str
<code>type(obj)</code>	Тип объекта	type
Сравнения значений		
<code><</code>	Меньше, чем	bool
<code>></code>	Больше, чем	bool
<code><=</code>	Не больше	bool
<code>>=</code>	Не меньше	bool
<code>==</code>	Равно	bool
<code>!=</code>	Не равно	bool
<code><>^b</code>	Не равно	bool
Сравнение объектов		
<code>is</code>	То же самое, что и...	bool
<code>is not</code>	Не то же самое, что и...	bool
Булевы операторы		
<code>not</code>	Логическое отрицание	bool
<code>and</code>	Логическое умножение	bool
<code>or</code>	Логическое сложение	bool

^a Булевы сравнения возвращают либо **True**, либо **False**.

^b Перестала использоваться в Python 3.0; вместо нее используется `!=`.

Операторы и функции численного типа

В табл. Б.3 представлены операторы и функции (встроенные и “заводские”), которые применяются к численным объектам Python.

Таблица Б.3. Операторы и встроенные функции для всех численных типов

Оператор/ встроенная функция	Описание	int	long ^a	float	complex	Результат ^a
<code>abs()</code>	Абсолютное значение	•	•	•	•	число
<code>bin()</code>	Двоичная строка	•	•	•	•	str
<code>chr()</code>	Символ	•	•			str
<code>coerce()</code>	Численное приведение типа	•	•	•	•	tuple

Окончание табл. Б.3

Оператор/ встроенная функция	Описание	int	long ^a	float	complex	Результат ^a
<code>complex()</code>	Комплексная “заводская” функция	•	•	•	•	complex
<code>divmod()</code>	Деление по модулю	•	•	•	•	tuple
<code>float()</code>	“Заводская” функция с плавающей запятой	•	•	•	•	float
<code>hex()</code>	Шестнадцатеричная строка	•	•			str
<code>int()</code>	Целочисленная “заводская” функция	•	•	•	•	int
<code>long()</code> ^a	“Заводская” функция с длинными числами	•	•	•	•	long
<code>oct()</code>	Восьмеричная строка	•	•			str
<code>ord()</code>	Перечислимый тип (строка)					int
<code>pow()</code>	Возведение в степень	•	•	•	•	number
<code>round()</code>	Округление с плавающей запятой			•		float
<code>sum()</code> ^c	Суммирование	•	•	•		float
<code>**d</code>	Возведение в степень	•	•	•	•	number
<code>+e</code>	Без изменений	•	•	•	•	number
<code>-d</code>	Отрицание	•	•	•	•	number
<code>~d</code>	Двоичная инверсия	•	•			int/long
<code>**c</code>	Возведение в степень	•	•	•	•	number
<code>*</code>	Умножение	•	•	•	•	number
<code>/</code>	Классическое, или истинное, деление	•	•	•	•	number
<code>//</code>	Деление с округлением результата	•	•	•	•	number
<code>%</code>	По модулю/остаток	•	•	•	•	number
<code>+</code>	Сложение	•	•	•	•	number
<code>-</code>	Вычитание	•	•	•	•	number
<code><<</code>	Сдвиг на один двоичный разряд влево	•	•			int/long
<code>>></code>	Сдвиг на один двоичный разряд вправо	•	•			int/long
<code>&</code>	Поразрядное И	•	•			int/long
<code>^</code>	Поразрядное исключающее ИЛИ	•	•			int/long
<code> </code>	Поразрядное ИЛИ	•	•			int/long

^a Тип long перестал использоваться в Python 3.0; вместо него следует использовать int.

^b Результат “число” указывает любой из численных типов; возможно, такой же, как у оператора.

^c Новый в Python 2.3.

^d ** Относится только к унарным операторам.

^e Унарный оператор.

Операторы и функции типа “последовательность”

В табл. Б.4 приведена совокупность операторов, функций (встроенных и “заводских”), а также встроенных методов, которые могут быть использованы с последовательными типами.

Таблица Б.4. Операторы, функции и встроенные методы последовательного типа

Оператор, встроенная функция или метод	str	list	tuple
[] (создание списка)		•	
()			•
"""	•		
append()		•	
capitalize()	•		
center()	•		
chr()	•		
cmp()	•	•	•
count()	•	•	• ^a
decode()	•		
encode()	•		
endswith()	•		
expandtabs()	•		
extend()		•	
find()	•		
format()	• ^a		
hex()	•		
index()	•	•	• ^a
insert()		•	
isalnum()	•		
isalpha()	•		
isdecimal()	• ^b		
isdigit()	•		
islower()	•		
isnumeric()	• ^b		
isspace()	•		
istitle()	•		
isupper()	•		
join()	•		
len()	•	•	•
list()	•	•	•
ljust()	•		
lower()	•		
lstrip()	•		
max()	•	•	•
min()	•	•	•

Продолжение табл. Б.4

Оператор, встроенная функция или метод	str	list	tuple
ord()	•		
partition()	• ^c		
pop()		•	
raw_input()	•		
remove()		•	
replace()	•		
repr()	•	•	•
reverse()		•	
rfind()	•		
rindex()	•		
rjust()	•		
rpartition()	• ^c		
rsplit()	• ^d		
rstrip()	•		
sort()		•	
split()	•		
splitlines()	•		
startswith()	•		
str()	•	•	•
strip()	•		
swapcase()	•		
title()	•		
translate()	•		
tuple()	•	•	•
type()	•	•	•
upper()	•		
zfill()	• ^e		
• (атрибуты)	•	•	
[] (срез)	•	•	•
[:]	•	•	•
*	•	•	•
%	•		
+	•	•	•
in	•	•	•
not in	•	•	•

^a Новое в Python 2.6 (появление первых методов для tuples).^b Только для строк Unicode в Python 2.x; новое в Python 3.0.^c Новое в Python 2.5.^d Новое в Python 2.4.^e Новое в Python 2.2.2.

Символы преобразования оператора строкового формата

В табл. Б.5 перечислены символы форматирования, которые можно использовать с оператором строкового формата (%).

Таблица Б.5. Символы преобразования оператора строкового формата

Символ формата	Преобразование
%c	Символ (целое число [ASCII-значение] или строка длиной 1)
%r ^a	Преобразование строки посредством repr() до форматирования
%s	Преобразование строки посредством str() до форматирования
%d / %i	Десятичное целое число со знаком
%u ^b	Десятичное целое число без знака
%o ^b	Восьмеричное целое число (без знака)
%x ^b / %X ^b	Шестнадцатеричное целое число (без знака) (буквы нижнего/ВЕРХНЕГО регистра)
%e / %E	Экспоненциальная система обозначения (с помощью строчной буквы e/ПРОПИСНОЙ буквы E)
%f / %F	Действительное [вещественное] число с плавающей точкой (дробная часть отбрасывается естественным образом)
%g / %G	Самое короткое среди %e, %f/%E% и %F%
%%	Символ процента (%) unescaped

^a Новый в Python 2.0; по-видимому, уникален только для Python.

^b В Python 2.4+ %u/%o/%x/%X отрицательного целого числа вернет строку со знаком.

Директивы оператора строкового формата

При использовании оператора строкового формата (см. табл. Б.5) вы можете улучшить или точно настроить отображение соответствующего объекта с помощью директив, представленных в табл. Б.6.

Таблица Б.6. Вспомогательные директивы оператора формата

Символ	Функциональность
*	Аргумент указывает ширину или точность
-	Используйте выравнивание по левому краю
+	Используйте знак "плюс" (+) для положительных чисел
<sp>	Используйте заполнение пробелами для положительных чисел
#	Добавляйте восьмеричный ведущий ноль (0), или шестнадцатеричный ведущий 0x, или 0X в зависимости от того, использовался ли x или X
0	Используйте заполнение нулями (вместо пробелов) при форматировании чисел
%	%% оставляет одиночный литеральный %
(var)	Переменная отображения (аргументы словаря)
m.n	m — минимальная совокупная ширина; n — количество отображаемых разрядов после десятичной точки (если таковые имеются)

Встроенные методы строкового типа

Описания строковых встроенных методов из табл. Б.4 приведены в табл. Б.7.

Таблица Б.7. Встроенные методы строкового типа

Метод	Описание
<code>строка.capitalize()</code>	Преобразует первую букву <i>строки</i> в прописную
<code>строка.center(ширина)</code>	Возвращает заполненную пробелами <i>строку</i> , причем исходная <i>строка</i> выровнена по центру относительно суммарной <i>ширины</i> столбцов
<code>строка.count(str, beg=0, end=len(строка))</code>	Подсчитывает, сколько раз <i>str</i> встречается в <i>строке</i> или в какой-либо подстроке <i>строки</i> , если заданы начальный индекс <i>beg</i> и конечный индекс <i>end</i>
<code>строка.decode(encoding='UTF-8', errors='strict')^e</code>	Возвращает декодированную строковую версию <i>строки</i> ; в случае ошибки возвращается (по умолчанию) <code>ValueError</code> , если в качестве <i>errors</i> не задано <code>ignore</code> или <code>replace</code>
<code>строка.encode(encoding='UTF-8', errors='strict')^a</code>	Возвращает кодированную строковую версию <i>строки</i> ; в случае ошибки возвращается (по умолчанию) <code>ValueError</code> , если в качестве <i>errors</i> не задано <code>ignore</code> или <code>replace</code>
<code>строка.endswith(str, beg=0, end=len(строка))^b</code>	Определяется, если <i>строка</i> или какая-либо подстрока <i>строки</i> (если заданы начальный индекс <i>beg</i> и конечный индекс <i>end</i>) заканчивается на <i>str</i> ; в этом случае возвращает <code>True</code> , в противном случае — <code>False</code>
<code>строка.expandtabs(tabsize=8)</code>	Расширяет табуляции в <i>строке</i> до нескольких пробелов; по умолчанию, если <i>tabsize</i> не указан, равняется восьми пробелам на каждую табуляцию
<code>строка.find(str, beg=0, end=len(строка))</code>	Определяется, если <i>str</i> встречается в <i>строке</i> или в какой-либо подстроке <i>строки</i> , если заданы начальный индекс <i>beg</i> и конечный индекс <i>end</i> ; возвращает индекс, если результат поиска оказался успешным; в противном случае возвращает <code>-1</code>
<code>строка.format(*args, **kwargs)</code>	Выполняет форматирование строки, основываясь на передаваемых <i>args</i> и/или <i>kwargs</i>
<code>строка.index(str, beg=0, end=len(строка))</code>	То же, что и <code>find()</code> , но возвращает исключение, если <i>str</i> не найдена
<code>строка.isalnum()^{a,b,c}</code>	Возвращает <code>True</code> , если <i>строка</i> содержит по меньшей мере 1 символ, а все символы — алфавитно-цифровые; в противном случае возвращает <code>False</code>
<code>строка.isalpha()^{a,b,c}</code>	Возвращает <code>True</code> , если <i>строка</i> содержит по меньшей мере 1 символ, а все символы — алфавитные; в противном случае возвращает <code>False</code>
<code>строка.isdecimal()^{a,b,d}</code>	Возвращает <code>True</code> , если <i>строка</i> содержит только десятичные цифры; в противном случае возвращает <code>False</code>
<code>строка.isdigit()^{b,c}</code>	Возвращает <code>True</code> , если <i>строка</i> содержит только цифры; в противном случае возвращает <code>False</code>
<code>строка.islower()^{b,c}</code>	Возвращает <code>True</code> , если <i>строка</i> содержит по меньшей мере 1 символ, а все символы представлены в нижнем регистре; в противном случае возвращает <code>False</code>
<code>строка.isnumeric()^{b,c,d}</code>	Возвращает <code>True</code> , если <i>строка</i> содержит только цифровые символы; в противном случае возвращает <code>False</code>

Метод	Описание
<code>строка.isspace()</code> ^{bc}	Возвращает True, если строка содержит только символы пробела; в противном случае возвращает False
<code>строка.istitle()</code> ^{bc}	Возвращает True, если символы в строке соответствуют правилам оформления заголовка (см. <code>title()</code>); в противном случае возвращает False
<code>строка.isupper()</code> ^{bc}	Возвращает True, если строка содержит по меньшей мере один символ, а все символы представлены в верхнем регистре; в противном случае возвращает False
<code>строка.join(seq)</code>	Объединяет (конкатенирует) строковые представления элементов в последовательности <code>seq</code> в строку; в качестве разделителя используется строка
<code>строка.ljust(width)</code>	Возвращает заполненную пробелами строку, причем исходная строка выровнена по левому краю в соответствии с общим количеством столбцов <code>width</code>
<code>строка.lower()</code>	Преобразует все буквы верхнего регистра в строке в нижний регистр
<code>строка.lstrip()</code>	Удаляет все ведущие пробелы в строке
<code>строка.replace(str1, str2, num=строка.count(str1))</code>	Заменяет все <code>str1</code> в строке на <code>str2</code> или заменяет не менее чем <code>num</code> случаев появления <code>str1</code> , если задано <code>num</code>
<code>строка.rfind(str, beg=0, end=len(строка))</code>	То же самое, что <code>find()</code> , но поиск в строке ведется в обратном направлении
<code>строка.rindex(str, beg=0, end=len(строка))</code>	То же самое, что <code>index()</code> , но поиск в строке ведется в обратном направлении
<code>строка.rjust(width)</code>	Возвращает заполненную пробелами строку, причем исходная строка выровнена по правому краю в соответствии с общим количеством столбцов <code>width</code>
<code>строка.rstrip()</code>	Удаляет все концевые пробелы в строке
<code>строка.split(str="", num=строка.count(str))</code>	Разделяет строку в соответствии с разделителем <code>str</code> (если не задан явно, используется пробел) и возвращает список подстрок; разделение на самое большее <code>num</code> подстрок (если задано значение <code>num</code>)
<code>строка.splitlines(num=строка.count('\n'))^{bc}</code>	Разделяет строку по всем (или <code>num</code>) токенам NEWLINE и возвращает список каждой строки с удаленными токенами NEWLINE
<code>строка.startswith(str, beg=0, end=len(строка))^b</code>	Определяет, начинается ли строка или какая-либо подстрока строки (если заданы начальный индекс <code>beg</code> и конечный индекс <code>end</code>) с подстроки <code>str</code> ; возвращает True, если результат этой проверки оказывается положительным, и False — в противном случае
<code>строка.strip([obj])</code>	Выполняет <code>lstrip()</code> и <code>rstrip()</code> по отношению к строке
<code>строка.swapcase()</code>	Инвертирует регистр для всех букв в строке
<code>строка.title()</code> ^{bc}	Возвращает версию строки, которая соответствует правилам оформления заголовка, т.е. все слова начинаются с прописной буквы, а остальные буквы каждого слова — строчные (см. также <code>istitle()</code>)
<code>строка.translate(str, del=" ")</code>	Транслирует строку в соответствии с таблицей трансляции <code>str</code> (256 символов), удаляя символы из строки <code>del</code>
<code>строка.upper()</code>	Преобразует все буквы нижнего регистра в строке в верхний регистр

Окончание табл. Б.7

Метод	Описание
<code>строка.zfill(width)</code>	Дополняет исходную строку ведущими нулями, чтобы суммарное количество символов составляло <i>width</i> . <code>zfill()</code> , предназначенная для чисел, сохраняет любой заданный знак (минус один ноль)

- ^a Применимо только к строкам Unicode в версии 1.6 и ко всем типам строк в версии 2.0.
- ^b Отсутствует как функция модуля `string` в версии 1.5.2.
- ^c Новый в Python 2.1.
- ^d Применимо только к строкам Unicode.
- ^e Новый в Python 2.2.

Встроенные методы списочного типа

В табл. Б.8 представлены полные описания и синтаксис использования для встроенных методов списочного типа, приведенных в табл. Б.4.

Таблица Б.8. Встроенные методы списочного типа

Списочный метод	Операция
<code>список.append(obj)</code>	Добавляет <i>obj</i> в конец списка
<code>список.count(obj)</code>	Возвращает число, показывающее, сколько раз <i>obj</i> встречается в списке
<code>список.extend(seq)^a</code>	Присоединяет содержимое <i>seq</i> к списку
<code>список.index(obj, i=0, j=len(список))</code>	Возвращает наименьший индекс <i>k</i> , где <code>список[k] == obj</code> и $i \leq k < j$; в противном случае возвращается <code>ValueError</code>
<code>список.insert(index, obj)</code>	Вставляет <i>obj</i> в список со смещением <i>index</i>
<code>список.pop(index=-1)^a</code>	Удаляет и возвращает <i>obj</i> при заданном или последнем <i>index</i> из списка
<code>список.remove(obj)</code>	Удаляет объект <i>obj</i> из списка
<code>список.reverse()</code>	Переставляет в обратной последовательности объекты списка
<code>список.sort(функ=None, key=None, reverse=False)</code>	Сортирует элементы списка с необязательной функцией сравнения; <i>key</i> — это обратный вызов при извлечении элементов для сортировки, и если флаг <i>reverse</i> равен <code>True</code> , то список сортируется в обратной последовательности

- ^a Новый в Python 1.5.2.

Встроенные методы словарного типа

В табл. Б.9 представлены полные описания и синтаксис использования для встроенных методов словарного типа.

Таблица Б.9. Встроенные методы словарного типа

Метод	Операция
<code>словарь.clear^a()</code>	Удаляет все элементы словаря
<code>словарь.copy^a()</code>	Возвращает (поверхностную) копию словаря
<code>словарь.fromkeys^b(seq, val=None)</code>	Создает и возвращает новый словарь с элементами <i>seq</i> как ключами и <i>val</i> как начальным значением (если не задан, то по умолчанию считается равным <code>None</code>) для всех ключей

Метод	Операция
<code>словарь.get(key, default=None)</code> ^a	Для ключа <i>key</i> возвращает значение или <i>default</i> , если <i>key</i> нет в словаре (обратите внимание: значением <i>default</i> по умолчанию является <i>None</i>)
<code>словарь.has_key(key)</code> ^e	Возвращает <i>True</i> , если <i>key</i> находится в словаре; в противном случае — <i>False</i> . В версии 2.2 его роль частично снижена операторами <i>in</i> и <i>not in</i> , однако он все еще обеспечивает функциональный интерфейс
<code>словарь.items()</code> <code>словарь.iter*()</code>	Возвращает <i>iterable</i> кортежных пар (ключ, значение) словаря <i>iteritems()</i> , <i>iterkeys()</i> , <i>itervalues()</i> — это методы, которые ведут себя точно так же, как их неитераторные аналоги, но возвращают итератор, а не список
<code>словарь.keys()</code> <code>словарь.pop(key[, default])</code>	Возвращает <i>iterable</i> ключей словаря Подобен <i>get()</i> , но удаляет и возвращает <i>dict[key]</i> , если присутствует <i>key</i> , и порождает <i>KeyError</i> , если <i>key</i> нет в словаре, а <i>default</i> не задан
<code>словарь.setdefault(key, default=None)</code> ^d	Подобен <i>get()</i> , но устанавливает <i>dict[key]=default</i> , если <i>key</i> уже нет в словаре
<code>словарь.update(dict2)</code> ^a <code>словарь.values()</code>	Добавляет пары ключ-значение из <i>dict2</i> в словарь Возвращает <i>iterable</i> значений словаря

^a Новый в Python 1.5.^b Новый в Python 2.3.^c Новый в Python 2.2.^d Новый в Python 2.0.^e Его роль снижена в Python 2.2, а в Python 3.0 его вообще нет, вместо него следует использовать *in*.^f *Iterable* — это заданное представление, начиная с версии Python 3.0, и список во всех предыдущих версиях.

Операторы и встроенные функции типов множеств

В табл. Б.10 представлены разные операторы, функции (встроенные и “заводские”), а также встроенные методы, которые применимы к обоим типам множеств (*set* [изменяемые] и *frozenset* [неизменяемые]).

Таблица Б.10. Операторы, функции и встроенные методы типов множеств

Функция/ метод	Эквивалент оператора	Описание
Все типы множеств		
<code>len(s)</code> <code>set([obj])</code>		Мощность множества: количество элементов в <i>s</i> “Заводская” функция изменяемого множества. Если <i>obj</i> задан, он должен быть итерабельным. Новые элементы множества берутся из <i>obj</i> ; если <i>obj</i> не задан, создается пустое множество
<code>frozenset([obj])</code>		“Заводская” функция неизменяемого множества. Действует так же, как <i>set()</i> , но возвращает неизменяемое множество

Окончание табл. Б.10

Функция/ метод	Эквивалент оператора	Описание
	<code>obj in s</code>	Тест на принадлежность к множеству: является ли <code>obj</code> элементом <code>s</code> ?
	<code>obj not in s</code>	Тест на непринадлежность к множеству: <code>obj</code> не является элементом <code>s</code> ?
	<code>s == t</code>	Тест на равенство: содержит ли <code>s</code> и <code>t</code> в точности одни и те же элементы?
	<code>s != t</code>	Тест на равенство: противоположность <code>==</code>
	<code>s < t</code>	(Строгий) тест подмножества; <code>s != t</code> и все элементы <code>s</code> являются членами <code>t</code>
<code>s.issubset(t)</code>	<code>s <= t</code>	Тест подмножества (допускает неточные подмножества): все элементы <code>s</code> являются членами <code>t</code>
	<code>s > t</code>	(Строгий) тест надмножества; <code>s != t</code> и все элементы <code>t</code> являются членами <code>s</code>
<code>s.issuperset(t)</code>	<code>s >= t</code>	Тест надмножества (допускает неточные надмножества): все элементы <code>t</code> являются членами <code>s</code>
<code>s.union(t)</code>	<code>s t</code>	Операция объединения: элементы в <code>s</code> или <code>t</code>
<code>s.intersection(t)</code>	<code>s & t</code>	Операция пересечения: элементы в <code>s</code> и <code>t</code>
<code>s.difference(t)</code>	<code>s - t</code>	Операция разности: элементы в <code>s</code> , которые не являются элементами <code>t</code>
<code>s.symmetric_difference(t)</code>	<code>s ^ t</code>	Операция симметричной разности: элементы либо <code>s</code> , либо <code>t</code> , но не того и другого
<code>s.copy()</code>		Операция копирования: вернуть (поверхностную) копию <code>s</code>
Только изменяемые множества		
<code>s.update(t)</code>	<code>s = t</code>	Операция обновления (объединения): члены <code>t</code> добавляются к <code>s</code>
<code>s.intersection_update(t)</code>	<code>s &= t</code>	Операция обновления пересечения: <code>s</code> содержит лишь члены исходного <code>s</code> и <code>t</code>
<code>s.difference_update(t)</code>	<code>s -= t</code>	Операция обновления разности: <code>s</code> содержит лишь исходные члены, которых нет в <code>t</code>
<code>s.symmetric_difference_update(t)</code>	<code>s ^= t</code>	Операция обновления симметричной разности: <code>s</code> содержит лишь члены <code>s</code> или <code>t</code> , но не того и другого
<code>s.add(obj)</code>		Операция добавления: добавить <code>obj</code> к <code>s</code>
<code>s.remove(obj)</code>		Операция удаления: удалить <code>obj</code> из <code>s</code> ; формируется <code>Key - Error</code> , если <code>obj</code> не в <code>s</code>
<code>s.discard(obj)</code>		Операция отбрасывания: более "дружественная к пользователю" версия <code>remove()</code> — удалить <code>obj</code> из <code>s</code> , если <code>obj</code> в <code>s</code>
<code>s.pop()</code>		Операция выталкивания: удалить и вернуть произвольный элемент <code>s</code>
<code>s.clear()</code>		Операция очистки: удалить все элементы <code>s</code>

Методы и атрибуты данных файловых объектов

В табл. Б.11 перечислены встроенные методы и атрибуты данных файловых объектов.

Таблица Б.11. Методы для файловых объектов

Атрибут файловых объектов	Описание
<code>файл.close()</code>	Закрывает <i>файл</i>
<code>файл.fileno()</code>	Возвращает целочисленный дескриптор файла (FD) для <i>файла</i>
<code>файл.flush()</code>	Очищает внутренний буфер для <i>файла</i>
<code>файл.isatty()</code>	Возвращает <code>True</code> , если <i>файл</i> представляет собой устройство типа <code>tty</code> (телетайп); в противном случае возвращает <code>False</code>
<code>файл.next^a()</code>	Возвращает следующую строку в файле [подобно <code>файл.readline()</code>] или формирует <code>StopIteration</code> , если строки исчерпались
<code>файл.read(size=-1)</code>	Считывает <i>size</i> байтов файла или все остающиеся байты, если не задан <i>size</i> или если <i>size</i> является отрицательной величиной, как строку и возвращает ее
<code>файл.readinto^b(buf, size)</code>	Считывает <i>size</i> байтов из <i>файла</i> в буфер <i>buf</i> (не поддерживается)
<code>файл.readline(size=-1)</code>	Считывает и возвращает одну строку из <i>файла</i> (включает символы завершения строки) — либо одну полную строку, либо максимум <i>size</i> символов
<code>файл.readlines(sizhint=0)</code>	Считывает и возвращает все строки из <i>файла</i> как список (включает символы завершения всех строк); если <i>sizhint</i> задан и больше нуля, возвращаются строки, состоящие примерно из <i>sizhint</i> байтов (может быть округлено до размера следующего буфера)
<code>файл.xreadlines^c()</code>	Предназначен для итерации; возвращает строки в <i>файле</i> , считываемые порциями более эффективным способом, чем это делает <code>readlines()</code>
<code>файл.seek(off, whence=0)</code>	Перемещается внутри <i>файла</i> в место, отстоящее на <i>off</i> байтов от <i>whence</i> (<code>0</code> == начало файла, <code>1</code> == текущее местоположение или <code>2</code> == конец файла)
<code>файл.tell()</code>	Возвращает текущее местоположение в <i>файле</i>
<code>файл.truncate(size= файл.tell())</code>	Урезает <i>файл</i> до самое большее <i>size</i> байтов; по умолчанию используется текущее местоположение в <i>файле</i>
<code>файл.write(str)</code>	Записывает строку <i>str</i> в <i>файл</i>
<code>файл.writelines(seq)</code>	Записывает <i>seq</i> строк в <i>файл</i> ; <i>seq</i> должно быть iterable, генерирующим строки; до появления версии 2.2 это было просто списком строк
<code>файл.closed</code>	<code>True</code> , если <i>файл</i> закрыт; в противном случае — <code>False</code>
<code>файл.encoding^d</code>	Кодирование, которое использует этот файл, — когда строки Unicode записываются в файл, они преобразуются в строки байтов с помощью <code>файл.encoding</code> ; значение <code>None</code> указывает, что для преобразования строк Unicode должно использоваться кодирование, предусмотренное в системе по умолчанию
<code>файл.mode</code>	Режим доступа, с которым был открыт <i>файл</i>
<code>файл.name</code>	Имя файла

Окончание табл. Б.11

Атрибут файловых объектов	Описание
<code>файл.newlines^d</code>	None, если не было считано никаких разделителей строк; строка, состоящая из одного типа разделителей строк; или кортеж, состоящий из всех типов символов завершения строки, считанных к этому моменту
<code>файл.softspace</code>	0, если пробел явно затребован с помощью <code>print</code> , 1 в противном случае. Редко используется программистами — в общем случае лишь для внутреннего использования

^a Новый в Python 2.2.^b Новый в Python 1.5.2, но не поддерживается.^c Новый в Python 2.1, но в Python 2.3 его роль снижена.^d Новый в Python 2.3.

Исключения в Python

В табл. Б.12 перечислены исключения в Python.

Таблица Б.12. Встроенные исключения в Python

Исключение	Описание
<code>BaseException^a</code>	Корневой класс для всех исключений
<code>SystemExit^b</code>	Завершение запроса интерпретатора Python
<code>KeyboardInterrupt^c</code>	Пользователь прервал выполнение (обычно путем нажатия <Ctrl+C>)
<code>Exception^d</code>	Корневой класс для регулярных исключений
<code>StopIteration</code>	У итерации больше нет значений
<code>GeneratorExit^a</code>	Исключение отправлено генератору, чтобы приказать ему завершить работу
<code>SystemExit^f</code>	Завершение запроса интерпретатора Python
<code>StandardError^d</code>	Базовый класс для всех стандартных встроенных исключений
<code>ArithmeticError^d</code>	Базовый класс для всех ошибок численных вычислений
<code>FloatingPointError^d</code>	Ошибка в вычислении с плавающей точкой
<code>OverflowError</code>	Вычисление превысило максимальное значение для численного типа
<code>ZeroDivisionError</code>	Ошибка деления (или модуля) на ноль (все численные типы)
<code>AssertionError^d</code>	Неудачное завершение оператора <code>assert</code>
<code>AttributeError</code>	Такого атрибута у объекта нет
<code>EOFError</code>	Достигнут маркер конца файла без ввода с помощью встроенной функции
<code>EnvironmentError</code>	Базовый класс для ошибок среды операционной системы
<code>IOError</code>	Неудачное завершение операции ввода-вывода
<code>OSError</code>	Ошибка операционной системы
<code>WindowsError</code>	Неудачная попытка вызова системы MS Windows
<code>ImportError</code>	Неудачная попытка импортировать модуль или объект
<code>KeyboardInterrupt^f</code>	Пользователь прервал выполнение (обычно путем нажатия <Ctrl+C>)

Исключение	Описание
LookupError ^d	Базовый класс для ошибок недопустимого поиска данных
IndexError	Такой индекс отсутствует в последовательности
KeyError	Такой ключ отсутствует в отображении
MemoryError	Ошибка выхода за пределы оперативной памяти (нефатальная для интерпретатора Python)
NameError	Необъявленный/неинициализированный объект (не атрибут)
UnboundLocalError	Доступ неинициализированной локальной переменной
ReferenceError	Слабая ссылка попыталась обратиться к объекту, оказавшемуся в собранном мусоре
RuntimeError	Групповая ошибка во время выполнения, выдаваемая по умолчанию
NotImplementedError	Нереализованный метод
SyntaxError	Ошибка в синтаксисе Python
IndentationError	Ошибка структурированного расположения текста
TabError ^g	Неправильное сочетание TAB и пробелов
SystemError	Групповая ошибка системы интерпретатора
TypeError	Недопустимая операция для типа
ValueError	Задан недопустимый аргумент
UnicodeError ^h	Ошибка, связанная с Unicode
UnicodeDecodeError	Ошибка Unicode во время декодирования
UnicodeEncodeError	Ошибка Unicode во время кодирования
UnicodeTranslateError ⁱ	Ошибка Unicode во время трансляции
Error ^j	
Warning ^j	Корневой класс для всех предупреждений
DeprecationWarning ^j	Предупреждение о сниженных возможностях
FutureWarning ^j	Предупреждение о логических структурах, которые в будущем изменятся семантически
OverflowWarning ^k	Старое предупреждение для auto-long
PendingDeprecationWarning ^j	Предупреждение о возможностях, которые будут снижены в будущем
RuntimeWarning ^j	Предупреждение о подозрительном поведении во время выполнения
SyntaxWarning ^j	Предупреждение о подозрительном синтаксисе
UserWarning ^j	Предупреждение, генерируемое пользовательским кодом

^a Новое в Python 2.5.^b До появления Python 2.5 — Exception подкласса SystemExit^c До появления Python 2.5 — StandardError подкласса KeyboardInterrupt^d Новое в Python 1.5; этот релиз появился, когда исключения, базирующиеся на классах, пришли на смену строкам.^e Новое в Python 2.2.^f Только для Python 1.5 – Python 2.4.x включительно.^g Новое в Python 2.0.^h Новое в Python 1.6.ⁱ Новое в Python 2.3.^j Новое в Python 2.1.^k Новое в Python 2.2, но изъято в Python 2.4.

Специальные методы для классов

В табл. Б.13 представлена совокупность специальных методов, которые могут быть реализованы, чтобы предоставить возможность объектам, определенным пользователем, усваивать поведение и функциональность стандартных типов Python.

Таблица Б.13. Специальные методы для настройки классов

Специальный метод	Описание
Базовая настройка	
<code>C.__init__(self[, arg1, ...])</code>	Конструктор (с любыми аргументами, задаваемыми по выбору)
<code>C.__new__(self[, arg1, ...])^a</code>	Конструктор (с любыми аргументами, задаваемыми по выбору). Обычно используется для создания подклассов неизменяемых типов данных
<code>C.__del__(self)</code>	Деструктор
<code>C.__str__(self)</code>	Представление строки, которое поддается распечатке; встроенная <code>str()</code> и оператор <code>print</code>
<code>C.__repr__(self)</code>	Представление строки, которое поддается оцениванию; встроенная <code>repr()</code> и оператор <code>' '</code>
<code>C.__unicode__(self)^b</code>	Строковое представление в Unicode; встроенная <code>unicode()</code>
<code>C.__call__(self, *args)</code>	Обозначить экземпляры, которые можно вызывать
<code>C.__nonzero__(self)</code>	Определить значение <code>False</code> для объекта; встроенная <code>bool()</code> (начиная с версии 2.2)
<code>C.__len__(self)</code>	"Длина" (подходящая для класса); встроенная <code>len()</code>
Сравнение объектов (значений)^c	
<code>C.__cmp__(self, obj)</code>	Сравнение объектов; встроенная <code>cmp()</code>
<code>C.__lt__(self, obj)</code> и <code>C.__le__(self, obj)</code>	Меньше, чем/не больше, чем; операторы <code><</code> и <code><=</code>
<code>C.__gt__(self, obj)</code> и <code>C.__ge__(self, obj)</code>	Больше, чем/не меньше, чем; операторы <code>></code> и <code>>=</code>
<code>C.__eq__(self, obj)</code> и <code>C.__ne__(self, obj)</code>	Равно/не равно (чему-либо); операторы <code>==</code> , <code>!=</code> и <code><></code>
Атрибуты	
<code>C.__getattr__(self, attr)</code>	Получить атрибут; встроенная <code>getattr()</code>
<code>C.__setattr__(self, attr, val)</code>	Задать атрибут; встроенная <code>setattr()</code>
<code>C.__delattr__(self, attr)</code>	Удалить атрибут; оператор <code>del</code>
<code>C.__getattribute__(self, attr)^a</code>	Получить атрибут; встроенная <code>getattr()</code>
<code>C.__get__(self, attr)</code>	Получить атрибут; встроенная <code>getattr()</code>
<code>C.__set__(self, attr, val)</code>	Задать атрибут; встроенная <code>setattr()</code>
<code>C.__delete__(self, attr)</code>	Удалить атрибут; оператор <code>del</code>
Настройка классов/Эмуляция типов	
Численные типы: бинарные операторы^d	
<code>C.__add__(self, obj)</code>	Сложение; оператор <code>+</code>
<code>C.__sub__(self, obj)</code>	Вычитание; оператор <code>-</code>
<code>C.__mul__(self, obj)</code>	Умножение; оператор <code>*</code>
<code>C.__div__(self, obj)</code>	Деление; оператор <code>/</code>
<code>C.__truediv__(self, obj)^f</code>	Истинное деление; оператор <code>/</code>
<code>C.__floordiv__(self, obj)^e</code>	Деление с округлением результата; оператор <code>//</code>

Специальный метод	Описание
<code>C._mod_(self, obj)</code>	Модуль/остаток; оператор %
<code>C._divmod_(self, obj)</code>	Деление и модуль; встроенная <code>divmod()</code>
<code>C._pow_(self, obj[, mod])</code>	Возведение в степень; встроенная <code>pow()</code> ; оператор **
<code>C._lshift_(self, obj)</code>	Сдвиг влево; оператор <<
<code>C._rshift_(self, obj)</code>	Сдвиг вправо; оператор >>
<code>C._and_(self, obj)</code>	Поразрядное И; оператор &
<code>C._or_(self, obj)</code>	Поразрядное ИЛИ; оператор
<code>C._xor_(self, obj)</code>	Поразрядное исключающее ИЛИ; оператор ^
Численные типы: унарные операторы	
<code>C._neg_(self)</code>	Унарное отрицание
<code>C._pos_(self)</code>	Унарное "без изменений"
Численные типы: унарные операторы	
<code>C._abs_(self)</code>	Абсолютное значение; встроенная <code>abs()</code>
<code>C._invert_(self)</code>	Поразрядная инверсия; оператор ~
Численные типы: численное преобразование	
<code>C._complex_(self, com)</code>	Преобразовать в комплексное; встроенная <code>complex()</code>
<code>C._int_(self)</code>	Преобразовать в целое; встроенная <code>int()</code>
<code>C._long_(self)</code>	Преобразовать в длинное; встроенная <code>long()</code>
<code>C._float_(self)</code>	Преобразовать в число с плавающей запятой; встроенная <code>float()</code>
Численные типы: базовое представление (строка)	
<code>C._oct_(self)</code>	Восьмеричное представление; встроенная <code>oct()</code>
<code>C._hex_(self)</code>	Шестнадцатеричное представление; встроенная <code>hex()</code>
Численные типы: численное приведение типа данных	
<code>C._coerce_(self, num)</code>	Привести к такому же численному типу; встроенная <code>coerce()</code>
Последовательные типы	
<code>C._len_(self)</code>	Количество элементов в последовательности
<code>C._getitem_(self, ind)</code>	Получить один элемент последовательности
<code>C._setitem_(self, ind, val)</code>	Задать один элемент последовательности
<code>C._delitem_(self, ind)</code>	Удалить один элемент последовательности
<code>C._getslice_(self, ind1, ind2)</code>	Получить срез последовательности
<code>C._setslice_(self, i1, i2, val)</code>	Задать срез последовательности
<code>C._delslice_(self, ind1, ind2)</code>	Удалить срез последовательности
<code>C._contains_(self, val)f</code>	Проверить принадлежность к последовательности; ключевое слово <code>in</code>
<code>C._add_(self, obj)</code>	Конкатенация; оператор +
<code>C._mul_(self, obj)</code>	Повторение; оператор *
<code>C._iter_(self)e</code>	Создать класс итератора; встроенная <code>iter()</code>

Окончание табл. Б.13

Специальный метод	Описание
Типы отображения	
<code>C._len_(self)</code>	Количество элементов в отображении
<code>C._hash_(self)</code>	Хешировать значение функции
<code>C._getitem_(self, key)</code>	Получить значение с заданным ключом
<code>C._setitem_(self, key, val)</code>	Установить значение с заданным ключом
<code>C._delitem_(self, key)</code>	Удалить значение с заданным ключом

^a Новый в Python 2.2; только для использования с классами нового стиля.

^b Новый в Python 2.3.

^c Все, за исключением `cmp()`, новые в Python 2.1.

^d "*" либо ничего (self OP obj), "+" (obj OP self), либо "i" для операции на месте (новый в Python 2.0), т.е. add , radd или iadd .

^c Новый в Python 2.2.

^f НОВЫЙ в Python 1.6.

Сводка операторов Python

В табл. Б.14 представлен полный перечень операторов Python с указанием, к каким стандартным типам они применимы. Эти операторы отсортированы в порядке приоритета их выполнения (старшинства): от более высокого к более низкому, причем те, которые входят в заштрихованную определенным образом группу, имеют одинаковый приоритет.

Таблица Б.14. Операторы Python († — унарные)

[illegible]

Окончание табл. Б.14

Оператор ^a	int ^b	long	float	complex	str	list	tuple	dict	set, frozenset ^c
>	•	•	•	•	•	•	•	•	•
<=	•	•	•	•	•	•	•	•	•
>=	•	•	•	•	•	•	•	•	•
==	•	•	•	•	•	•	•	•	•
!=	•	•	•	•	•	•	•	•	•
<>	•	•	•	•	•	•	•	•	•
is	•	•	•	•	•	•	•	•	•
is not	•	•	•	•	•	•	•	•	•
in					•	•	•		•
not in					•	•	•		•
not [†]	•	•	•	•	•	•	•	•	•
and	•	•	•	•	•	•	•	•	•
or	•	•	•	•	•	•	•	•	•

^a Может также включать соответствующие расширенные операторы присвоения.^b Операции, связанные с булевыми типами, будут выполняться на операндах как ints.f.^c (Оба) типа множеств; новые в Python 2.4.

ПРИЛОЖЕНИЕ

В

Версия Python 3: эволюция языка программирования

*У Матца (автора Ruby) есть замечательное высказывание:
“Открытый исходный код должен развиваться;
в противном случае он обречен на смерть”.*

Гвидо ван Россум, март 2008 г.
(из выступления на конференции PyCon)

Версия Python 3 представляет собой такую эволюцию языка, в результате которой большинство “старого” кода, разработанного для интерпретаторов версии 2.x, выполняться не будет. Это не означает, что вам уже не удастся распознать старый код или что приведение старого кода в работоспособное состояние под версией 3.x потребует массового переноса. Вообще говоря, новый синтаксис имеет много общего со старым синтаксисом. Однако, учитывая отсутствие в версии 3.x оператора `print`, проблемы с выполнением старого кода, действительно, могут возникнуть. В этом приложении мы обсуждаем `print` и другие изменения в версии 3.x; мы также прольем свет на изменения, которым должен подвергнуться язык Python, чтобы стать лучше, чем он был прежде. Наконец, мы познакомим читателей с несколькими инструментами миграции, которые должны помочь вам перейти к версии Python 3.

В.1. Почему изменяется язык Python

В настоящее время язык Python переживает стадию самой значительной трансформации с момента его появления в начале 1990-х годов. Даже переход в 2000 г. от версии 1.x к версии 2.x был связан с относительно небольшими переменами: программы, предназначенные для 1.5.2, выполнялись вполне корректно в версии Python 2.0. Одной из основных причин стабильности языка Python на протяжении многих лет была твердая приверженность ядра команды разработчиков идее обеспечения обратной совместимости. Однако со временем создатель этого языка, Гвидо ван Россум, а также Эндрю Кюхлинг и другие пользователи (ссылки на соответствующие публикации можно найти в разделе “Список литературы”, помещенном в конце этого приложения) выявили ряд “назойливых” дефектов (проблем, передающихся “по наследству” от старых версий к новым). Устойчивость этих дефектов указала на важность создания принципиально новой версии как на необходимое условие успешного эволюционирования языка. Появление в 2008 г. версии 3.0 ознаменовало собой первый в истории этого языка сознательный отказ от обязательного обеспечения обратной совместимости.

В.2. Что изменилось

Изменения в версии Python 3 нельзя назвать ошеломляющими: вряд ли человек, хорошо знакомый с предыдущими версиями Python, мог бы сказать, что не узнает Python. В этом приложении представлен обзор некоторых из самых важных изменений.

- Оператор `print` становится функцией `print()`.
- Строки преобразуются в кодировку Юникод по умолчанию.
- Реализован единственный тип класса.
- Обновлен синтаксис для исключений.
- По-другому определены целые числа.
- Повсеместно используются `iterable`.

В.2.1. Оператор `print` становится функцией `print()`

Переход к функции `print()` является изменением, которое делает непригодной для использования большую часть уже существующего кода на языке Python. Почему же было решено перейти от оператора к встроенной функции? Использование оператора `print` ограничивает возможности разработчика во многих отношениях, о чем подробно рассказал Гвидо ван Россум в своей лекции “Python Regrets” (“Извинения по поводу Python”). В этой лекции он поведал о том, что, по его мнению, относится к числу недостатков языка Python. Кроме того, использование оператора `print` не позволяет внести в этот язык необходимые улучшения. Однако в случае использования встроенной функции `print()` появляется возможность добавить новые параметры-ключевые слова, с помощью которых можно избавиться от некоторых моделей стандартного поведения, а функцию `print()` можно, при желании, заменить, как и любую другую. Ниже приведены примеры “до и после”.

Python 2.x

```
>>> i = 1
>>> print 'Python' 'is', 'number', i
Pythonis number 1
```

Python 3.x

```
>>> i = 1
>>> print('Python' 'is', 'number', i)
Pythonis number 1
```

Пропуск запятой между литералами 'Python' и 'is' является намеренным: это было сделано для того, чтобы показать, что непосредственная конкатенация строковых литералов не изменилась. С другими примерами можно ознакомиться в документе “What's New in Python 3.0” (см. раздел “Список литературы” в конце этого приложения). Дополнительную информацию об этом изменении можно найти в PEP 3105.

В.2.2. Строки: Юникод по умолчанию

Следующей новацией, на которую обращают внимание нынешние пользователи языка Python, является то, что строки теперь представлены в кодировке Юникод по умолчанию. Это изменение относится к числу долгожданных. Не проходит и дня, чтобы бесчисленные разработчики на языке Python, работая со строками в кодировке Юникод и обычными строками в кодировке ASCII, не столкнулись с проблемами такого рода:

```
UnicodeEncodeError: 'ascii' codec can't encode character
u'\xae' in position 0: ordinal not in range(128)
```

При использовании версии 3.x ошибки такого типа перестанут быть повседневным явлением. Более подробную информацию об использовании кодировки Юникод в среде Python можно найти в документе по кодировке Юникод, озаглавленном HOWTO (соответствующий веб-адрес см. в разделе “Список литературы” в конце этого приложения). При использовании модели, принятой в новой версии языка Python, пользователям даже не придется пользоваться такими терминами, как “строки Юникод” и “строки ASCII/не-Юникод”. Суть этой новой модели достаточно подробно изложена в документе “What's New in Python 3.0”.

Вместо строк Юникод и 8-битовых строк в версии Python 3 используются концепции *текста* и (двоичных) *данных*. Весь текст представлен в кодировке Юникод; однако закодированный текст в кодировке Юникод представлен как двоичные данные. Типом, который используется для хранения текста, является `str`, а типом, который используется для хранения данных, — `bytes`.

Относительно синтаксиса: поскольку сейчас по умолчанию используется кодировка Юникод, ведущая буква `u` или `U` игнорируется. Аналогично новые объекты типа `bytes` требуют наличия ведущей буквы `b` или `B` для *своих* литералов (более подробную информацию об этом можно найти в PEP 3112).

В табл. В.1 сравниваются разные типы строк; здесь же показано, как они изменятся при переходе от версии 2.x к 3.x. Эта таблица включает также упоминание о новом изменяемом типе `bytearray`.

Таблица В.1. Строки в Python 2 и 3

Python 2	Python 3	Изменяемый?
<code>str("")</code>	<code>bytes(b'')</code>	Нет
<code>unicode(u'')</code>	<code>str("")</code>	Нет
отсутствует	<code>bytearray</code>	Да

В.2.3. Единственный тип класса

До появления версии Python 2.2 объекты языка Python не вели себя подобно классам в других языках: классами были объекты “класс”, а экземплярами — объекты “экземпляр”. Это резко контрастирует с тем, что люди считают нормальным: классы являются типами, а экземпляры — объектами таких типов. Из-за этого недостатка вы не могли распределять типы данных по подклассам и модифицировать их. В версии Python 2.2 ядро команды разработчиков предложило концепцию *классов нового стиля*, которые ведут себя во многом именно так, как того ожидают от них. Кроме того, это изменение означало, что регулярные типы Python можно распределять по подклассам (это изменение подробно описано в эссе Гвидо ван Россума “Унификация типов и классов в Python 2.2”). Язык Python 3 поддерживает только классы нового стиля.

В.2.4. Обновленный синтаксис для исключений

Обработка исключений

В прошлом синтаксис, позволяющий выявить исключение и аргумент/экземпляр исключения, имел следующую форму:

```
except ValueError, e:
```

Чтобы выявить несколько исключений с помощью одной и той же программы обработки, использовался следующий синтаксис:

```
except (ValueError, TypeError), e:
```

Требуемые этим синтаксисом круглые скобки запутывали некоторых пользователей, которые зачастую пытались написать недопустимый код, который мог выглядеть примерно так:

```
except ValueError, TypeError, e:
```

Новое ключевое слово **as** призвано гарантировать, что вас не смутит запятая в исходном синтаксисе; однако круглые скобки все еще требуются, когда вы пытаетесь распознать несколько типов исключений с помощью одной и той же программы обработки. Ниже приведены два эквивалентных примера нового синтаксиса, которые иллюстрируют указанное изменение:

```
except ValueError as e:
except (ValueError, TypeError) as e:
```

Оставшиеся выпуски версии 2.x (начиная с 2.6) принимают обе формы при создании программ обработки исключений, что, несомненно, облегчает процесс переноса. Более подробную информацию об этом изменении см. в PEP 3110.

Вызов исключений

Самый популярный синтаксис для вызова исключений в версии Python 2.x выглядит так:

```
raise ValueError, e
```

Чтобы подчеркнуть, что вы действительно создаете экземпляр исключения, единственный синтаксис, поддерживаемый в версии Python 3.x, выглядит так:

```
raise ValueError(e)
```

В действительности этот синтаксис отнюдь не нов. Он появился примерно десять лет тому назад в версии Python 1.5 (не пугайтесь, ваше зрение не подводит вас!), когда исключения трансформировались из строк в классы, и мы уверены, вы согласитесь с тем, что синтаксис для создания экземпляров классов гораздо больше похож на то, что появилось впоследствии, чем на то, что было прежде.

В.2.5. Обновления, касающиеся целых чисел

Единственный тип целых чисел

Унификация двух разных типов целых чисел в языке Python, `int` и `long`, началась еще в версии Python 2.2. Сейчас это изменение практически завершилось, и новый тип `int` ведет себя подобно `long`. Как следствие, исключения `OverflowError` уже не случаются, когда вы превышаете естественный размер целых чисел, а концевая буква `L` уже не используется. Об этом изменении можно прочитать в PEP 237. Тип `long` по-прежнему существует в версии Python 2.x, однако в версии Python 3.0 его уже нет.

Изменения, касающиеся деления

Нынешний оператор деления (`/`) не дает ожидаемого ответа для тех пользователей, которые еще не имеют достаточного опыта в программировании; именно поэтому его было решено изменить. Если это изменение и породило какие-то противоречия, то лишь потому, что программисты привыкли к функциональности деления с округлением результата в меньшую сторону. Чтобы уяснить механизм возникновения путаницы в связи с этим, попытайтесь убедить новичка-программиста, что результатом деления 1 на 2 является 0 ($1 / 2 == 0$). Проще всего описать это изменение с помощью подходящих примеров. Ниже приведено несколько примеров, заимствованных из статьи “Keeping Up with Python: The 2.2 Release”, опубликованной в июльском выпуске *Linux Journal* за 2002 г. Дополнительную информацию об этом изменении см. в PEP 238.

Классическое деление

Операция деления, предусмотренная по умолчанию в версии Python 2.x, работает так: заданы два целочисленных операнда; операция `/` выполняет деление нацело с

округлением результата в меньшую сторону (дробная часть отбрасывается, как в приведенном выше примере). Если в операции деления участвует по меньшей мере одно число с плавающей точкой (float), то выполняется истинное деление:

```
>>> 1 / 2 # floor
0
>>> 1.0 / 2.0 # true
0.5
```

Истинное деление

Если в версии Python 3.x заданы два численных операнда, то операция / всегда возвращает число с плавающей точкой (float):

```
>>> 1 / 2 # true
0.5
>>> 1.0 / 2.0 # true
0.5
```

Чтобы проверить работу истинного деления начиная с версии Python 2.2, либо импортируйте операцию division из модуля `_future_`, либо используйте переключатель `-Qnew`.

Деление с округлением результата в меньшую сторону

Оператор деления “двойная косая черта” (//) был добавлен в версию Python 2.2 для того, чтобы всегда, независимо от типа операндов, выполнялась операция деления с округлением результата в меньшую сторону и чтобы начать процесс перехода:

```
>>> 1 // 2 # floor
0
>>> 1.0 // 2.0 # floor
0.0
```

Двоичные и восьмеричные литералы

В версию Python 2.6+ были внесены незначительные изменения, касающиеся целочисленных литералов, чтобы обеспечить совместимость недесятичных (шестнадцатеричных, восьмеричных и новых двоичных) форматов литералов. Шестнадцатеричное представление, с его ведущими 0x или 0X (где восьмеричное раньше предвлялось лишь одним 0), осталось прежним. Для некоторых пользователей такой формат стал причиной путаницы, поэтому, в целях единообразия, было решено заменить его на 0o. Таким образом, вместо 0177 вы должны теперь использовать 0o177. Наконец, новый двоичный литерал позволяет задавать двоичные разряды целочисленного значения, предвляемого ведущими 0b (например, 0b0110). Версия Python 3 не принимает 0177. Более подробную информацию об этих изменениях см. в PEP 3127.

В.2.6. Повсеместное использование итераторов

Еще одной неотъемлемой темой версии 3.x является более экономичное использование оперативной памяти. Использование итераторов оказывается гораздо более эффективным, чем поддержание списков в оперативной памяти целиком, особенно

когда предполагается, что в отношении соответствующих объектов будут выполняться итерации. В любом случае оперативную память следует расходовать экономно. Таким образом, код, который в прежних версиях языка Python возвращал списки, в версии Python 3 уже не делает этого.

Например, функции `map()`, `filter()`, `range()` и `zip()`, а также методы словарей `keys()`, `items()` и `values()` возвращают тот или иной вид итератора. Да, такой синтаксис может быть более неудобным, если вы хотите взглянуть на свои данные, однако он гораздо лучше с точки зрения экономичного использования оперативной памяти. Соответствующие изменения в основном скрыты от глаз пользователя: если значения, возвращаемые указанными функциями, вы используете лишь для выполнения итераций, то не заметите никаких изменений.

В.3. Инструменты миграции

Как вы, наверное, уже заметили, большинство изменений в версии Python 3.x не похоже на безудержную мутацию хорошо знакомого нам синтаксиса языка Python. Однако этих изменений оказалось вполне достаточно, чтобы они ознаменовали собой радикальное изменение старой базы кодов. Разумеется, эти изменения не могли не сказаться на пользователях, а это указывает на необходимость эффективного плана перехода. Большинству эффективных планов сопутствует создание надежных инструментов и вспомогательных средств, облегчающих процесс перехода. К числу таких инструментов относятся следующие (разумеется, есть и другие инструменты, о которых мы здесь не упоминаем): конвертор кодов `2to3`, самый последний выпуск Python 2.x (как минимум 2.6), а также внешний инструмент `3 to 2` и библиотека `six`. Первые два мы обсудим в этой главе, а последние два изучите самостоятельно.

В.3.1. Инструмент `2to3`

Инструмент `2to3`, используя в качестве основы код для версии Python 2.x, пытается сгенерировать работоспособный эквивалент для версии Python 3.x. Ниже перечислены некоторые из выполняемых им действий.

- Преобразует оператор `print` в функцию `print()`.
- Удаляет суффикс `long (L)`.
- Заменяет `<>` на `!=`.
- Заменяет строки, заключенные в одиночные обратные апострофы (`'...'`), на `repr(...)`.

Этот инструмент выполняет большой объем ручной работы, но, к сожалению, не всю эту работу: остальную ручную работу придется выполнить самому пользователю. Подробнее о переносе предложений (porting suggestions) и инструменте `2to3` можно прочитать в документе “What’s New in Python 3.0”, а также на веб-странице этого инструмента (<http://docs.python.org/3.0/library/2to3.html>). В приложении Г мы также кратко упомянем об “инструменте-спутнике” `2to3` под названием `3to2`.

В.3.1. Версия Python 2.6+

Из-за проблем совместимости выпуски языка Python, которые привели к появлению версии Python 3.0, играют гораздо более важную роль в переходе к Python 3.0.

В частности, следует упомянуть о версии Python 2.6 — первом и самом главном из таких выпусков. Для пользователей этот выпуск представляет собой первую возможность, когда они могут начать кодировать с прицелом на семейство выпусков версии 3.x, поскольку в версии 2.x обеспечивается ретроподдержка многих возможностей, заложенных в выпусках 3.x.

По мере возможностей выпуски финальной версии 2.x (2.6 и более новые) включают новые функции и синтаксис из версии 3.x, оставаясь в то же время совместимыми с существующим кодом за счет сохранения более старых функций и синтаксиса. Такие функции описаны в документе “What's New in Python 3.0” для всех таких релизов. Некоторые из этих “миграционных” возможностей подробно рассматриваются в приложении Г.

В.4. Выводы

В целом, изменения, описанные в этом приложении, окажут большое влияние с точки зрения обновлений, которые необходимо внести в интерпретатор, однако они не должны радикально изменить сам способ написания программистами своего кода на языке Python. Это лишь вопрос изменения старых привычек, таких как использование круглых скобок с `print`, т.е. написания `print()`. После того как вы освоите эти изменения, вы значительно облегчите себе процесс перехода на новую платформу. Поначалу многое покажется непривычным и удивительным, но со временем вы будете воспринимать эти изменения как должное. Главное, не паниковать; версия Python 2.x будет еще долго использоваться. Переход к версии Python 3.0 будет происходить медленно, постепенно, осторожно и практически безболезненно. Все мы являемся свидетелями рождения нового поколения языка Python!

В.5. Список литературы

Andrew Kuchling, “Python Warts,” July 2003, [http://web.archive.org/web/20070607112039, http://www.amk.ca/python/writing/warts.html](http://web.archive.org/web/20070607112039/http://www.amk.ca/python/writing/warts.html)

A. M. Kuchling, “What's New in Python 2.6,” June 2011 (for 2.6.7), <http://docs.python.org/whatsnew/2.6.html>.

A. M. Kuchling, “What's New in Python 2.7,” December 2011 (for 2.7.2), <http://docs.python.org/whatsnew/2.7.html>.

Wesley J. Chun, “Keeping Up with Python: The 2.2 Release,” July 2002, <http://www.linuxjournal.com/article/5597>.

PEP Index, <http://www.python.org/dev/peps>.

“Unicode HOWTO,” December 2008, <http://docs.python.org/3.0/howto/unicode.html>.

Guido van Rossum, “Python Regrets,” July 2002, <http://www.python.org/doc/essays/ppt/regrets/PythonRegrets.pdf>.

Guido van Rossum, “Unifying Types and Classes in Python 2.2,” April 2002, <http://www.python.org/2.2.3/desccintro.html>.

Guido van Rossum, “What's New in Python 3.0,” December 2008, <http://docs.python.org/3.0/whatsnew/3.0.html>.

Переход к версии Python 3 на основе выпуска Python 2.6+

Мы стремимся постоянно развивать этот язык; нам нужно все время двигаться вперед, в противном случае язык обречен на смерть.

Юкихио “Матц” Мацумото, сентябрь 2008 г.
(из выступления на конференции Lone Star Ruby;
в данном случае Мацумото цитирует слова Гвидо ван Россума)

Г.1. Язык Python 3: следующее поколение

В настоящее время язык Python переживает стадию самой значительной трансформации с момента появления первого выпуска этого языка программирования зимой 1991 г. В версии Python 3 не обеспечивается обратная совместимость со всеми предыдущими версиями языка Python, поэтому проблема переноса будет иметь более важное значение, чем в прошлом.

Однако не стоит ожидать исчезновения версии Python 2.x в обозримом будущем. На самом деле последние выпуски версии Python 2.x будут развиваться параллельно выпускам Python 3.x, что послужит гарантией плавного перехода от нынешнего поколения к следующему. Выпуск Python 2.6 является первым из этих финальных выпусков версии Python 2.x.

Настоящий документ дополняет материал, изложенный в приложении В, “Python 3: эволюция языка программирования”, углубляя и детализируя его в необходимых случаях.

Г.1.1. Гибридная версия 2.6+ как инструмент перехода

Версия Python 2.6 и оставшиеся выпуски 2.x являются гибридными интерпретаторами. Это означает, что с их помощью может выполняться значительная часть кода версии Python 1.x и все программное обеспечение версии 2.x; более того, с их помощью может даже выполняться определенная часть кода версии 3.x (код, который является “своим” для версии 3.x, но который может выполняться в версии 2.6+). Кое-кто даже утверждает, что выпуски Python вплоть до версии 2.2 (по нисходящей) уже представляли собой гибридные интерпретаторы, поскольку они поддерживают создание как классических классов, так и классов нового стиля, впрочем, на большее разработчики этих выпусков и не замахивались.

Выпуск версии 2.6 представляет собой первую версию, обладающую специфическими возможностями версии 3.x. Иными словами, выпуск версии 2.6 обеспечивает ретроподдержку специфических возможностей версии 3.x. Ниже перечислены самые важные из этих возможностей.

- Целые числа
 - Единый целочисленный тип
 - Новые двоичные и модифицированные восьмеричные литералы
 - Классическое или истинное деление
 - Переключатель деления `-Q`
- Встроенные функции
 - `print` или `print()`
 - `reduce()`
 - Прочие обновления
- Объектно-ориентированное программирование
 - Два разных объекта класса
- Строки
 - Литералы `bytes`
 - Тип `bytes`
- Исключения
 - Обработка исключений
 - Вызов исключений
- Другие инструменты перехода и рекомендации
 - Предостережения: переключатель `-3`
 - Инструмент `2to3`

В этом приложении не обсуждаются другие новые возможности версии 2.x, которые относятся к категории автономных (это означает, что они никак не сказываются

на переносе приложений в версию 3.x). Итак, незамедлительно приступаем к рассмотрению интересующего нас вопроса по существу.

Г.2. Целые числа

В версии Python 3.x целые числа подверглись нескольким изменениям, которые коснулись типов целых чисел, литералов, а также операции деления целых чисел. Ниже мы опишем каждое из этих изменений, подчеркнув роль, которую играют версия 2.6, а также последующие версии с точки зрения миграции.

Г.2.1. Единый целочисленный тип

В предыдущих версиях Python было предусмотрено два типа целых чисел, `int` и `long`. Первоначальные целые числа типа `int` были ограничены по своему размеру архитектурой платформы, на которой выполнялся соответствующий код (т.е. 32- или 64-разрядной), тогда как целые числа типа `long` не были ограничены по своему размеру (единственным исключением в этом смысле был объем виртуальной памяти, предоставляемый операционной системой). Процесс унификации этих двух типов и их объединения в один целочисленный тип `int` начался в версии Python 2.2 и завершится в версии 3.0¹. Новый единый тип `int` будет неограниченным по размеру, а прежнее обозначение типа `long` (`L` или `l`) вообще упразднено. Подробнее об этом изменении см. в PEP 237.

В версии 2.6 от целочисленного типа `long` осталось совсем немного, а именно поддержка концевого `L`. Такая поддержка включена для обеспечения обратной совместимости, т.е. для обеспечения работоспособности кода, в котором используются целые числа типа `long`. Тем не менее пользователям следует решительно избавляться от целых чисел типа `long` в уже существующем коде и отказаться от дальнейшего использования целых чисел этого типа при кодировании новых приложений, разрабатываемых на основе версии Python 2.6+.

Г.2.2. Новые двоичные и модифицированные восьмеричные литералы

В версию Python 3 внесено небольшое изменение в альтернативный базовый формат для целых чисел. Соответствующий синтаксис был, по сути, упрощен с целью обеспечения его единства с уже существующим шестнадцатеричным форматом, использующим префикс в виде ведущих `0x` (или `0X` для прописных букв), например `0x80`, `0xffff`, `0xDEADBEEF`.

Новый двоичный литерал позволяет создать двоичные разряды для целого числа, предваряемые ведущими `0b` (например, `0b0110`). Первоначальное восьмеричное представление начиналось с одиночного `0`, однако такой формат вносил путаницу в умы некоторых пользователей, поэтому было решено заменить его на `0o`, приведя его таким образом в соответствие с шестнадцатеричными и двоичными литералами, как указывалось выше. Иными словами, вместо `0177` вы должны теперь использовать формат `0o177`. Ниже приведено еще несколько примеров.

¹ Тип `bool` также может считаться частью этого "уравнения", поскольку числа типа `bool` ведут себя в "численных" ситуациях подобно `0` и `1`, но вовсе не ведут себя так, будто их естественными значениями являются `False` и `True` соответственно.

Python 2.x

```
>>> 0177
127
```

Python 3 (в том числе 2.6+)

```
>>> 0o177
127
>>> 0b0110
6
```

Как новый двоичный, так и модифицированный формат восьмеричного литерала реализован в версии 2.6 с целью облегчения миграции. По сути, версия 2.6 и последующие выпуски принимают — в выполняемой ими роли инструментов перехода — оба восьмеричных формата, тогда как ни один из выпусков версии 3.x не принимает старый формат 0177. Дополнительную информацию об этих изменениях, касающихся целочисленных литералов, см. в PEP 3127.

Г.2.3. Классическое или истинное деление

Изменением, которого так долго ждали и которое, тем не менее, остается неоднозначным для многих пользователей, является изменение оператора деления (/). Традиционная операция деления работает следующим образом: заданы два целочисленных операнда; операция / выполняет деление нацело с округлением результата в меньшую сторону. Если в операции деления участвует по меньшей мере одно число с плавающей точкой (float), то выполняется истинное деление.

Версия Python 2.x: классическое деление

```
>>> 1 / 2 # floor
0
>>> 1.0 / 2.0 # true
0.5
>>> 1.0 / 2 # true (2 автоматически преобразовывается в float)
0.5
```

В языке Python 3 оператор / всегда возвращает число с плавающей точкой, независимо от типа операнда.

Версия Python 3.x: истинное деление

```
>>> 1 / 2 # true
0.5
>>> 1.0 / 2 # true
0.5
```

Оператор деления “двойная косая черта” (//) был добавлен в версии Python 2.2 в качестве “посредника” для того, чтобы всегда, независимо от типа операндов, выполнялась операция классического деления с округлением результата в меньшую сторону и чтобы начать процесс перехода.

Версии Python 2.2+ и 3.x: деление с округлением результата в меньшую сторону

```
>>> 1 // 2 # floor
0
>>> 1.0 // 2 # floor
0.0
```

В версии 3.x использование оператора `//` будет единственным способом обеспечения функциональности деления с округлением результата в меньшую сторону. Чтобы проверить выполнение операции истинного деления в версии Python 2.2+, либо импортируйте оператор `division` в свой код из модуля `_future_`, либо используйте опцию командной строки `-Q` (обсуждается ниже).

Версия Python 2.2+: опция командной строки `division`

Если вы не хотите импортировать оператор `division` в свой код из модуля `_future_`, но хотите, чтобы всегда выполнялась преимущественно операция истинного деления, используйте переключатель `-Qnew`. Существуют и другие варианты использования `-Q`, перечисленные в табл. Г.1.

Таблица Г.1. Опции командной строки операции деления `-Q`

Опция	Описание
old	Всегда выполнять классическое деление
new	Всегда выполнять истинное деление
warn	Предостерегать против операций <code>int/int</code> и <code>long/long</code>
warnall	Предостерегать против любого использования /

Например, опция `-Qwarnall` используется в сценарии `Tools/scripts/fixdiv.py`, включенном в дистрибутив исходного кода Python.

Как вы уже могли догадаться, все меры по обеспечению перехода уже реализованы в версии Python 2.2, и никакая особая дополнительная функциональность в том, что касается этой командной строки, не была добавлена в версии 2.6 или 2.7 для обеспечения перехода к версии Python 3. В табл. Г.2 приведена сводка операторов деления и их функциональности в разных выпусках Python.

Таблица Г.2. Предусмотренная по умолчанию функциональность операторов деления в разных выпусках Python

Оператор	2.1–	2.2+	3.x ^a
/	Классическое	Классическое	Истинное
//	Неприменимо	С округлением результата в меньшую сторону	С округлением результата в меньшую сторону

^a Столбец “3.x” также применим к Python 2.2+ с `-Qnew` или импортом `_future_.division`

Подробнее об изменении, внесенном в оператор деления, можно прочитать в PEP 238, а также в статье “Keeping Up with Python: The 2.2 Release”, которую я написал для июльского выпуска *Linux Journal* 2002 г.

Г.3. Встроенные функции

Г.3.1. Оператор `print` или функция `print()`

Не секрет, что одной из самых распространенных причин противоречия между приложениями Python 2.x и 3.x является изменение оператора `print`, который в версии 3.x превратился во встроенную функцию. Это изменение призвано обеспечить большую гибкость функции `print()`, большую ее способность к модернизации и, по мере необходимости, к взаимозаменяемости.

Версия Python 2.6 и более новые выпуски поддерживают либо оператор `print`, либо встроенную функцию `print()`. По умолчанию предполагается использование прежнего варианта, как это должно быть в версии Python 2.x. Чтобы отказаться от оператора `print` и пользоваться лишь встроенной функцией `print()` в каком-либо из приложений “режима Python 3”, вам следует просто импортировать оператор `print_function` из модуля `_future_`:

```
>>> print 'foo', 'bar'
foo bar
>>>
>>> from __future__ import print_function
>>> print
<built-in function print>
>>> print('foo', 'bar')
foo bar
>>> print('foo', 'bar', sep='-')
foo-bar
```

Приведенный выше пример демонстрирует огромные возможности `print()` как функции. Пользуясь оператором `print`, мы отображаем для пользователя строки `foo` и `bar`, но мы не можем изменить разделитель или разграничитель между строками, используемый по умолчанию, роль которого исполняет пробел. В отличие от оператора `print`, функция `print()` допускает такую возможность за счет использования аргумента `sep` при вызове этой функции. Аргумент `sep` заменяет разделитель, используемый по умолчанию, и допускает дальнейшее расширение возможностей `print`.

Обратите внимание: в данном случае речь идет об одностороннем импорте. Это означает, что у нас нет возможности выполнить обратное преобразование функции `print()` в оператор. Даже выдав сообщение “`del print_function`”, мы не сможем повлиять на эту ситуацию. Это важное изменение подробно рассматривается в документе PEP 3105.

Г.3.2. Функция `reduce()` перенесена в модуль `functools`

В версии Python 3.x функция `reduce()` “разжалована” (к неудовольствию многих функциональных программистов на языке Python) из категории встроенных функций в одну из функций модуля `functools`. Начало этому было положено в версии 2.6:

```
>>> from operator import add
>>> reduce(add, range(5))
10
>>>
>>> import functools
>>> functools.reduce(add, range(5))
10
```

Г.3.3. Прочие обновления

Одной из ключевых тем в версии Python 3.x является переход к более широкому использованию итераторов, особенно для встроенных функций и методов, которые, по сложившейся традиции, возвращали списки. Другие итераторы подверглись изменению по причине внесения изменений в целые числа. Ниже перечислены самые заметные встроенные функции, которые изменились в версии Python 3.x.

- `range()`
- `zip()`
- `map()`
- `filter()`
- `hex()`
- `oct()`

Начиная с версии Python 2.6 программисты могли обращаться к новым и обновленным функциям путем импортирования модуля `future_builtins`. Ниже приведен пример, который демонстрирует как старую, так и новую функции `oct()` и `zip()`.

```
>>> oct(87)
'0127'
>>>
>>> zip(range(4), 'abcd')
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
>>> dict(zip(range(4), 'abcd'))
{0: 'a', 1: 'b', 2: 'c', 3: 'd'}
>>>
>>> import future_builtins
>>> future_builtins.oct(87)
'0o127'
>>>
>>> future_builtins.zip(range(4), 'abcd')
<itertools.izip object at 0x374080>
>>> dict(future_builtins.zip(range(4), 'abcd'))
{0: 'a', 1: 'b', 2: 'c', 3: 'd'}
```

Если вы хотите использовать в своей нынешней среде Python 2.x лишь версии этих функций для Python 3.x, то можете заменить старые версии этих функций, импортировав все новые функции в свое пространство имен. Приведенный ниже пример демонстрирует этот процесс с помощью функции `oct()`.

```
>>> from future_builtins import *
>>> oct(87)
'0o127'
```

Г.4. Объектно-ориентированное программирование: два разных объекта классов

Оригинальные классы языка Python теперь называются *классическими классами*. Оригинальные классы языка Python обладают множеством недостатков и со временем

были заменены на классы *нового стиля*. Этот переход начался в Python 2.2 и продолжается по сей день.

Традиционные классы используют следующий синтаксис:

```
class ClassicClass:
    pass
```

А классы нового стиля используют такой синтаксис:

```
class NewStyleClass(object):
    pass
```

Классы нового стиля имеют так много преимуществ по сравнению с классическими классами, что последние оставались лишь для обеспечения обратной совместимости, но в версии Python 3 от них было решено полностью отказаться. При использовании классов нового стиля наконец удалось унифицировать типы и классы (см. эссе Гвидо ван Россума “Унификация типов и классов в Python 2.2”, а также документы PEP 252 и PEP 253).

Каких-либо других изменений, которые были бы внесены в версию Python 2.6 и более свежие выпуски для обеспечения миграции, не существует, если не считать декораторы классов как новую возможность версии 3.x. Нужно лишь иметь в виду, что все версии 2.2+ функционируют как гибридные интерпретаторы, допуская возможность объектов классов и экземпляров таких классов. В версии Python 3 оба синтаксиса, показанные в предыдущих примерах, приводят лишь к созданию классов нового стиля. Такое поведение не представляет серьезной проблемы для переноса, но вы должны иметь в виду, что классических классов в версии Python 3 уже нет.

Г.5. Строки

Одним из особенно заметных изменений в версии Python 3.x является изменение типа строки, предусмотренного по умолчанию. Python 2.x поддерживает строки в кодировках ASCII и Юникод, причем по умолчанию используется ASCII. В версии Python 3 зеркальная ситуация: по умолчанию используется Юникод, а строки ASCII теперь называются `bytes`. Структура данных `bytes` содержит байтовые значения и, вообще говоря, не должна была бы считаться строкой, поскольку представляет собой неизменяемый массив байтов, который содержит данные.

Нынешние строковые литералы потребуют в версии Python 3.x наличия ведущей буквы `b` или `B`, а нынешние строковые литералы в кодировке Юникод утратят свою ведущую букву `u` или `U`. Наименования типа и встроенные функции поменяются со `str` на `bytes` и с `unicode` на `str`. Кроме того, появился новый изменяемый строковый тип `bytearray`, который, подобно `bytes`, также представляет собой массив байтов — правда, изменяемый.

Более подробную информацию об использовании строк Юникод можно найти в документе HOWTO, а узнать о предстоящих изменениях в типах строк можно в документе PEP 3137. Сведения о разных типах строк и о том, как они будут изменяться при переходе от версии 2.x к 3.x, см. в табл. В.1.

Г.5.1. Литералы bytes

Чтобы упростить переход к использованию объектов bytes в версии Python 3.x, вы можете, при желании, предварить обычную строку (ASCII или двоичную) в версии Python 2.6 ведущим символом b или B, создавая таким образом литералы bytes (b' ' или B' ') как синонимы литералов str (' '). Этот ведущий индикатор никак не сказывается на каком-либо объекте str (как на самом объекте, так и на любых операциях с ним). Таким образом, этот ведущий индикатор является чисто “декоративным”, однако он подготавливает вас к ситуациям в версии Python 3, для которых вам необходимо создавать такой литерал. Более подробную информацию о литералах bytes см. в документе PEP 3112.

Тип bytes — это str

Вам не понадобится чересчур напрягать свое воображение, чтобы понять, что если в версии Python 2.6+ обеспечивается поддержка литералов типа bytes, то в этой версии должны существовать сами по себе объекты типа bytes. Действительно, тип bytes синонимичен str, как показано в приведенном ниже примере.

```
>>> bytes is str
True
```

Следовательно, вы можете использовать тип bytes или функцию bytes() в версии Python 2.6+ там же, где используете тип str или функцию str(). Более подробную информацию об объектах bytes см. в документе PEP 358.

Г.6. Исключения

Версии Python 2.6 и более современные выпуски версии 2.x обладают несколькими возможностями, которые вы можете использовать для переноса обработки исключений и порождения исключений в версии Python 3.x.

Г.6.1. Обработка исключений (использование оператора as)

Синтаксис в версии Python 3, предназначенный для перехвата и обработки отдельно взятого исключения, выглядит так:

```
except ValueError as e:
```

Переменная e содержит экземпляр исключения, который объясняет, почему была сгенерирована ошибка. Она является необязательной, как и в целом фраза as e. Следовательно, данное изменение в действительности касается лишь тех пользователей, которые сохраняют это значение.

Эквивалентный синтаксис в версии Python 2 использует запятую вместо ключевого слова as:

```
except ValueError, e:
```

Это изменение было внесено в версии Python 3.x из-за путаницы, которая возникает, когда программисты пытаются обрабатывать несколько исключений с помощью одной и той же программы обработки.

Чтобы перехватывать разные исключения с помощью одной и той же программы обработки, начинающие программисты нередко пишут такой (неправильный) код:

```
except ValueError, TypeError, e:
```

В действительности, если вы пытаетесь перехватывать несколько исключений, нужно использовать кортеж, который содержит эти исключения:

```
except (ValueError, TypeError), e:
```

Ключевое слово **as** в версии Python 3.x (и версии 2.6+) призвано гарантировать, что запятая в исходном синтаксисе уже не будет причиной путаницы. Однако круглые скобки все еще требуются, когда вы попытаетесь перехватывать более одного типа исключений с помощью одной и той же программы обработки:

```
except (ValueError, TypeError) as e:
```

Для того чтобы облегчить процесс переноса, версия Python 2.6 и более современные выпуски принимают либо запятую, либо **as** при определении обработчиков исключений, которые сохраняют соответствующий экземпляр. С другой стороны, в версии Python 3 допускается лишь идиома с **as**. Более подробную информацию об этом изменении см. в документе PEP 3110.

Г.6.2. Генерация исключений

Изменение в версии Python 3.x, касающееся генерации исключений, в действительности не является изменением; на самом деле оно даже не имеет никакого отношения к решению проблемы перехода, связанной с версией Python 2.6. Синтаксис версии Python 3, касающийся генерации исключений (и создающий дополнительную причину для соответствующего исключения), выглядит так:

```
raise ValueError('Invalid value')
```

Давние пользователи языка Python наверняка использовали следующую идиому (хотя оба подхода поддерживаются во всех выпусках версии 2.x):

```
raise ValueError, 'Invalid value'
```

Для того чтобы подчеркнуть, что генерация исключений эквивалентна созданию экземпляров класса исключений и обеспечению некой дополнительной гибкости, версия Python 3 поддерживает лишь первую идиому. Хорошо уже то, что вам не придется ждать, пока вы перейдете на версию 2.6, чтобы приступить к использованию этого метода, — как указывалось в приложении В, этот синтаксис фактически допустим еще с появления версии Python 1.x.

Г.7. Другие инструменты перехода и рекомендации

В дополнение к версии Python 2.6 разработчики располагают доступом к определенному массиву инструментов, которые могут облегчить им переход к версии Python 3.x. В частности, в этом массиве предусмотрены такие инструменты, как переключатель `-3` (который обеспечивает выдачу предупреждений об устаревших средствах языка) и инструмент `2to3` (подробнее о нем можно прочитать на сайте <http://>

docs.python.org/3.0/library/2to3.html). Однако самым важным инструментом, который вы можете “написать”, является эффективный план перехода. Вообще говоря, равноценной замены планированию просто не существует.

Очевидно, что изменения в версии Python 3.x не вылились в некую безудержную мутацию хорошо знакомого синтаксиса языка Python. Напротив, переменные оказались вполне достаточными, чтобы преодолеть старую базу кодирования. Разумеется, эти переменные не могли не сказаться на пользователях, поэтому я не зря подчеркнул важность составления эффективного плана перехода. Большинство эффективных планов подкрепляются инструментами и средствами, призванными помочь вам в осуществлении перехода. В рекомендациях по переносу, изложенных в документе “What’s New in Python 3.0”, особо подчеркивается важность хорошего тестирования кода в дополнение к использованию ключевых инструментов. Ниже кратко перечислена *суть* того, о чем подробно рассказывается на сайте <http://docs.python.org/3.0/whatsnew/3.0.html#porting-to-python-3-0>:

1. (Необходимое условие.) Начните с исчерпывающего описания теста.
2. Выполните перенос на версию Python 2.6. Это потребует не большего объема работы, чем в среднем перенос с версии Python 2.x на версию Python 2.(x+1). Убедитесь, что все ваши тесты проходят.
3. (Все еще используете версию 2.6.) Включите переключатель командной строки `-3`. Это позволяет активизировать выдачу предупреждений о возможностях, которые были изъяты из версии Python 3.0 или изменены в этой версии. Выполните еще раз комплект своих тестов и исправьте код, который генерирует предупреждения. Убедитесь еще раз, что все ваши тесты проходят.
4. Проведите трансляцию `2to3` (“исходный в исходный”) по своему дереву исходного кода. Выполните результат этой трансляции под Python 3.0. Устраните вручную любые оставшиеся проблемы и продолжайте до тех пор, пока все тесты не пройдут еще раз.

Еще одной альтернативой, которую следует рассмотреть, является инструмент `3to2`. Как нетрудно догадаться по названию этого инструмента, он выполняет действие, противоположное `2to3`: использует код для версии Python 3 и пытается создать работоспособный эквивалент для версии Python 2. Эта библиотека сопровождается неким сторонним разработчиком и не является частью стандартной библиотеки; однако она является весьма интересной альтернативой, поскольку побуждает людей кодировать для версии Python 3 как основного инструмента разработки, а это, согласитесь, не так уж плохо. Более подробные сведения об инструменте `3to2` представлены на сайте <http://pypi.python.org/pypi/3to2>.

Третьей альтернативой перенос вообще не является: вместо этого нужно, для начала, написать код, который выполняется как в версии 2.x, так и в 3.x (без каких-либо изменений в исходном коде). Возможно ли это?

Г.8. Написание кода, совместимого как с версией 2.x, так и с версией 3.x

Поскольку мы находимся на стадии перехода от версии Python 2 к версии Python 3, у вас может возникнуть вопрос: нельзя ли написать код, который выполнялся бы без модификации и на Python 2, и на Python 3? Такое требование представляется вполне

разумным, но с чего следует начать? Что приводит в негодность большую часть кода Python 2 при его выполнении каким-либо интерпретатором версии 3.x?

Г.8.1. Операция `print` или функция `print()`?

Если ход ваших мыслей совпадает с моим, то вы, наверное, ответили бы на вопрос, поставленный в предыдущем разделе, так: конечно же, оператор `print`. Это является для нас вполне подходящей отправной точкой, а потому воспользуемся ею. Проблема заключается в том, что в версии 2.x `print` является оператором, т.е. в версии 2.x `print` является ключевым или зарезервированным словом, тогда как в версии 3.x `print` — встроенной функцией. Иными словами, поскольку в этом случае речь идет о синтаксисе языка, вы не можете использовать операторы `if`, а в языке Python все еще нет макроса `#ifdef`.

Давайте попробуем заключить аргументы `print` в круглые скобки:

```
>>> print('Hello World!')
Hello World!
```

Здорово! Этот прием срабатывает как в среде Python 2, так и в Python 3. Проблема решена? К сожалению, не вполне.

```
>>> print(10, 20) # Python 2
(10, 20)
```

На этот раз вам не повезло, поскольку прежний вариант представляет собой кортеж, тогда как в версии Python 3 вы передаете функции `print()` несколько аргументов:

```
>>> print(10, 20) # Python 3
10 20
```

Обдумаем эту проблему еще раз. Возможно, мы можем выяснить, является ли `print` ключевым словом. Возможно, вы помните о существовании модуля `keyword`, который содержит список ключевых слов. Поскольку в версии 3.x `print` не будет ключевым словом, вам может прийти в голову следующее простое решение:

```
>>> import keyword
>>> 'print' in keyword.kwlist
False
```

Будучи умелым программистом, вы, наверное, проверили бы этот вариант в версии 2.x, рассчитывая получить в ответ `True`. Несмотря на то что вы рассуждаете в целом правильно, неудача постигнет вас по другой причине:

```
>>> import keyword
>>> if 'print' in keyword.kwlist:
...     from __future__ import print_function
...
File "<stdin>", line 2
SyntaxError: from __future__ imports must occur at the beginning of
the file
```

Один из вариантов работоспособного решения требует от вас использовать какую-либо функцию, которая обладает такими же возможностями, как `print`. Одной из таких функций является `sys.stdout.write()`, а еще одним решением — функция `distutils.log.warn()`. Так уж случилось, что во многих главах этой книги мы

решили использовать последний из перечисленных вариантов. Наверное, вариант `sys.stderr.write()` также работает, если вы используете вывод без буферизации.

В этом случае пример “Hello World!” выглядел бы так:

```
# Python 2.x
print 'Hello World!'
# Python 3.x
print('Hello World!')
```

Приведенная ниже строка работала бы в обеих версиях.

```
# Python 2.x & 3.x compatible
from distutils.log import warn as printf
printf('Hello World!')
```

Это напоминает мне, почему мы не использовали `sys.stdout.write()`. В этом случае нам пришлось бы добавить в конце строки символ NEWLINE, чтобы согласовать поведение:

```
# Python 2.x & 3.x compatible
import sys
sys.stdout.write('Hello World!\n')
```

Реальная проблема заключается отнюдь не в этом маленьком неудобстве, а в том, что в данном случае эти функции не являются истинным “заместителем” `print` или `print()`; они работают только тогда, когда вам удастся обойтись лишь одной строкой, представляющей ваш вывод. Все, что является более сложным, требует больших усилий с вашей стороны.

Г.8.2. Импортируйте в решение свой способ

В других ситуациях жизнь оказывается несколько проще, и вы можете просто *импортировать* правильное решение. В приведенном ниже коде мы хотим импортировать функцию `urlopen()`. В версии Python 2 эта функция находится в библиотеке `urllib` или `urllib2` (мы воспользуемся `urllib2`), а в версии Python 3 она интегрирована в модуль `urllib.request`. Ваше решение, которое годится как для версии 2.x, так и для 3.x, в этом случае выглядит достаточно элегантно и просто:

```
try:
    from urllib2 import urlopen
except ImportError:
    from urllib.request import urlopen
```

В целях более экономного расходования оперативной памяти вы, возможно, предпочли бы воспользоваться какой-либо хорошо известной версией встроенного итератора (Python 3), например `zip()`. В версии Python 2 подходящей версией итератора является `itertools.izip()`. В версии Python 3 функция `itertools.izip()` заменяет собой `zip()` и использует его имя. Если вы настаиваете именно на этой версии итератора, ваш оператор импортирования также оказывается достаточно простым:

```
try:
    from itertools import izip as zip
except ImportError:
    pass
```

Одним из примеров, который выглядит не столь уж элегантно, является класс `StringIO`. В версии Python 2 версия, написанная на чистом языке Python, заключена в модуль `StringIO`, а это означает, что вы можете обратиться к ней посредством модуля `StringIO.StringIO`. Существует также более быстрая действующая версия на языке C, которая содержится в модуле `cStringIO.StringIO`. В зависимости от вашей инсталляции среды Python вы можете сначала отдать предпочтение варианту `cStringIO`, но вернуться к модулю `StringIO`, если окажется, что у вас в распоряжении нет модуля `cStringIO`.

В версии Python 3 Юникод является типом строки, предусмотренным по умолчанию, но если вы работаете с сетями, то вам, наверное, придется работать не со строками Юникод, а со строками ASCII или строками байтов, поэтому вместо модуля `StringIO` придется использовать модуль `io.BytesIO`. Для того чтобы добиться требуемого результата, импортирование будет выглядеть несколько более неуклюже:

```
try:
    from io import BytesIO as StringIO
except ImportError:
    try:
        from cStringIO import StringIO
    except ImportError:
        from StringIO import StringIO
```

Г.8.3. Составление полной картины из отдельных фрагментов

Если вам повезет, то описанных выше изменений будет вполне достаточно, а остальной ваш код должен оказаться проще, чем начальный фрагмент кода. Если вы инсталлируете импорты `distutils.log.warn()` [как `printf()`], `url*.urlopen()`, `*.StringIO` и обычный импорт `xml.etree.ElementTree` (версии 2.5 и более современные), то можете написать очень короткий синтаксический анализатор, чтобы отобразить главные новости из службы Google News. Этот синтаксический анализатор будет состоять примерно из восьми строк кода:

```
g = urlopen('http://news.google.com/news?topic=h&output=rss')
f = StringIO(g.read())
g.close()
tree = xml.etree.ElementTree.parse(f)
f.close()
for elmt in tree.getiterator():
    if elmt.tag == 'title' and not \
        elmt.text.startswith('Top Stories'):
        printf('- %s' % elmt.text)
```

Этот сценарий действует одинаково в версиях 2.x и 3.x, без каких-либо изменений кода. Разумеется, если вы используете версию 2.4 или какую-либо из более старых версий, вам придется загрузить `ElementTree` отдельно.

Фрагменты кода, приведенные в этом подразделе, заимствованы из главы 14, поэтому, чтобы увидеть соответствующую полную версию в действии, взгляните на файл `goognewsrss.py`.

Кому-то покажется, что эти изменения не оставляют камня на камне от элегантности вашего исходного кода на языке Python. В конце концов, читабельность кода тоже кое-что значит! Если вы предпочитаете сохранить четкость своего кода и в то

же время написать код, способный работать без изменений как под версией 2.x, так и под 3.x, обратите внимание на пакет `six`.

Пакет `six` — это библиотека обеспечения совместимости, основная функция которой заключается в обеспечении интерфейса, позволяющего сохранить код *вашего* приложения неизменным, скрывая при этом от разработчика сложности, описанные в этом подразделе. Подробнее о пакете `six` можно прочитать на сайте <http://packages.python.org/six>.

Используете ли вы библиотеку `six` или решаете задачу обеспечения совместимости самостоятельно, мы надеялись показать в этом кратком материале, что нет ничего невозможного в том, чтобы написать код, способный выполняться как в версии 2.x, так и в 3.x. Вывод из сказанного выше заключается в том, что вам, возможно, придется в какой-то мере пожертвовать элегантностью и простотой своего исходного кода на языке Python, добившись взамен этого истинной мобильности “2to3”. Я уверен, что нам еще не раз придется возвращаться к этому вопросу в течение ближайших двух-трех лет, прежде чем мы полностью перейдем на новое поколение языка Python.

Г.9. Резюме

Мы являемся свидетелями значительных перемен, связанных с переходом на новое поколение языка Python. Сложности, сопутствующие этим переменам, объясняются лишь тем, что код версии 3.x несовместим с прежними выпусками Python. Однако как бы значительны ни были эти перемены, они не требуют от программистов совершенно новых способов мышления. Между тем код требует существенной переработки. Чтобы облегчить разработчикам процесс перехода, текущий и будущие выпуски оставшихся интерпретаторов версии 2.x будут включать средства обратного переноса возможностей версии 3.x.

Версия Python 2.6 является первым из интерпретаторов “двойного режима”. С помощью этого интерпретатора вы можете приступить к программированию на основе версии 3.x. Версия Python 2.6 и более новые выпуски способны работать со всем программным обеспечением версии 2.x, а также понимать значительную часть кода версии 3.x. (Нынешняя цель заключается в том, что версия 2.7 должна стать последним выпуском в серии 2.x. За получением более подробной информации о фиктивном выпуске Python 2.8 обращайтесь к документу PEP 404 на сайте <http://www.python.org/dev/peps/pep-0404>.) Таким образом, финальные выпуски версии 2.x помогают упростить задачу переноса и процесс миграции. Они облегчат вам переход к программированию на следующем поколении языка Python.

Предметный указатель

A

Amazon Web Services, 608
API для баз данных, 287
 JDBC, 287
 ODBC, 287

E

Elastic Compute Cloud, 608

G

Google App Engine, 606
Google Cloud Storage, 608

S

Simple Storage System, 608

T

Twitter, 684

A

Авторизация, 571; 688

Агент

 MTA, 142; 143

 Exim, 144

 Lotus Notes Domino, 144

 Microsoft Exchange, 144

 Postfix, 143

 qmail, 144

 Sendmail, 143

 MUA, 147

Адаптер, 278

 MySQLdb, 288

 PoPy, 290

 Pycopg, 290

 PyGreSQL, 290; 291

 PyMongo, 327

 pysqlite, 291

 psycopg, 290; 291

 PyPgSQL, 290; 291

Аппаратная инфраструктура, 611

Архитектура

 “клиент–сервер”, 238

 “клиент–сервер”, 84

Аутентификация, 418; 571

Б

База данных

 NoSQL, 326

 нереляционная, 287

 Bigtable, 287

 Cassandra, 287

 CouchDB, 287

 MongoDB, 287; 326

 Redis, 287

 SimpleDB, 287

 Tokyo Cabinet, 287

 реляционная, 286

 Embarcadero Interbase, 287

 Gadfly, 287

 IBM DB2, 286

 IBM Informix, 286

 Ingres, 287

 Microsoft SQL Server, 286

 MySQL, 287

 Oracle, 286

 PostgreSQL, 287

 SAP, 287

 SQLite, 287

 Sybase, 286

Библиотека

 httplib2, 685

 imaplib, 153

 JavaScript, 501

 NNTP, 135

 Pango, 264

 poplib, 149

 simplejson, 685

 smtplib, 156

 Tile, 266

 Tix, 259

 Tk, 265

 Tkinter, 236

 Twisted, 116

 Twython, 573; 685

 oauth2, 685

 Swing, 737

 Tweezy, 685

Блокировка

 глобальная, 185

 простая, 189

 со взаимным исключением, 189

 примитивная, 189

Брандмауэр, 408

В

Веб-администрирование, 611

Веб-клиент, 423

Веб-платформа, 500

 Django, 502

 Django-nonrel, 599

 OAuth, 599

 virtualenv, 599

 Twitter Developers, 599

Веб-программирование, 410

Веб-сервер, 440

 BaseHTTPServer, 445; 494

 CGIHTTPServer, 445; 495

 http.server, 445; 495

 SimpleHTTPServer, 445; 495

 базовый, 440

 wsgiref, 445; 495

Виртуальная машина Python, 185

Виртуальный канал, 90

Внешний процесс, 486
 Выражение-генератор, 139
 Выход
 из потока, 186
 из процесса, 186
 Вычисления
 облачные, 606

Г

Главный цикл интерпретатора, 185
 Графический элемент, 239
 дочерний, 240
 родительский, 240
 Графический элемент Tk, 242
 Button, 242
 Canvas, 242
 Checkbutton, 242
 Entry, 242
 Frame, 242
 Label, 242
 LabelFrame, 242
 Listbox, 242
 Menu, 242
 Menubutton, 242
 Message, 242
 Radiobutton, 242
 Scale, 242
 Scrollbar, 242
 Spinbox, 242
 Text, 242
 Toplevel, 242
 PanedWindow, 242
 Группирование, 47; 55

Д

Двоичный семафор, 189
 Дейтаграмма, 91
 Декоратор, 207
 Демон, 195
 Дескриптор привязки, 431
 Диспетчер геометрии, 241
 Grid, 241
 Packer, 241
 Placer, 241
 Диспетчеризация
 динамическая, 344
 статическая, 344
 Домен, 88

З

Замыкание Клини, 44
 Запрос, 275; 454
 Захват, 40

И

Импорт, 393
 Инструментарий
 FOX, 270
 GTK+, 261
 Microsoft MFC, 270
 OpenGL, 270
 PyGTK, 257
 Sun Microsystems Java/Swing, 270
 Tile, 265

Tix, 257
 wxPython, 257
 FLTK, 270
 Pmw, 257
 Tk, 236
 Инструмент
 2to3, 783
 3to2, 795
 Интернет, 407
 Интернет-клиент, 122
 Интернет-программированием, 410
 Интерпретатор
 Groovy, 611
 JRuby, 611
 Jython, 611; 736
 Quercus, 611
 Scala, 611
 Rhino, 611
 Интерфейс
 CGI, 452
 WSGI, 487
 общего шлюза, 452; 453
 шлюза веб-сервера, 452
 API
 Channel, 615
 Datastore, 615
 Email, 615
 Images, 658
 Memcache, 615
 Twitter, 685
 URLfetch, 615
 XMPP, 615
 Инфраструктура
 Concurrency, 116
 Twisted, 111
 Инфраструктура-как-служба, IaaS, 608
 Исключение, 793

К

Класс
 Binary, 285
 BINARY, 285
 CharField, 514
 Connection, 282
 DAATETIME, 285
 Date, 285
 DateFromTicks, 285
 django.db.models.Model, 514
 ftplib.FTP, 126
 html5lib, 435
 HTMLParser, 430
 imaplib.IMAP4, 154
 imaplib.IMAP4*, 153
 imaplib.IMAP4_SSL, 154
 imaplib.IMAP4_stream, 154
 LMTP, 145
 ModelForm, 552
 nntplib.NNTP, 134
 NUMBER, 285
 poplib.POP3, 149; 150
 poplib.POP3_SSL, 149
 ROWID, 285
 smtplib.SMTP, 145
 TimestampFromTicks, 285
 Time, 285

- TimeFromTicks, 285
- нового стиля, 792
- традиционный, 792
- Cursor, 283
- SMTP_SSL, 145
- STRING, 285
- Timestamp, 285
- Класс Connection
 - close(), 282
 - commit(), 282
 - cursor(), 282
 - errorhandler(), 282
 - rollback(), 282
- Класс cursor
 - arraysize, 283
 - close(), 284
 - connection, 283
 - description, 284
 - executemany(op, args), 284
 - fetchmany(), 284
 - fetchone(), 284
 - __iter__(), 284
 - next(), 284
 - nextset(), 284
 - rowcount, 284
 - setinputsizes(), 284
 - callproc(func[, args]), 284
 - execute(op [, args]), 284
 - fetchall(), 284
 - lastrowid, 284
 - setoutputsize(), 284
- Класс FTP
 - cwd(), 126
 - delete(), 126
 - dir(), 126
 - login(), 126
 - mkd(), 126
 - nlst(), 126
 - pwd(), 126
 - quit(), 126
 - rename(), 126
 - retrbinary(), 126
 - retrlines(), 126
 - rmd(), 126
 - storbinary(), 126
 - storlines(), 126
- Класс IMAP4
 - close(), 155
 - fetch(), 155
 - login(), 155
 - logout(), 155
 - search(), 155
 - select(), 155
 - noop(), 155
- Класс LockType
 - acquire(), 189
 - locked(), 190
 - release(), 190
- Класс NNTP
 - article(), 134
 - body(), 134
 - group(), 134
 - last(), 134
 - next(), 134
 - post(), 134
 - quit(), 134
 - xhdr(), 134
 - head(), 134
 - stat(), 134
- Класс POP3
 - delete(), 151
 - list(), 150
 - quit(), 151
 - retr(msgnum), 150
 - stat(), 150
 - user(), 150
 - pass (passwd), 150
- Класс SMTP
 - helo(), 145
 - login(), 145
 - quit(), 145
 - sendmail, 145
 - starttls(), 145
 - ehlo(), 145
 - set_debuglevel(), 145
- Класс Thread, 195
 - Daemon, 196
 - getName(), 196
 - __init__, 196
 - isAlive(), 196
 - isDaemon(), 196
 - join(), 196
 - run(), 196
 - setDaemon(), 196
 - setName(name), 196
 - start(), 196
 - Ident, 196
 - name, 196
- Клиент, 84
 - COM, 341
 - TCP, 97
 - UDP, 103
- Командный интерпретатор, 518
- Компоненты URL
 - frag, 411
 - host, 411
 - net_loc, 411
 - params, 411
 - passwd, 411
 - path, 411
 - port, 411
 - prot_sch, 411
 - query, 411
 - user, 411
- Концепция
 - частичного применения функций, 248
- Ключ по реби еля, 688
- Компиляция, 391
- Конечная точка связи, 87
- Консоль администрирования, 612
- Крон-задание, 668
- Курсор, 275
- M**
- Маркер доступа, 688
- Массив
 - JSON, 713
- Межпроцессное взаимодействие, 88; 184
 - прозрачное, 89
- Ме асимвол, 39

Метод

- group(), 49; 51
- groupdict(), 49
- groups(), 49; 51
- search(), 53
- split(), 61
- сокета
 - accept(), 92
 - bind(), 92
 - close(), 93
 - connect(), 92
 - connect_ex(), 92
 - detach(), 93
 - f leno(), 93
 - getpeername(), 93
 - getsockname(), 93
 - getsockopt(), 93
 - gettimeout(), 93
 - ioctl(), 93
 - listen(), 92
 - makefile(), 93
 - recv(), 92
 - recvfrom(), 92
 - recvfrom_into(), 92
 - recv_into(), 92
 - send(), 92
 - sendall(), 92
 - sendto(), 92
 - setblocking(), 93
 - setsockopt(), 93
 - settimeout(), 93
 - shutdown(), 93

Микроблогинг, 684

Модель данных, 579

Модель

DOM, 717

Модуль

- APSW, 332
- async*, 116
- asynchat, 116
- asyncore, 116
- binascii, 172
- binhex, 172
- Boa Constructor, 269
- CGIHTTPServer, 441
- cgitb, 444; 455; 494
- Cookie, 444; 494
- cookielib, 444; 494
- CouchDB, 333
- couchdb-python, 333
- csv, 711
- csva, 730
- cx_Oracle, 333
- DB-API, 280
- DCOracle2, 333
- DocXMLRPCServer, 445; 730
- EasyGUI, 269
- email, 156; 171
- eric, 270
- Firebird (InterBase), 332
- ftplib, 172
- FXPy, 270
- Gadfly, 332
- GNOME-Python, 269
- GTK+, 263

- htmlentitydefs, 444; 494
- htmllib, 444; 494
- HTMLparser, 444; 494
- HTMLParser, 430
- httplib, 172; 418; 444; 494; 668
- http.server, 441
- imaplib, 153; 172; 418
- ingresmod, 333
- Ingres DBI, 333
- json, 712
- jsonb, 730
- KInterbasDB, 332
- mailcap, 171
- marshal, 712
- MaxDB (SAP), 332
- mimetypes, 172
- MimeWriter, 172
- mimify, 172
- MongoDB, 333
- mutex, 232
- MySQL, 332
- MySQL Connector Python, 332
- MySQLdb, 332
- nntplib, 172
- Oracle, 333
- pickle, 712
- Pmw, 257; 269
- PostgreSQL, 332
- psycpg, 290; 332
- pyExceclerator, 372
- PyGreSQL, 332
- PyGTK, 257; 263; 269
- PyGUI, 270
- PyMongo, 333
- PyMongo3, 333
- pymssql, 332
- PyOpenGL, 270
- PyQt, 270
- PyQtGPL, 270
- pysqlite, 332
- PythonCard, 269
- Queue, 232
- Queue/queue, 225
- quopri, 172
- re, 48
- sapdb, 332
- sdb.dbapi, 332
- sdb.sql, 332
- select, 116
- sgmlib, 444; 494
- SimpleHTTPServer, 441
- SimpleXMLRPCServer, 445; 730
- smtplib, 171
- smtplib, 172; 418
- socket, 91; 116
- SocketServer, 107; 116; 232
- SQLite, 332
- sqlite3, 332
- SQL Server, 332
- subprocess, 229; 232
- sybase, 333
- Sybase, 332
- thread, 189; 232
- threading, 194; 232
- TIDE, 269

- Tile/Ttk, 265; 269
- Tix, 257; 259; 269
- Tkinter, 238; 269
- TkZinc, 269
- urlfetch, 668
- urllib, 413; 417; 444; 668
 - quote, 417
 - quote_plus, 417
 - un_quote, 417
 - unquote_plus, 417
 - urlencode, 418
 - urlopen, 417
 - urlretrieve, 417
- urllib2, 415; 444; 668
- urllib.error, 444
- urllib.parse, 444
- urllib.request, 415; 444
- urlparse, 412; 444
- webbrowser, 444; 494
- win32com.client, 343
- win32.constants, 349
- win32ui, 270
- wxPython, 257; 269
- wxWidgets, 261
- xlrd, 372
- xlwt, 372
- xml, 445
- xml.dom, 445
- xml.dom, 730
- xml.etree, 445
- xml, 445; 717
- xml.parsers.expat, 445
- xml.parsers.expatc, 730
- xmlrpc, 172; 445; 730
- xml.sax, 445
- xml.saxb, 731
- adodbapi, 332
- base64, 171
- BaseHTTPServer, 441
- cgi, 444; 454; 494
- concurrent.futures, 230; 232
- Ingres, 333
- mailbox, 171
- m lib, 171
- mimetools, 172
- multiprocessing, 229; 232
- poplib, 172; 418
- pyFLTK, 270
- PyKDE, 270
- PyPgSQL, 332
- PyUNO, 372
- robotparser, 444; 494
- swing, 270
- wxGlade, 269
- xml.etree.ElementTree, 731
- zxJDBC, 750
- Модуль DB-API
 - apilevel, 280
 - connect(), 280
 - paramstyle, 280
 - threadsafety, 280
- Модуль email
 - email.message_from_string(), 158; 159
 - make_img_msg(), 156
 - make_mpa_msg(), 156
 - message.get_payload(), 158
 - message.walk(), 158
 - msg.get_payload(), 159
 - msg.walk(), 159
 - part.get_content_type(), 159
- Модуль Queue/queue
 - empty(), 226
 - Empty, 225
 - full(), 226
 - get(), 226
 - get_nowait(), 226
 - join(), 226
 - LifoQueue, 225
 - put(), 226
 - put_nowait(), 226
 - qsize(), 225
 - Queue, 225
 - Full, 225
 - PriorityQueue, 225
 - task_done(), 226
- Модуль threading
 - active_count(), 202
 - Barrier, 194
 - BoundedSemaphore, 194
 - Condition, 194
 - enumerate(), 202
 - Event, 194
 - Lock, 194
 - Rlock, 194
 - Semaphore, 194
 - setprofile(), 202
 - settrace(), 202
 - stack_size(), 202
 - T read, 194
 - Timer, 194
 - currentThread(), 202
- Модуль urllib
 - geturl(), 415
 - quote(), 416
 - quote_plus(), 416
 - unquote_plus(), 417
 - urlencode(), 417
 - urlopen(), 414
 - urlretrieve(), 415
 - unquote(), 417
- Модуль urlparse
 - urljoin(), 413
 - urlparse(), 412
 - urlunparse(), 412
- Мьютекс, 189
- Н**
- Надстройка, 491
- О**
- Облако
 - гибридное, 607
 - общего пользования, 607
 - частное, 607
- Обратный вызов, 240
- Объект блокировки, 189
- Объектно-реляционный преобразователь, 306
 - Dejavu, 306
 - Durus, 306
 - ForgetSQL, 306

- PDO, 306
- PyDO/PyDO2, 306; 333
- QLime, 306
- SQLObject, 306; 320; 333
- SQLObject2, 333
- Storm, 306; 333
- SQLAlchemy, 306; 333
- Ответ, 454
- Отметка времени, 97
- Очередь, 659
- Объект
 - JSON, 713
- Окно
 - корневое, 239
 - верхнего уровня, 239; 241
- Оператор
 - замыкания, 44
 - подавления жадного поведения, 78
 - положительного замыкания, 44
 - чередования, 1, 40
- П**
- Паке
 - BeautifulSoup, 445; 495
 - deferred, 664
 - DocXMLRPCServer, 725
 - email, 156
 - html5lib, 495
 - HTMLgen, 445
 - lxml, 495
 - Mechanize, 445
 - SimpleXMLRPCServer, 725
 - six, 799
 - xml, 717
 - xmlrpc.client, 725
 - xmlrpclib, 725
 - xmlrpc.server, 725
- Пакет SQLAlchemy
 - all(), 314
 - delete(), 314
 - filter(), 314
 - first(), 314
 - join(), 314
 - limit(), 314
 - offset(), 314
 - order_by(), 314
 - update(), 314
 - filter_by(), 314
 - one(), 314
- Перегрузка, 44
- Песочница, 612
- Платформа-как-служба, PaaS, 608
- Платформа
 - bottle, 617
 - Django, 599; 617
 - Flask, 617
 - Google App Engine, 501
 - Google App Engine Oil, 617
 - JavaScript, 500
 - Kay, 617
 - Pinax, 599
 - POSIX, 507
 - Pyramid, 501; 618
 - Pyramid & Pylons, 599
 - tipfy, 618
 - TurboGears, 599
 - web2py, 618
 - webapp, 617
 - полнофункциональная, 501
 - GAE Framework, 617
 - MVCEngine, 617
- Поведение
 - событийно-управляемое, 240
- Повторение, 55
- Поиск, 38
- Поток, 184
 - ReplyThread, 183
 - UserRequestThread, 183
 - RequestProcessor, 183
- Проверка, 393
 - отрицательная
 - опережающая, 40
 - ретроспективная, 40
 - положительная
 - опережающая, 40
 - ретроспективная, 40
- Прокси-сервер, 409
 - обратный, 409
 - прямой, 409
- Прорисовка, 240
- Процесс, 184
 - тяжеловесный, 184
 - легковесный, 184
- Пи-поток, 186
- Порт
 - данных, 124
 - передачи команда, 124
 - управления, 124
- Приложение
 - Excel, 343
 - Outlook, 349
 - Word, 346
 - WSGI, 487
 - гибридное, 680
 - CGI, 454
 - PowerPoint, 347
- Приостановка, 184
- Принцип
 - DRY, 514
 - SaaS, 159
- Программирование
 - однопоточное, 187
 - многопоточное, 182
- Программное обеспечение-как-служба, SaaS, 608
- Протоколе
 - IPv6, 89
- Протокол
 - ESMTP, 143
 - FTP, 123
 - HTTP, 123; 407
 - IMAP, 148
 - IMAP4, 148; 153
 - IMAP4rev1, 148
 - JSON-RPC, 725
 - LMTP, 143
 - NNTP, 132
 - OAuth, 571
 - POP, 147
 - POP3, 149

- SMTP, 143
- SOAPjr, 725
- TCP, 90
- TIPC, 89
- UDP, 91
- UUCP, 123
- XML-RPC, 725
- Интернета, 90
- передачи файлов, FTP, 123
- пользовательских дейтаграмм, 91
- управления передачей, 90
- передачи сетевых новостей, NNTP, 132

Р

- Распознавание шаблонов, 38
- Регулярное выражение, 36
- Режим
 - асинхронный, 182
 - жадный, 44
 - нежадный, 44
 - синхронный, 182
- Расширение, 380

С

- Секрет маркера доступа, 688
- Секрет потребителя, 688
- Семафор, 220
- Сервер
 - UDP, 102
- Сетевой журнал, 684
- Сокет, 88
 - AF_INET, 89
 - AF_INET6, 89
 - AF_LOCAL, 88
 - AF_TIPC, 89
 - AF_UNIX, 88
 - BSD, 88
 - Linux, 89
 - TCP, 90
 - UDP, 91
 - Unix, 88
 - Беркли, 88
 - поточковый, 90
 - с установлением соединения, 90
 - файловый, 88
 - AF_NETLINK, 89
 - SOCK_DGRAM, 91
 - SOCK_STREAM, 90
 - сетевой, 88
- Сопоставление, 38
- Состояние состязания, 185
- Спин-блокировка, 198
- Схема базы данных, 275
- Семейство
 - протоколов, 88
 - сокетов, 88
 - адресов, 88
 - AF_INET, 89
 - AF_INET6, 89
 - AF_NETLINK, 89
 - AF_TIPC, 89
- Сервер, 84
 - Apache, 486
 - API, 485

- Memcached, 607
- TCP, 93; 101
- WSGI, 488
- Yahoo! Finance, 680
- базы данных, 86
- веб-сервер, 85
- оконный, 86
- эталонный, 489

Сеть

- доставки контента, 607
- социальная, 684
 - Facebook, 684
 - Google+, 685
 - MySpace, 684
 - Twitter., 685

Система

- POSIX, 66
- новостей Usenet, 131
- передачи сообщений, 142

Служба

- App Identity, 615
- Appstats, 615
- Blobstore, 615
- Capabilities, 615
- Channel, 615
- Cloud SQL, 615
- Conversion, 615
- Cron, 616; 668
- Datastore, 616
- Denial-of-Service, 616
- Download, 616
- Files, 616
- Google+, 740
- Images, 616
- Logs, 616
- Mail, 616
- MapReduce, 616
- Memcache, 616
- Namespaces, 616
- NDB, 616
- OAuth, 616
- OpenID, 616
- Pipeline, 616
- Prospective Search, 616
- Search, 616
- Socket, 616
- Task Queue, 616
- URLfetch, 616; 667
- WarmUp, 617
- XMPP, 617
- доменных имен, 142
- облачная, 159
- Backends, 615
- Cloud Storage, 615
- Matcher, 616
- Users, 617

- Событие, 240
- Спецификация
 - DB-API Python, 279
- Ссылка
 - заимствованная, 397
 - собственная, 397
- Строка, 792
 - бесформатная, 58

Т

Твит, 685
 Тикер, 680
 Точка привязки, 430
 Технология
 WSGI, 485
 Тип
 bytes, 792
 bytearray, 792

У

Указатель URL, 411
 структура, 575
 Уровень защищенного сокета, 407

Ф

Файл
 соо ie, 475
 настроек, 569
 Ферма, 409
 Форма Django, 550
 Формат
 CSV, 708
 JSON, 712
 Функция
 compile(), 48; 50
 findall(), 48; 58
 finditer(), 48; 58
 purge(), 49
 search(), 48
 split(), 49
 sub(), 49; 60
 subn(), 60
 жадная, 185
 match(), 48; 51
 модуля thread
 allocate_lock(), 189
 exit(), 189
 start_new_thread(), 189
 сокета
 create_connection(), 106
 fromfd(), 106
 getaddrinfo(), 106
 getdefaulttimeout(), 106
 getfqdn(), 106
 gethostbyaddr(), 106
 gethostbyname(), 106
 gethostbyname_ex(), 106
 gethostname(), 106
 getnameinfo(), 106
 getprotobyname(), 106
 getservbyname(), 106
 getservbyport(), 106

htonl(), 106
 htons(), 106
 inet_aton(), 106
 inet_ntoa(), 106
 inet_ntop(), 106
 inet_pton(), 106
 ntohl(), 106
 ntohs(), 106
 setdefaulttimeout(), 106
 socket(), 91; 106
 socketpair(), 106
 ssl(), 106

Ш

Шаблон
 Active Record, 311
 MTV, 518
 MVC, 518

Э

Электронная почта, 141

Ю

Юникод, 472

Я

Язык
 Cython, 399
 Javascript, 712
 JavaScript, 611
 Perl, 236
 PHP, 611
 Psyco, 399
 PyPy, 400
 Pyrex, 399
 Ruby, 236; 611
 SGML, 431
 Tcl, 236
 XML, 716
 Язык SQL
 вставка строки, 277
 обновление строки, 277
 создание базы данных, 276
 создание таблицы, 276
 удаление базы данных, 276
 удаление таблицы, 276
 использование базы данных, 276
 удаление строки, 277
 Языковая исполняющая среда, 611