



Бен Фрейн

HTML5 и CSS3

Разработка сайтов

для любых браузеров и устройств

Ben Frain

Responsive Web Design with HTML5 and CSS3

Learn responsive design using HTML5 and CSS3
to adapt websites to any browser or screen size

[PACKT]
PUBLISHING
BIRMINGHAM - MUMBAI

Бен Фрэйн

HTML5 и CSS3

**Разработка сайтов
для любых браузеров и устройств**



Москва • Санкт-Петербург • Нижний Новгород • Воронеж
Ростов-на-Дону • Екатеринбург • Самара • Новосибирск
Киев • Харьков • Минск

2014

ББК 32.988.02-018
УДК 004.738.5
Ф86

Бен Фрейн

Ф86 HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств. — СПб.: Питер, 2014. — 304 с.: ил.
ISBN 978-5-496-00185-4

Сегодня как никогда остро стоит проблема адаптивного веб-дизайна. Все больше планшетных компьютеров, смартфонов и даже телевизоров используется для выхода в Интернет. Разработчикам веб-страниц требуется принимать во внимание огромное разнообразие размеров экранов, а также учитывать особенности соответствующего пользовательского взаимодействия. Адаптивный веб-дизайн позволяет наилучшим образом отобразить содержимое сайтов на экранах устройств, используемых для просмотра. При этом веб-страницы будут хорошо смотреться на дисплеях не только современных устройств, но и тех, что появятся в ближайшее время.

Начните разрабатывать сайты в соответствии с новой методологией адаптивного веб-дизайна, благодаря чему они будут красиво отображаться на экранах любых размеров. Читайте эту книгу, попутно создавая и улучшая адаптивные веб-дизайны с использованием HTML5 и CSS3. Вы научитесь применять на практике новые технологии и методики, призванные стать инструментами будущего для веб-разработчиков клиентских приложений.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988.02-018
УДК 004.738.5

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-1849693189 англ.
ISBN 978-5-496-00185-4

Copyright © 2012 Packt Publishing
© Перевод на русский язык ООО Издательство «Питер», 2014
© Издание на русском языке, оформление ООО Издательство «Питер», 2014

Краткое содержание

Об авторе	17
Благодарности	18
О рецензентах	19
Предисловие	21
От издательства	25
Глава 1. Знакомство с HTML5, CSS3 и адаптивным веб-дизайном	27
Глава 2. Медиазапросы: поддержка разных областей просмотра	53
Глава 3. Использование «резиновых» макетов	79
Глава 4. HTML5 для адаптивных веб-дизайнов	115
Глава 5. CSS3: селекторы, типографика и цветовые режимы	151
Глава 6. Великолепная эстетика с использованием CSS3	187
Глава 7. CSS3-переходы, трансформации и анимации	219
Глава 8. Покорение форм с помощью HTML5 и CSS3.	245
Глава 9. Решение кросс-браузерных проблем с адаптивностью.	273

Оглавление

- Краткое содержание 5**
- Об авторе 17**
- Благодарности 18**
- О рецензентах 19**
- Предисловие 21**
 - Какие темы рассматриваются в этой книге 21
 - Что необходимо знать, прежде чем приступить к чтению 22
 - Целевая аудитория книги 22
 - Соглашения 22
 - Отзывы читателей 23
 - Опечатки 24
 - Нарушение авторских прав 24
 - Вопросы 24
- От издательства 25**

Глава 1. Знакомство с HTML5, CSS3 и адаптивным веб-дизайном	27
1.1. Почему важна поддержка смартфонов (а устаревших версий Internet Explorer — нет).	28
1.2. Бывают ли ситуации, когда адаптивный веб-дизайн не станет правильным выбором?	29
1.3. Определение адаптивного веб-дизайна	30
1.4. Почему следует отдавать предпочтение именно адаптивному веб-дизайну?	31
1.5. Примеры адаптивного веб-дизайна	31
Инструменты для тестирования веб-страниц в разных областях просмотра	32
Онлайн-источники вдохновения	39
1.6. Преимущества HTML5	40
Экономия времени и кода благодаря HTML5	40
Новые семантически значимые элементы тегов HTML5	41
1.7. CSS3 делает возможным адаптивный веб-дизайн и многое другое	42
Важный момент: применение CSS3 не приведет к каким-либо нарушениям!	43
Как CSS3 позволяет решать повседневные задачи, связанные с дизайном?	43
1.8. Смотри, мама, нет изображений!	46
1.9. Можно ли реализовать потенциал HTML5 и CSS3 уже сегодня?	49
1.10. Адаптивный веб-дизайн — это не чудодейственное средство	50
1.11. Как объяснить заказчикам, что сайты не должны выглядеть одинаково во всех браузерах	50
1.12. Резюме	52

Глава 2. Медиазапросы: поддержка разных областей просмотра	53
2.1. Медиазапросы можно использовать уже сегодня	53
2.2. Почему адаптивным веб-дизайнам необходимы медиазапросы?	54
Синтаксис медиазапросов	54
Что позволяют проверить медиазапросы?	57
Использование медиазапросов для изменения дизайна	58
Наилучший способ загрузки медиазапросов для адаптивных веб-дизайнов	58
2.3. Наш первый адаптивный веб-дизайн	59
Не пугайтесь, но наш дизайн имеет фиксированную ширину	59
Адаптивные веб-дизайны: делаем изображения как можно более экономичными	63
Обрезка содержимого в меньших по размеру областях просмотра	65
2.4. Как сделать так, чтобы современные мобильные браузеры не изменяли автоматически размер нашей страницы	67
2.5. Подстраиваем дизайн под области просмотра с разной шириной	71
2.6. В адаптивных веб-дизайнах первым всегда должно идти содержимое	72
2.7. Медиазапросы — это лишь часть решения	77
2.8. Резюме	77
Глава 4. Использование «резиновых» макетов	79
3.1. Фиксированные макеты не ориентированы на будущее	80
3.2. Почему для адаптивных веб-дизайнов необходимы пропорциональные макеты	80

3.3. Преобразование дизайна на основе фиксированного макета в дизайн на основе пропорционального макета	81
Формула для запоминания	81
Задание контекста для пропорциональных элементов.	83
Всегда важно помнить о контексте	90
3.4. Использование единиц em вместо пикселей для задания размеров верстки.	93
3.5. «Резиновые» изображения.	95
Как сделать так, чтобы изображения масштабировались относительно области просмотра	95
Приоритетные правила для конкретных изображений.	97
«Притормаживаем» «резиновые» изображения	99
Невероятно универсальное свойство max-width	100
3.6. Обеспечение разных по величине изображений для разных по размеру экранов	101
Установка Adaptive Images	102
Размещение фоновых изображений в другом месте	104
3.7. Где «резиновые» сетки и медиазапросы объединяются друг с другом.	106
3.8. Сеточные системы CSS.	107
3.9. Резюме	113
Глава 4. HTML5 для адаптивных веб-дизайнов	115
4.1. Какие части HTML5 можно использовать уже сегодня?	116
Большинство сайтов может быть написано на HTML5	116
Полизаполнения, прослойки и Modernizr.	116
4.2. Как писать HTML5-страницы.	117
Эффект экономии от использования HTML5	118

Разумный подход к написанию HTML5-разметки	119
Приветствуем великий тег <a>	120
Устаревшие HTML-функции.	120
4.3. Новые семантические HTML5-элементы	121
Элемент <section>	122
Элемент <nav>	122
Элемент <article>.	122
Элемент <aside>	123
Элемент <hgroup>	123
Элемент <header>	125
Элемент <footer>.	125
Элемент <address>	125
4.4. Практическое использование структурных элементов HTML5.	126
4.5. HTML5-семантика уровня текста.	132
Элемент 	132
Элемент	133
Элемент <i>.	133
Применение семантики уровня текста в нашей разметке	133
4.6. Обеспечение большей доступности для вашего сайта с помощью WAI-ARIA	135
4.7. Вложение мультимедиа	138
4.8. Добавление видео и аудио с использованием HTML5	139
Обеспечение альтернативных файлов-источников	141
Резервные варианты для устаревших браузеров	141
Теги <audio> и <video> работают почти одинаково	142

4.9. Адаптивное видео	142
4.10. Автономные веб-приложения	145
Вкратце об автономных веб-приложениях.	146
Делаем веб-страницы доступными в автономном режиме . .	146
Понятие файла манифеста	147
Автоматическое добавление страниц в кэш	148
О комментарии с указанием номера версии	148
Просмотр сайта в автономном режиме	149
Устранение неполадок с автономными веб-приложениями.	149
4.11. Резюме	150

Глава 5. CSS3: селекторы, типографика и цветовые режимы . . 151

5.1. Что CSS3 предлагает разработчикам клиентских приложений.	152
Поддержка CSS3 в Internet Explorer версии от 6 до 8.	152
Использование CSS3 для дизайна и разработки страниц в браузере	153
5.2. Анатомия CSS-правил	153
5.3. Префиксы поставщиков и их использование.	153
5.4. Легко реализуемые и полезные CSS3-трюки.	156
Множественные колонки CSS3 для адаптивных веб-дизайнов	156
Перенос слов	159
5.5. Новые CSS3-селекторы и их использование	160
Селекторы атрибутов CSS3.	160
Селекторы атрибутов CSS3 с совпадениями по подстроке	160
Практический пример из реальной жизни.	161

Структурные псевдоклассы CSS3	162
Селектор :last-child.	163
Селектор :nth-child.	166
Принцип работы :nth-правил	167
Селектор псевдокласса отрицания (:not)	170
Изменения в псевдоэлементах	171
5.6. Пользовательская веб-типографика	173
CSS-правило @font-face	173
Реализация веб-шрифтов с помощью @font-face.	174
5.7. Помогите — мои заголовки с применением CSS3-правила @font-face выглядят неаккуратно	178
5.8. Новые цветовые форматы CSS3 и альфа-прозрачность.	180
RGB-цвет	181
HSL-цвет	182
Значения резервных цветов для Internet Explorer версий 6, 7 и 8.	183
Альфа-каналы	183
5.9. Резюме	185
Глава 6. Великолепная эстетика с использованием CSS3	187
6.1. Создание теней, отбрасываемых текстом, с помощью CSS3.	188
Допустимые шестнадцатеричные, HSL- и RGB-значения цветов	188
Пикселы, единицы em или rem	189
Предотвращение отбрасывания текстом тени	190
Создание эффекта рельефного текста	192
Множественные тени, отбрасываемые текстом	193

6.2. Создание теней, отбрасываемых блочными элементами	193
Внутренняя тень.	194
Множественные тени	195
6.3. Фоновые градиенты	197
Линейные фоновые градиенты	197
Радиальные фоновые градиенты	201
Повторяющиеся градиенты.	204
6.4. Фоновые градиентные узоры	206
6.5. Кое-какие соображения насчет CSS3	208
6.6. Сводим воедино CSS3-свойства	210
6.7. Множественные фоновые изображения	214
Размеры фоновых изображений	215
Позиционирование фоновых изображений	216
Сокращенный способ определения фона	216
6.8. Прочие CSS3-свойства	216
6.9. Масштабируемые значки, идеально подходящие для адаптивных веб-дизайнов	216
6.10. Резюме	217

Глава 7. CSS3-переходы, трансформации и анимации. 219

7.1. Что такое CSS3-переходы и как мы можем их использовать	220
Свойства, используемые для объявления переходов.	221
Собирательное свойство transition	222
Применение разных по длительности эффектов перехода к различным свойствам.	223
Понятие временных функций	223
Занятные переходы для адаптивных сайтов	224

7.2. 2D-трансформации CSS3	225
Что можно трансформировать?	226
scale	227
translate	227
rotate	228
skew	228
matrix	229
Свойство transform-origin	230
7.3. Вкратце об обеспечении 3D-трансформаций CSS3	231
Анализ 3D-эффекта	233
3D-трансформации не готовы к повсеместному внедрению	236
7.4. Анимация с помощью CSS3	237
7.5. Резюме	244
Глава 8. Покорение форм с помощью HTML5 и CSS3	245
8.1. HTML5-формы	245
Понятие составных частей HTML5-форм	248
placeholder	248
required	248
autofocus	250
autocomplete	250
list (и ассоциированный элемент <datalist>)	251
Типы полей ввода HTML5	252
email	252
number	254
url	254

tel	256
search	256
pattern	258
color	258
Типы полей ввода date и time.	259
date	259
month	260
week	260
time	261
datetime и datetime-local.	261
range	263
8.2. Добавление полизаполнений для браузеров, не поддерживающих требуемые функции.	264
8.3. Стилизация HTML5-форм с помощью CSS3	266
8.4. Резюме	272
Глава 9. Решение кросс-браузерных проблем с адаптивностью	273
9.1. Прогрессивное улучшение против плавного сокращения возможностей	277
9.2. Следует ли вам заботиться о том, чтобы сайт работал в устаревших версиях Internet Explorer?	279
Статистика	279
Личный выбор	280
9.3. Modernizr — «швейцарский армейский нож» разработчика клиентских приложений	281
Устранение проблем со стилизацией с помощью Modernizr	283

Modernizr добавляет поддержку HTML5-элементов в устаревшие версии браузера Internet Explorer	285
Добавление поддержки медиазапросов min-width и max-width в Internet Explorer версий 6, 7 и 8	286
Условная загрузка с помощью Modernizr	288
9.4. Преобразование списка навигационных ссылок в раскрывающееся меню (условно).	290
9.5. Устройства с экранами высокого разрешения (будущее).	294
9.6. Резюме	297

Об авторе

Бен Фрейн (Ben Frain) — независимый веб-дизайнер и разработчик клиентских приложений с более чем десятилетним опытом, сотрудничающий с дизайнерскими фирмами со всего мира. Он также работает журналистом — его перу принадлежит множество публикаций, касающихся платформы Mac, перспективных технологий, дизайна сайтов и технологических систем в авиационной промышленности.

Ранее Бен был актером на телевидении, но не получил должного признания. В то же время он окончил Солфордский университет со степенью в области медиа и исполнительского искусства. Бен написал четыре в равной степени недооцененных (по его мнению) сценария и все еще надеется (хотя все меньше и меньше), что ему удастся продать хотя бы один из них. В свободное от работы время он любит играть в футбол, пока его тело (и жена) все еще позволяет ему это.

Вы можете посетить сайт www.benfrain.com, а также связаться с автором через Twitter по адресу twitter.com/benfrain.

Благодарности

Прежде всего выражаю признательность веб-сообществу. Без их совместных знаний и проявленного бескорыстия при документировании решений я не смог бы добиться тех результатов в веб-разработке, которыми горжусь.

Далее хочу поблагодарить основателя адаптивного веб-дизайна — Итана Маркотта (Ethan Marcotte). Это человек, с которым мне никогда не доводилось встречаться или общаться, но чья методология сейчас оказывает влияние на мой повседневный подход к созданию сайтов. Разумеется, все недостатки или ошибки в том, как я преподношу читателям адаптивную методологию, следует записать исключительно на мой счет.

И наконец, выражаю признательность моей семье. Любой, кто смотрел фильм «Уайетт Эрп» (тоже недооцененный), знает, что *«нет ничего важнее кровных уз. Все остальные — просто чужаки»*.

О рецензентах

Эд Хендерсон (Ed Henderson) — опытный веб-разработчик, который любит проектировать и создавать различные вещи в режиме онлайн.

Он не боится «испачкать руки и промочить ноги» и открыт для большинства технологий, если они полезны и увлекательны.

У Эда есть степень в области информатики, а также собственный бизнес (Web Man Walking). Ему доводилось работать фрилансером, на постоянной основе, по контракту, при этом он хорошо знает все тонкости веб-индустрии, начиная с веб-страниц и заканчивая веб-приложениями и социальными сетями.

Эд хорошо зарабатывает на генерации свежих идей. Он был программистом, разработчиком программного обеспечения, а теперь это выдающийся человек в веб-индустрии, который любит иметь дело со всем, что привлекает внимание, является интересным и перспективным. Умение добиваться положительных результатов и превращать идеи в полезные, работающие решения — основные достоинства Эда.

Скорее всего, вы не знаете, но Эд — папа из Jack Draws Anything (<http://jackdrawsanything.com/>), а также лауреат престижной награды Social Campaign of the Year 2011 («Социальная кампания 2011 года») от журнала .Net.

Эд вместе со своей прекрасной женой Роуз живет в Верхнем Кокензи, графство Ист-Лотиан, Шотландия, где также проживают его закадычные друзья Джек, Тоби и Ной и остальные члены команды Hendo.

Вы можете посетить сайт Эда по адресу <http://edhenderson.com> (Эд постоянно занят работой, так что извините его за беспорядок) или связаться с ним через Twitter: @edhenderson.

Мовис Ледфорд (Mauvis Ledford) — опытный веб-разработчик, специализирующийся на клиентской архитектуре. Он активно трудится в этой области на протяжении уже девяти лет, два последних года из которых сконцентрировал на разработке веб-приложений для мобильных устройств и HTML5.

У Мовиса есть собственная интернет-компания, оказывающая консультационные услуги и специализирующаяся на адаптивном веб-дизайне и веб-приложениях, поддерживающих повсеместное развертывание. Вы найдете ее по адресу <http://www.brainswap.me>. Кроме того, Мовис работал по контракту в Disney Mobile, Skype, Netflix и многих молодых компаниях в Сан-Франциско.

Камруджаман Шохел (Kamrujaman Shohel) — разработчик клиентских приложений и специалист в разнообразных областях. У него имеется серьезный опыт работы проектировщиком интерфейсов пользователя и клиентских приложений,

а также консультантом по обеспечению юзабилити. Окончив обучение в 2004 году, он начал свою карьеру как PHP-разработчик в SSR IT, а затем работал аналитиком в Multimode Group (подразделение Microsoft). Ему всегда нравилось проектирование клиентских приложений, поскольку эта область позволяла проявить креативность. В результате Камруджаман изменил свою карьеру и в январе 2005 года превратился в успешного разработчика клиентских приложений в Right Brain Solution Limited. Это прекрасный специалист по HTML, HTML5, CSS3, jQuery, jQuery UI, PHP, Photoshop CS5 и Illustrator CS5.

Последние два года Камруджаман работал старшим проектировщиком клиентских приложений (был главой команды) в Trenza Softwares, а также консультантом в сфере высоких технологий в Mesovison Consultancy Limited. Он любит исследовать дизайн интерфейсов, интерактивность, пользовательскую совместимость и функциональность высококлассных веб-приложений. Кроме того, он планирует написать книгу по HTML5, CSS3, jQuery, jQuery Mobile или jQuery UI. Камруджаман мечтает основать собственную компанию, в которой люди будут помогать друг другу развивать свои таланты.

Камруджаман постоянно работает (за исключением времени, отведенного на сон). Кроме того, он стремится регулярно освежать свои знания, читая технические книги и изучая вопросы проектирования клиентских приложений. Он прекрасно разбирается в PHP, C, C#, VB.NET, ASP.NET, CakePHP, Zend Framework, Drupal, Joomla! и WordPress. Будучи состоявшимся проектировщиком клиентских приложений, он считает, что ключ к совершенству — практика, поэтому постоянно улучшает свои навыки, используя в работе новые технологии.

Предисловие

Если вы решили, что вам необходимо создать мобильную версию вашего сайта, не торопитесь, подумайте еще! Адаптивный веб-дизайн позволит вам разработать единый вариант сайта, который будет отлично смотреться на экранах смартфонов, настольных компьютеров и другой подобной техники. Дизайн такого сайта будет легко приспосабливаться под определенный размер пользовательского экрана, обеспечивая наиболее качественное взаимодействие, возможное на устройствах как сегодняшнего, так и завтрашнего дня.

Эта книга — исчерпывающее практическое руководство по тому, как сделать адаптивным имеющийся дизайн с фиксированной шириной. Кроме того, в ней описываются новейшие и самые полезные методики, обеспечиваемые HTML5 и CSS3 и позволяющие сделать дизайн сайта более компактным и удобным в сопровождении, чем когда-либо прежде. В этой книге также разъясняются общие и наиболее эффективные методы написания и доставки кода, изображений и файлов.

Если вам понятны HTML и CSS, то вы сможете создавать адаптивные веб-дизайны.

Какие темы рассматриваются в этой книге

В главе 1 «Знакомство с HTML5, CSS3 и адаптивным веб-дизайном» поясняется, что такое адаптивный веб-дизайн, приводятся примеры таких дизайнов, а также подчеркиваются преимущества и эффект экономии от использования HTML5 и CSS3.

Из главы 2 «Медиазапросы: поддержка разных областей просмотра» вы узнаете, что такое медиазапросы, как их написать и применить к любому дизайну для адаптации CSS-стилей к возможностям устройств.

В главе 3 «Использование “резиновых” макетов» рассказывается о преимуществах «резиновых» макетов и демонстрируется, как можно с легкостью преобразовать текущий дизайн с фиксированной шириной в «резиновый» макет либо использовать CSS-фреймворк для быстрого создания адаптивных веб-дизайнов.

В главе 4 «HTML5 для адаптивных веб-дизайнов» исследуются преимущества написания кода на HTML5 (более компактный код, семантические элементы, автономное кэширование и WAI-ARIA для внедрения вспомогательных технологий).

В главе 5 «CSS3: селекторы, типографика и цветовые режимы» демонстрируется мощь CSS3-селекторов, позволяющих с легкостью преобразовывать все что угодно. Я также воспользуюсь CSS3-правилом `@font-face`, чтобы создать красивую

веб-типографику, и расскажу вам о таких новых цветовых режимах CSS3, как RGB(A) и HSL(A).

В главе 6 «Великолепная эстетика с использованием CSS3» рассказывается, как создавать тени, отбрасываемые текстом и блочными элементами, а также градиенты исключительно с помощью CSS3. Мы также поговорим о том, как добавлять разнообразные фоновые изображения и создавать значки с использованием шрифтов.

В главе 7 «CSS3-переходы, трансформации и анимации» я поведаю вам о том, как создавать, преобразовывать и анимировать экранные элементы, используя лишь CSS3.

В главе 8 «Покорение форм с помощью HTML5 и CSS3» иллюстрируется, как реализовать кросс-браузерные методики, касающиеся форм и работающие на всем, начиная от новейших смартфонов и заканчивая настольными браузерами.

В главе 9 «Решение кросс-браузерных проблем с адаптивностью» разъясняется, как обеспечить поддержку адаптивности в устаревших версиях Internet Explorer, как адаптировать тот или иной набор ссылок к меню на мобильных устройствах, как обеспечивать другое содержимое для устройств с экранами высокого разрешения и условно загружать ресурсы с помощью Modernizr.

Что необходимо знать, прежде чем приступить к чтению

Вам потребуется хорошо знать HTML и CSS. Кроме того, желательно иметь общее понятие о JavaScript. Хороший вкус в фильмах тоже может оказаться нелишним.

Целевая аудитория книги

Вы создаете два сайта — один для экранов мобильных устройств и еще один для более крупных мониторов? Или вы, возможно, слышали об адаптивном веб-дизайне, но точно не знаете, как соединить его с HTML5 и CSS3? Если так оно и есть, то эта книга обеспечит вас знаниями, необходимыми для того, чтобы перевести веб-страницы на следующий уровень, прежде чем это сделают все ваши конкуренты!

Издание предназначено для веб-дизайнеров и веб-разработчиков, которые в настоящий момент создают сайты с фиксированной шириной, используя HTML 4.1 и CSS 2.1. Здесь объясняется, как создавать адаптивные сайты с применением HTML5 и CSS3, приспосабливающиеся к любому размеру экранов устройств.

Соглашения

В этой книге вы увидите несколько текстовых стилей, которые позволят различать разные типы информации. Далее приведены примеры этих стилей и пояснение того, что они означают.

Фрагменты кода в тексте помечаются следующим образом: «HTML5 также допускает использование намного менее аккуратного синтаксиса, который тоже

будет считаться валидным. Например, `<script Src=js/jquery-1.6.2.js></script>` будет точно таким же валидным, как и предыдущий вариант».

Тот или иной блок кода помечается так:

```
<div class="header">
  <div class="navigation">
    <ul class="nav-list">
      <li><a href="#" title="Home">Главная</a></li>
      <li><a href="#" title="About">О нас</a></li>
    </ul>
  </div> <!-- конец navigation -->
</div> <!-- конец header -->
```

Если мы хотим обратить ваше внимание на определенную часть блока кода, то выделяем полужирным шрифтом соответствующие строки или элементы:

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 96%; /* Удерживание самого дальнего от центра элемента <div> */
}

#header {
  margin-right: 10px;
  margin-left: 10px;
  width: 97.9166667%; /* 940 × 960 */
}
```

Новые термины и важные слова выделяются полужирным шрифтом. Слова, которые вы будете видеть на экране, например в меню или диалоговых окнах, помечаются в тексте следующим образом: «Цвет навигационного меню не меняется с красного на черный, главная кнопка **These should have won** (Они должны были стать победителями) в области содержимого и кнопка **Full info** (Полная информация) во врезке отсутствуют, а все шрифты сильно отличаются от тех, что есть в графическом файле».



ПРИМЕЧАНИЕ

Предупреждения или важные примечания приводятся в таком блоке.



СОВЕТ

Советы и рекомендации даются в таком блоке.

Отзывы читателей

Отзывы читателей всегда приветствуются. Дайте нам знать, что вы думаете об этой книге: что вам понравилось, а что, возможно, нет. Отзывы читателей важны для нас, поскольку помогают создавать книги, из которых вы действительно сможете почерпнуть максимум знаний.

Общие отзывы присылайте нам по адресу feedback@packtpub.com, указав в теме сообщения название книги.

Если у вас есть тема, в которой вы являетесь экспертом и заинтересованы либо в написании книги по ней, либо в том, чтобы внести свой вклад в книгу по такой тематике, загляните в наше руководство для авторов по адресу www.packtpub.com/authors.

Опечатки

Несмотря на то что мы предприняли все усилия для того, чтобы обеспечить отсутствие ошибок в нашей книге, они все равно возможны. Если вы найдете ошибку в одной из наших книг, например в тексте или коде, то сообщите нам о ней — мы будем очень признательны. Сделав это, вы, возможно, избавите других читателей от разочарования и поможете нам улучшить последующие издания этой книги. Если вы обнаружите какие-либо опечатки, пожалуйста, сообщите нам о них, для чего зайдите по адресу <http://www.packtpub.com/support>, выберите соответствующую книгу, щелкните на ссылке **Errata submission form** (Форма для отправки сведений об опечатках) и введите информацию о найденных вами опечатках. Как только выявленные вами опечатки будут подтверждены, предоставленные вами сведения будут признаны корректными и эти опечатки будут размещены на нашем сайте либо добавлены в список найденных опечаток в разделе **Errata** (Опечатки), относящемся к соответствующей книге.

Нарушение авторских прав

Нарушение авторских прав в Интернете — это постоянная проблема, от которой страдает вся мультимедийная сфера. В издательстве Packt **очень серьезно** относятся к защите авторских прав и лицензий. Если вы столкнетесь в Интернете с нелегальными копиями наших книг в любой форме, пожалуйста, незамедлительно сообщите нам адрес или название сайта, чтобы мы могли предпринять соответствующие меры.

Просьба связаться с нами по адресу copyright@packtpub.com, указав в письме ссылку на материал, в легальности которого имеются сомнения.

Мы высоко ценим вашу помощь в защите наших авторов и нашей способности обеспечивать вас полезными книгами.

Вопросы

Вы можете связаться с нами по адресу copyright@packtpub.com, если у вас возникли вопросы касательно любого раздела книги, и мы сделаем все возможное, чтобы помочь вам найти ответы на них.

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты vinitski@minsk.piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства <http://www.piter.com> вы найдете подробную информацию о наших книгах.

1 Знакомство с HTML5, CSS3 и адаптивным веб-дизайном

До недавнего времени сайты создавались с фиксированной шириной, например, 960 пикселей с расчетом на то, что это обеспечит единообразие для всех конечных пользователей. Такая ширина не была слишком большой для экранов ноутбуков, а пользователи с мониторами высокого разрешения при просмотре веб-страниц попросту наблюдали много свободного пространства по бокам.

Однако сейчас получили распространение смартфоны. iPhone от компании Apple стал первым смартфоном, который обеспечил по-настоящему удобный просмотр веб-страниц, и многие другие последовали его примеру. В отличие от вчерашних устройств с маленькими сенсорными экранами, требовавшими от пользователей при работе с ними проворства чемпиона мира по игре в бильярд, современные телефоны позволяют комфортно путешествовать по Интернету. Кроме того, потребители все чаще используют дома устройства с небольшими дисплеями (например, планшетные компьютеры и нетбуки) вместо их полноэкранных собратьев. Бесспорным фактом является то, что количество людей, предпочитающих для путешествия по Интернету устройства с небольшими экранами, растет с постоянно увеличивающейся скоростью. В то же время многие люди по-прежнему пользуются мониторами с диагональю 27 и 30 дюймов. В настоящее время разница между просмотром веб-страниц на устройствах с самыми маленькими и самыми большими экранами велика как никогда.

К счастью, для этой постоянно расширяющейся среды браузеров и устройств появилось решение. Адаптивный веб-дизайн с использованием HTML5 и CSS3 позволит сайтам «просто работать» на множестве разных устройств с отличающейся диагональю экрана. А самое лучшее заключается в том, что реализация всех соответствующих методик не потребует прибегать к решениям на стороне сервера.

В этой главе мы сделаем следующее:

- рассмотрим важность поддержки устройств с небольшими экранами;
- дадим определение дизайну мобильных сайтов;

- дадим определение адаптивному веб-дизайну сайтов;
- взглянем на отличные примеры адаптивного веб-дизайна;
- изучим разницу между размерами областей просмотра и экранов;
- установим браузерные расширения для изменения областей просмотра и воспользуемся ими;
- применим HTML5 для создания более чистой и компактной разметки;
- применим CSS для решения распространенных сложных задач, связанных с дизайном.

1.1. Почему важна поддержка смартфонов (а устаревших версий Internet Explorer — нет)

Хотя к статистическим данным всегда следует относиться только как к ориентировочным, интересно отметить, что, согласно gs.statcounter.com, за 12 месяцев с июля 2010 по июль 2011 года доля использования мобильных браузеров от общего количества по всему миру увеличилась с 2,86 до 7,02 %. Все та же статистика свидетельствует, что доля использования Internet Explorer 6 упала с 8,79 до 3,42 %. Даже доля применения Internet Explorer 7 упала до 5,45 % по состоянию на июль 2011 года. Если заказчики часто просят вас: «Сделайте так, чтобы наш сайт работал в версиях Internet Explorer 6 и 7», то **справедливой реакцией на это с вашей стороны** может стать вопрос: «Возможно, нам стоит сосредоточить свои усилия на чем-то другом?» В настоящее время намного больше людей просматривают сайты на мобильных телефонах, чем на настольных компьютерах или ноутбуках, на которых установлена версия Internet Explorer 6 и 7.

Таким образом, увеличивается количество людей, использующих для путешествия по Интернету устройства с небольшими экранами, а браузеры на этих устройствах обычно без проблем справляются с обработкой существующих сайтов. Для этого они «сжимают» стандартный сайт, чтобы он уместился в видимой области (или **области просмотра**, если говорить технически правильным языком) на экране устройства. Затем пользователь сможет увеличить определенную область содержимого, которая его заинтересовала. Раз все так замечательно, то зачем нам, дизайнерам и разработчикам клиентских приложений, предпринимать еще какие-то действия?

Чем больше сайтов вроде показанного на рис. 1.1 вы просматриваете на смартфонах iPhone и Android, тем более очевидными становятся причины того, что нам потребуется предпринять дополнительные действия. Пользователям надоедает постоянно увеличивать и уменьшать области страниц, чтобы их было удобно просматривать, а затем прокручивать страницы влево и вправо, чтобы прочитать

предложения, не уместающиеся в области просмотра, и стараться при этом случайно не задеть ненужную ссылку. Мы, несомненно, можем предложить им кое-что получше!



Рис. 1.1. Пример веб-страницы на экране смартфона

1.2. Бывают ли ситуации, когда адаптивный веб-дизайн не станет правильным выбором?

Когда позволяет бюджет и того требует сложившаяся ситуация, по-настоящему мобильная версия сайта, возможно, будет предпочтительным вариантом. Такая версия сможет обеспечивать разное содержимое, оформление и взаимодействие

в зависимости от применяемого устройства, местоположения, скорости соединения и множества других переменных факторов, включая технические возможности соответствующего устройства. В качестве примера представьте сеть пиццерий. У нее может быть один стандартный сайт и его мобильная версия, обладающая функцией дополненной реальности, позволяющей найти пиццерию, взяв за основу ваше текущее местоположение, определяемое GPS. Такому решению требуется нечто большее, чем то, что может предложить адаптивный веб-дизайн.

Однако, хотя не каждый проект предполагает подобный уровень сложности, почти во всех случаях предпочтительным будет обеспечивать для пользователей адаптированное представление содержимого в зависимости от размера области просмотра их устройства. Например, на большинстве сайтов, несмотря на то что они преподносят пользователям однообразное содержимое, я бы менял способ того, как они это делают. На устройствах с маленькими экранами я, пожалуй, разместил бы менее важные элементы под основным содержимым, а в самом худшем случае вообще скрыл бы их из виду. Кроме того, я, возможно, изменил бы навигационные кнопки, чтобы приспособить их под нажатия пальцами, обеспечив удобное взаимодействие не только для тех пользователей, у которых есть возможность сделать точный щелчок кнопкой мыши! Верстка тоже должна масштабироваться ради удобочитаемости, позволяя пользователям читать текст без необходимости постоянно прокручивать страницу вправо-влево.

Кроме того, несмотря на стремление обеспечивать качественное обслуживание пользователей с небольшими областями просмотра, мы не хотим идти на какие-либо компромиссы касаясь дизайна в случае с теми из них, кто работает за стандартными ноутбуками и настольными экранами. Несмотря на наш подход «все включено», как насчет нескольких дополнительных улучшений для тех, у кого большие дисплеи, например, с разрешением 1900 пикселей и более по горизонтали? В общем, мне и, полагаю, вам тоже необходимо, чтобы дизайн реагировал на полный спектр размеров областей просмотра, которые могут использоваться.

1.3. Определение адаптивного веб-дизайна

Термин **адаптивный веб-дизайн** был придуман Итаном Маркоттом. В своей конструктивной статье на сайте A List Apart (<http://www.alistapart.com/articles/responsive-web-design/>) Итан объединил три существующие методики (макет гибкой сетки, гибкие изображения и медиазапросы) в единый подход и назвал его адаптивным веб-дизайном. Этот термин часто подразумевает то же самое, что и такие понятия, как «резиновый» дизайн, эластичный макет, «резиновый» макет, дизайн с непостоянными размерами, адаптивный макет, дизайн с поддержкой разных устройств и гибкий дизайн.

Это всего лишь несколько примеров! Вместе с тем мистер Маркотт и другие разработчики красноречиво доказывают, что истинная адаптивная методология на самом деле представляет собой нечто большее, чем простое изменение макета сай-

та в зависимости от размеров областей просмотра. Она призвана полностью изменить весь наш подход к веб-дизайну. Вместо того чтобы начинать с настольного сайта с фиксированной шириной, а затем уменьшать его и заново заливать содержимое на страницы при их отображении в меньших по размеру областях просмотра, нам следует предварительно создать дизайн для самых маленьких областей просмотра, а затем улучшать этот дизайн и содержимое для более крупных областей просмотра.



ОБ АДАПТИВНОМ ВЕБ-ДИЗАЙНЕ В ДВУХ СЛОВАХ

Если попытаться описать философию адаптивного веб-дизайна в двух словах, то я сказал бы, что это представление содержимого в наиболее доступном виде для любой области просмотра, используемой для его обозрения. С другой стороны, настоящий мобильный сайт потребует, если его интерфейс будет предполагать специфическое содержимое и функциональность в зависимости от устройства, применяемого для доступа к этому сайту. В подобных ситуациях мобильный сайт обеспечивает совершенно другое пользовательское взаимодействие, нежели его настольный аналог.

1.4. Почему следует отдавать предпочтение именно адаптивному веб-дизайну?

Адаптивный веб-дизайн будет регулировать поток содержимого страницы по мере изменения областей просмотра, однако это еще не все. Версия HTML5 предлагает больше функций, чем HTML 4, и ее более значимые семантические элементы лягут в основу нашей разметки. Медиазапросы CSS3 — важная составная часть адаптивного веб-дизайна, а дополнительные CSS3-модули позволят нам достичь невиданных ранее уровней гибкости. Мы уберем фоновую графику и сложные сценарии JavaScript, заменив их легковесными CSS3-градиентами, тенями, версткой, анимацией и трансформациями.

Перед тем как перейти к адаптивному веб-дизайну с использованием HTML5 и CSS3, взгляните на несколько примеров того, к чему следует стремиться. Кто уже успешно применяет в своей работе все эти новые адаптивные штучки и что мы можем почерпнуть из их опыта первопроходцев?

1.5. Примеры адаптивного веб-дизайна

Для полного тестирования ваших и чужих адаптивных веб-дизайнов сайтов потребуются отдельные системы, настроенные под каждое устройство и размер экрана. Несмотря на то что нет ничего лучше такого способа, основное тестирование может быть проведено простым изменением размеров окна браузера. Дополнением к такой методике станут разнообразные сторонние плагины и браузерные

расширения, которые показывают текущие размеры окна браузера или области просмотра в пикселах либо автоматически устанавливают для текущего окна или области просмотра размеры экрана по умолчанию (например, 1024 × 768). Это позволит вам легче разобраться в том, что происходит при изменении экран-ных областей просмотра.



ПРИВЫКЛИ К ПИКСЕЛАМ? ПРИДЕТСЯ ОТВЫКАТЬ!

Вам не стоит слишком привыкать к пикселям в качестве единиц измерения, поскольку во многих случаях мы будем использовать не их, а относительные единицы измерения (обычно это `em` и проценты). При рассмотрении того, как работают другие адаптивные веб-дизайны и где именно в них происходят изменения, нам понадобится удобная точка отсчета.

Инструменты для тестирования веб-страниц в разных областях просмотра

Пользователям Internet Explorer следует позаботиться о том, чтобы у них был инструмент Microsoft Internet Explorer Developer Toolbar. Его можно скачать по следующему URL-адресу: <http://www.microsoft.com/download/en/details.aspx?id=18359>.

Если вы используете Safari, то рекомендую вам Resize (<http://resizeSafari.com/>), хотя ResizeMe (http://web.me.com/aaronholla/Safari_Extensions/ResizeMe.html) похож на него и бесплатен.

Если же вы работаете в Firefox, то для этого браузера предназначено расширение Firesizer (<https://addons.mozilla.org/en-US/firefox/addon/firesizer/>), а пользователям Chrome следует обратить внимание на решение с метким названием Windows Resizer (<https://chrome.google.com/webstore/detail/kkelicaakdanhinjdeammilcgefonfh>).

Вы не фанат расширений? Есть альтернатива: я написал простую HTML-страницу для отображения текущих значений высоты и ширины области просмотра окна браузера. Используя JavaScript-библиотеку jQuery (<http://jquery.com>), эта страница извлекает текущие значения высоты и ширины области просмотра и выводит их на экране. Вы можете держать эту страницу открытой на дополнительной вкладке браузера, изменять размеры своего окна, а затем возвращаться на исследуемый сайт, чтобы узнать, как он «поживает». Очень простую страницу, позволяющую узнать размеры вашей текущей области просмотра, вы найдете по следующему URL-адресу: <http://benfrain.com/easily-display-the-viewport-size-of-your-page-for-responsive-designs/>.



РАЗМЕРЫ ОБЛАСТИ ПРОСМОТРА ИЛИ РАЗМЕРЫ ЭКРАНА?

Важно понимать, что размеры области просмотра и размеры экрана — это не одно и то же. Область просмотра относится к области содержимого в пределах окна браузера, включая панели инструментов, вкладки и т. д. Иными словами, она относится к области, в которой непосредственно отображается тот или иной сайт. Размеры экрана касаются физической области отображения на устройстве. Имейте в виду, что одни инструменты сообщают размеры, в которых также учтены такие браузерные элементы, как адресная строка, вкладки, окна для ввода поисковых запросов, а другие инструменты так не поступают. На рис. 1.2 действительные размеры области просмотра отображаются вверх справа (1156 × 921), в то время как плагин Firesizer показывает размеры окна внизу справа (1171 × 1023).



Рис. 1.2. Страница с отображенными размерами области просмотра и окна

Теперь вы вооружены всем необходимым для того, чтобы начать оценивать все достоинства адаптивного веб-дизайна. Откройте свой любимый браузер, запустите инструмент для определения размеров окон и взгляните на страницу <http://thinkvitamin.com/>.

Если вы будете смотреть на открывшуюся страницу в области просмотра шириной менее 1060 пикселей, то увидите макет, показанный на рис. 1.3.

Если же вы взглянете на сайт в области просмотра шире 930 пикселей, но уже 1060 пикселей, то увидите макет, показанный на рис. 1.4.

Вы заметили, что основная навигация сбоку от логотипа изменилась? Значки справа от основного содержимого оказались расположенными один под другим. Все идеально с точки зрения удобства пользования, и, что самое важное, ничто не выходит за границы экрана. Теперь взгляните, как все выглядит в области просмотра шириной менее 880 пикселей (рис. 1.5).

Верхний колонтитул остался без изменений, однако обратите внимание, что панель навигации справа стала еще более узкой; значки теперь располагаются так: два вверху и два внизу, в то время как текстовые блоки отрегулированы и текст заливается в них в соответствии с внесенными корректировками.

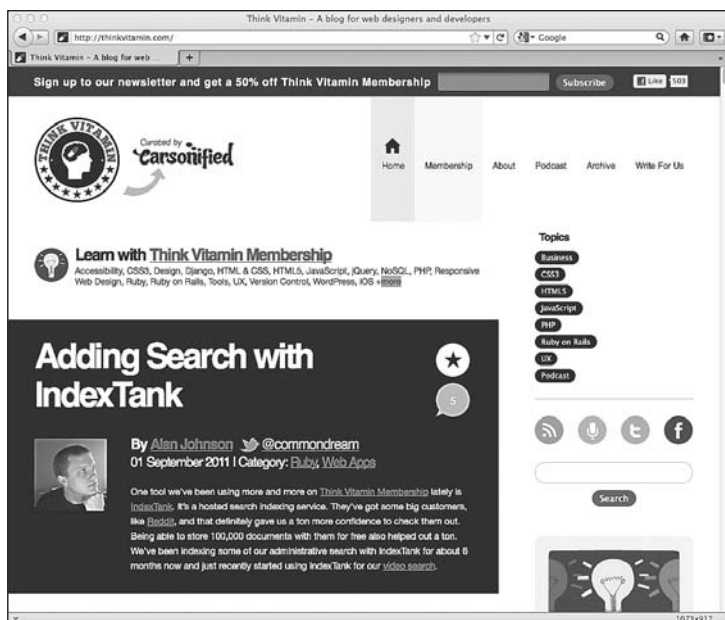


Рис. 1.3. Страница открыта в области просмотра шириной меньше 1060 пикселей

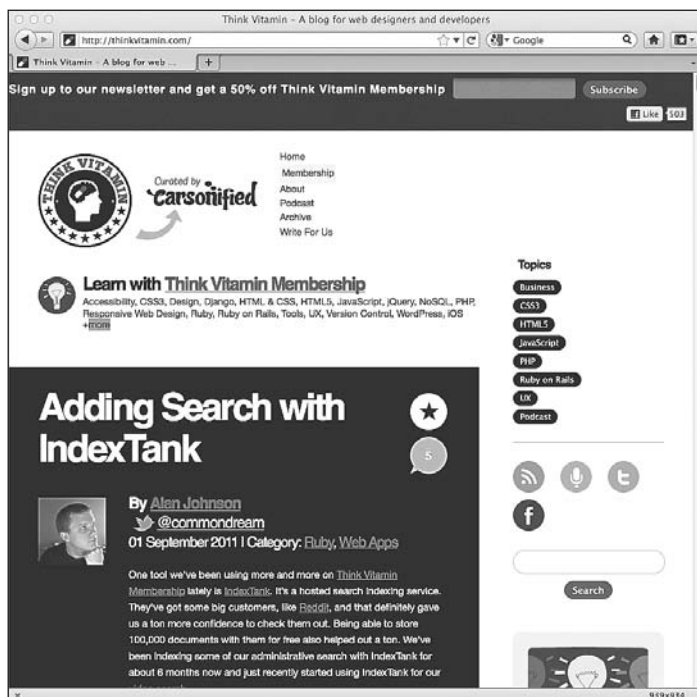


Рис. 1.4. Страница открыта в области просмотра шириной от 930 до 1060 пикселей

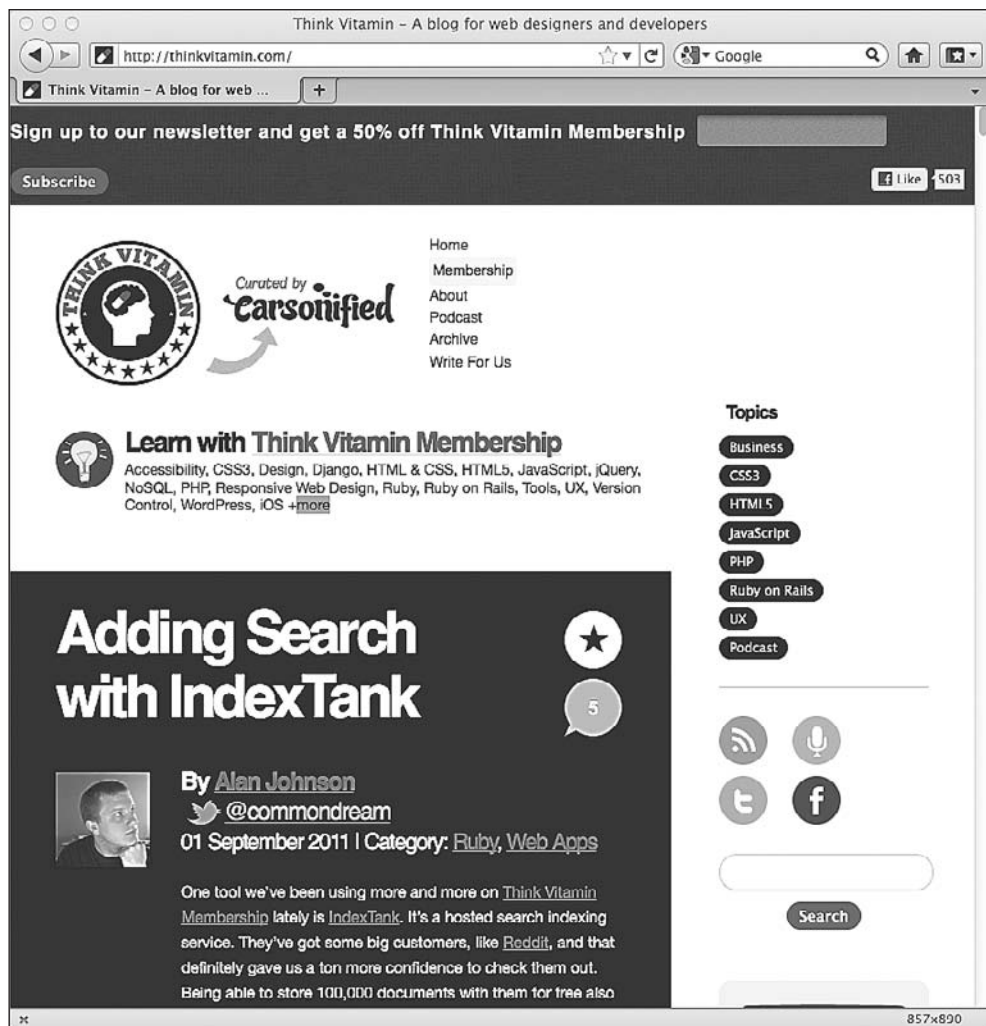


Рис. 1.5. Страница открыта в области просмотра шириной меньше 880 пикселей

Теперь сделайте ширину своей области просмотра меньше чем 600 пикселей, и вы увидите существенные перемены (рис. 1.6).

Как вам результат? Блок навигации целиком подстроился под новую область просмотра, позволив наиболее важной части сайта — содержимому — занять всю ширину окна браузера. Кроме того, обратите внимание, что ссылки в верхнем колонтитуле теперь располагаются горизонтально под логотипом в противоположность тому, как они располагались ранее — сбоку от него, а размеры самого логотипа изменились. Все эти изменения способствуют обеспечению оптимального интерфейса для пользователей в зависимости от размеров области просмотра.

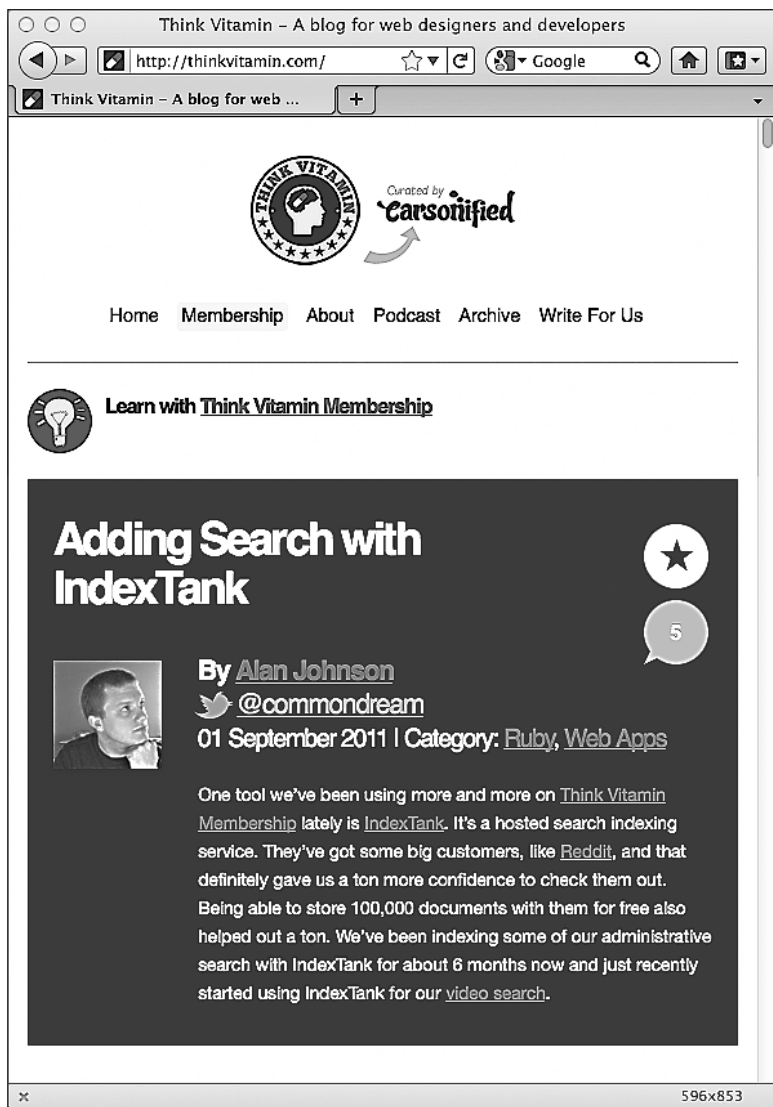


Рис. 1.6. Область просмотра шириной менее 600 пикселей

Взглянем на другой пример по адресу <http://2011.dconstruct.org/>. В широкой области просмотра (например, более 1350 пикселей) этот сайт выглядит, как показано на рис. 1.7.

Обратите особое внимание на сетку с девятью изображениями. Видите, что происходит по мере уменьшения ширины области просмотра (когда она становится меньше 960 пикселей)? Сетка из трех рядов, в каждом из которых содержится по три изображения, превращается в сетку из трех рядов с двумя изображениями в каждом и одного ряда с тремя изображениями, располагающегося в самом низу (рис. 1.8).



Рис. 1.7. Сайт dConstruct 2011



Рис. 1.8. При уменьшении размеров области просмотра изображения перестраиваются

Дальнейшее уменьшение ширины области просмотра до менее чем 720 пикселей приведет к тому, что мы столкнемся со следующей «контрольной точкой» в дизайне: ссылки в верхнем колонтитуле теперь включают изображения, которые обеспечивают более удобную целевую область для навигации с использованием сенсорного экрана (рис. 1.9).



Рис. 1.9. Ссылки сопровождаются изображениями

Если мы еще уменьшим ширину области просмотра, сделав ее менее 480 пикселей, то сетка с изображениями снова изменится и теперь в ней будет ряд с двумя изображениями, затем — с тремя, а потом — с четырьмя. Размеры этих изображений продолжают меняться по мере уменьшения ширины области просмотра где-то до 300 пикселей. В качестве наглядного примера взгляните на рис. 1.10, где показано, как все это будет выглядеть на экране iPhone.

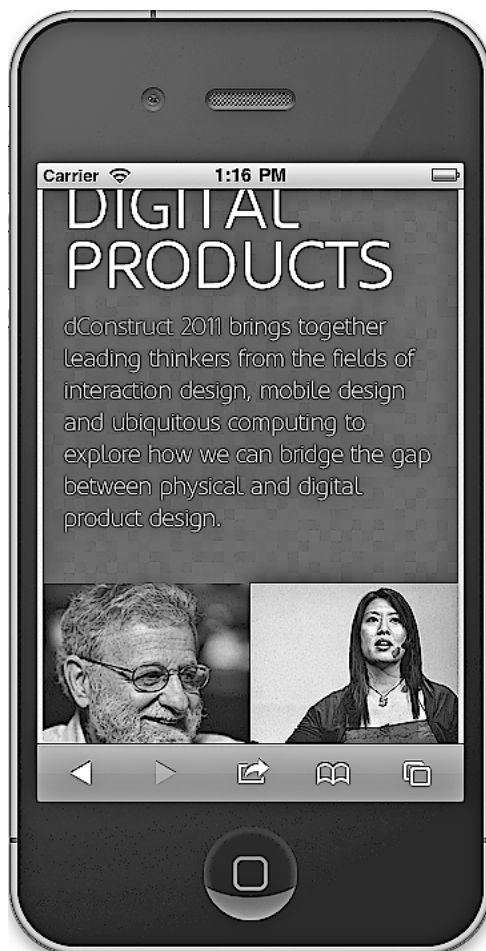


Рис. 1.10. Сайт dConstruct 2011 на экране смартфона

Онлайн-источники вдохновения

Веб-ресурс, который поможет вам обрести вдохновение, располагается по адресу <http://mediaqueri.es>. Однако не все сайты, представленные там, обязательно задействуют всю адаптивную методологию отображения содержимого сначала в небольших областях просмотра, а затем прогрессивно улучшая дизайн при отображении на более широких экранах. Тем не менее на этом этапе, пока мы рассматриваем возможности адаптивного веб-дизайна, можно обратиться к множеству замечательных примеров, чтобы почерпнуть из них идеи.

Несмотря на то что обзор этих сайтов с изменением размеров области просмотра наглядно показывает, на что способен адаптивный веб-дизайн, он не позволяет оценить достоинства HTML5. Они остаются за кадром, поэтому обратим на них внимание и посмотрим, чем так замечательна версия HTML5.

1.6. Преимущества HTML5

Язык HTML5 делает акцент на упрощении разметки, необходимой для создания соответствующих W3C-стандартам страниц и объединения всего требуемого CSS- и JavaScript-кода, а также файлов изображений. Если вести речь о пользователях, которые, вероятно, просматривают наши страницы, подключаясь к Интернету по каналу с невысокой скоростью передачи информации, а также об основной цели наших адаптивных веб-дизайнов, то нам необходимо, чтобы сайт не просто адаптировался к их более ограниченным областям просмотра, но и загружался настолько быстро, насколько это возможно. Несмотря на то что при удалении лишних элементов разметки объем передаваемых данных уменьшается совсем незначительно, даже малая экономия будет полезна!

HTML5 несет в себе дополнительные преимущества и функции по сравнению с предыдущей версией (HTML 4.01). Веб-разработчиков клиентских приложений, скорее всего, заинтересуют новые семантические HTML5-элементы, которые обеспечивают более осмысленный код для поисковых механизмов. HTML5 также позволяет организовать для пользователей обратную связь на основе базового взаимодействия с сайтами вроде отправки данных форм и т. п., зачастую устраняя необходимость в JavaScript-обработке форм, более требовательных к вычислительным ресурсам. Опять-таки это хорошая новость, поскольку мы сможем создать более компактную и быстрее загружающуюся кодовую базу.

Экономия времени и кода благодаря HTML5

Первая строка любого HTML-документа начинается с DOCTYPE (Document Type Declaration — **определение типа документа**). Честно говоря, эта часть будет автоматически добавляться используемым редактором кода веб-страниц либо можно скопировать и вставить ее из уже имеющегося шаблона (в действительности никто не вводит вручную определение типа документа HTML 4.01 целиком). До появления HTML5 DOCTYPE для стандартной страницы на HTML 4.01 выглядело следующим образом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```

Однако теперь, когда у нас есть HTML5, определение типа документа выглядит так:

```
<!DOCTYPE html>
```

Сейчас я не печатаю вручную определение типа документа каждый раз, когда пишу страницу, и, думаю, вы поступаете точно так же. «Что же такого особенного в HTML5?» — спросите вы. Как насчет добавления ссылок в JavaScript или CSS на ваших страницах? При использовании HTML 4.01 правильный вариант указания ссылки на файл сценария будет таким:

```
<script src="js/jquery-1.6.2.js" type="text/javascript"></script>
```

HTML5 упрощает его:

```
<script src="js/jquery-1.6.2.js"></script>
```

Как вы можете видеть, больше нет необходимости указывать атрибут `type`. Похожим образом дело обстоит с добавлением ссылок на CSS-файлы. HTML5 также допускает использование намного менее аккуратного синтаксиса, который будет считаться валидным. Например, `<script Src=js/jquery-1.6.2.js></script>` будет точно таким же валидным, как и предыдущий образец. Здесь нет кавычек вокруг источника сценария, а в именах тега и атрибута используется сочетание символов в верхнем и нижнем регистре. Однако для HTML5 это не имеет значения: данный код сможет пройти валидацию в W3C-валидаторе HTML5 (<http://validator.w3.org/>). Это хорошая новость для вас, если ваш код написан не очень аккуратно и, что важнее, если вы хотите убрать из своей разметки все лишние символы. Есть и другие особенности написания кода, которые облегчают жизнь. Предлагаю взглянуть на новые семантические HTML5-элементы.

Новые семантически значимые элементы тегов HTML5

При структурировании HTML-страницы стандартный способ разметки верхнего колонтитула и блока навигации выглядит примерно так:

```
<div class="header">
  <div class="navigation">
    <ul class="nav-list">
      <li><a href="#" title="Home">Home</a></li>
      <li><a href="#" title="About">About</a></li>
    </ul>
  </div> <!-- конец navigation -->
</div> <!-- конец header -->
```

Однако взгляните, как все это можно сделать с использованием HTML5:

```
<header>
  <nav>
    <ul id="nav-list">
      <li><a href="#" title="Home">Home</a></li>
      <li><a href="#" title="About">About</a></li>
    </ul>
  </nav>
</header>
```

Как вам результат? Вместо безликих тегов `<div>` для каждого структурного элемента (пусть даже и с добавленными именами классов) HTML5 дает нам семантически более значимые элементы. Общие структурные разделы страниц, например верхний колонтитул и навигационное меню (и многие другие, в чем вы вскоре убедитесь), располагают своими собственными тегами элементов. Наш код стал намного более понятным благодаря тегу `<nav>`, который «говорит» браузерам: «Эй, вот этот раздел является навигационным». Это хорошая новость для нас,

но, возможно, более важно то, что это хорошая новость и для поисковых механизмов. Теперь они смогут лучше понимать наши страницы, чем когда-либо прежде, и соответствующим образом ранжировать их содержимое.

При создании HTML-страниц я часто использую именно этот подход, зная, что они затем будут переданы команде разработчиков серверных приложений (это такие крутые ребята, имеющие дело с PHP, Ruby, .NET, ColdFusion и т. д.), прежде чем наконец будут размещены в Интернете. Чтобы оставаться в хороших отношениях с разработчиками серверных приложений, я часто снабжаю комментариями закрывающие теги `</div>`, благодаря чему другие люди (и я сам) могут с легкостью разобраться, где заканчиваются разделы `<div>`. HTML5 в значительной степени сводит на нет эту задачу. При взгляде на закрывающий тег, например `</header>`, сразу понятно, какой элемент он закрывает, и при этом не требуются никакие комментарии.

Здесь я лишь вкратце перечислил, чем для вас полезны семантические HTML5-элементы. Прежде чем двигаться дальше, вам предстоит познакомиться еще кое с чем, без чего вся эта новая эра веб-дизайна не наступила бы. Это CSS3.

1.7. CSS3 делает возможным адаптивный веб-дизайн и многое другое

Если вы занимаетесь веб-дизайном с середины 1990-х годов, значит, помните, что в то время все дизайны были основаны на таблицах, а стилизация переплеталась с содержимым. **Каскадные таблицы стилей** (Cascading Style Sheets — CSS) были созданы в качестве способа разделения дизайна и содержимого. Разработчики потратили некоторое время на то, чтобы вступить в новый, необычный мир CSS-дизайна, а сайты вроде <http://www.csszengarden.com> подготовили почву, наглядно показав, чего можно добиться благодаря каскадным таблицам стилей. С тех пор язык CSS стал стандартным инструментом для определения слоя представления веб-страниц, а на текущий момент утвержденной версией спецификации CSS является CSS 2.1. Версии CSS3 еще только предстоит пройти полное утверждение, однако это не значит, что многое из того, что в нее входит, не применимо уже сегодня. Рабочая группа W3C отмечает по адресу <http://www.w3.org/TR/CSS/#css3> следующее:

«Третья версия CSS базируется на второй, модуль за модулем, используя в качестве своего ядра спецификацию CSS 2.1. Каждый модуль привносит функциональность и/или заменяет часть спецификации CSS 2.1. Согласно задумке рабочей группы новые CSS-модули не будут противоречить спецификации CSS 2.1: они лишь будут привносить новые возможности и совершенствовать определения».

Многое в черновой спецификации W3C написано с использованием невразумительной юридической терминологии. Если же выражаться простым языком, то для нас важно, что версия CSS3 представляет собой набор «привинчиваемых» модулей, а не что-то консолидированное и неделимое. Поскольку ядро новой версии составляет CSS 2.1, к ней применимы все методики, используемые для CSS 2.1. Отдельные, более «зрелые» CSS3-модули (поскольку не все из них находятся в одинаковом состоянии готовности) можно активно применять уже сейчас, не дожидаясь, пока вся спецификация будет утверждена.

Важный момент: применение CSS3 не приведет к каким-либо нарушениям!

Пожалуй, самое главное заключается в том, что добавление в код свойств, которые непонятны устаревшим версиям браузеров, не приведет к каким-либо проблемам. Подобные версии браузеров (включая Internet Explorer 6, 7 и 8) благополучно пропустят CSS3-свойства, которые не смогут обработать. Это дает возможность разработчикам прогрессивно совершенствовать области дизайнов для браузеров, лучше оснащенных в функциональном плане, одновременно предусматривая надлежащий резервный вариант для устаревших версий браузеров.

Как CSS3 позволяет решать повседневные задачи, связанные с дизайном?

Взглянем на популярную задачу, с которой разработчики сталкиваются в большинстве проектов. Нужно создать скругленный угол для экранного элемента, скажем, для интерфейса с вкладками, или, например, для прямоугольного элемента вроде верхнего колонтитула. При использовании CSS 2.1 это можно сделать благодаря методике **sliding doors** («раздвижные двери») (<http://www.alistapart.com/articles/slidingdoors/>), в результате чего одно фоновое изображение будет располагаться позади другого. HTML-код прост, как показано далее:

```
<a href="#"><span>Box Title</span></a>
```

Мы добавим фон со скругленными углами для элемента `<a>`, создав два изображения. Первое — `headerLeft.png` — будет 15 пикселей в ширину и 40 пикселей в высоту, а второе — `headerRight.png` — в этом примере всегда будет шире верхнего колонтитула (в данном случае 280 пикселей). Каждое из этих изображений будет половиной «раздвижных дверей». По мере увеличения одного элемента (текста в наших тегах ``) фон будет заполнять соответствующее пространство, то есть в результате мы получим решение, которое обеспечит скругление углов и не потеряет актуальности со временем. Вот как будет выглядеть CSS-код в этом примере:

```
a {
  display: block;
  height: 40px;
  float: left;
  font-size: 1.2em;
  padding-right: 0.8em;
  background: url(images/headerRight.png) no-repeat scroll top right;
}
a span {
  background: url(images/headerLeft.png) no-repeat;
  display: block;
  line-height: 40px;
  padding-left: 0.8em;
}
```

На рис. 1.11 видно, как итоговый результат будет выглядеть в Google Chrome (v16):

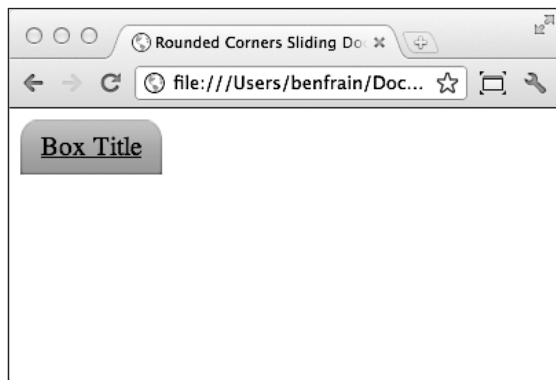


Рис. 1.11. Результат применения методики «раздвижных дверей»

Такой подход позволяет решить описанную ранее задачу, связанную с дизайном, однако требует дополнительной разметки (семантически у элемента `` нет значения) и отправки двух дополнительных HTTP-запросов (для изображений) на сервер для создания итогового эффекта на экране. Теперь мы можем объединить два изображения в одно, чтобы создать спрайт, а затем использовать CSS-свойство `background-position` для его сдвига. Однако, несмотря на то что такое решение снижает требования к скорости канала подключения пользователя, оно будет негибким. А что, если заказчик захочет, чтобы радиус скругления углов был меньше? Или пожелает применить другой цвет? Тогда нам придется заново переделывать наше изображение (-я). Прискорбно, но до появления CSS3 дизайнеры и разработчики клиентских приложений сталкивались именно с таким положением дел. Дамы и господа, я видел будущее, и там главенствует CSS3! Пересмотрим наш HTML-код, сделав его таким:

```
<a href="#">Box Title</a>
```

CSS-коду теперь можно придать следующий вид:

```
a {
  float: left;
  height: 40px;
  line-height: 40px;
  padding-left: 0.8em;
  padding-right: 0.8em;
  border-top-left-radius: 8px;
  border-top-right-radius: 8px;
  background-image: url(images/headerTiny.png);
  background-repeat: repeat-x;
}
```

На рис. 1.12 показано, как выглядит CSS3-версия нашей кнопки в том же браузере, что мы использовали чуть ранее (Chrome версии 16).

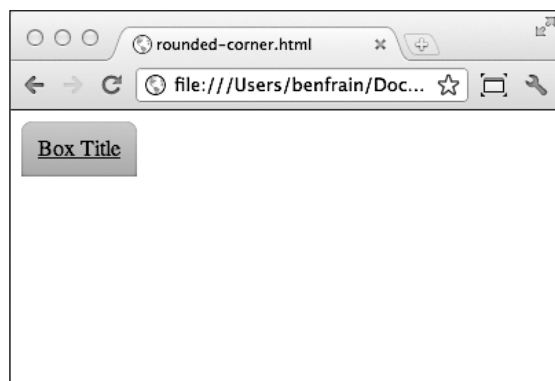


Рис. 1.12. Результат применения CSS3-кода

В этом примере два предыдущих изображения были заменены одним изображением шириной 1 пиксел, которое повторяется по оси X. Несмотря на то что изображение имеет ширину лишь 1 пиксел, оно составляет 40 пикселей в высоту и, надо надеяться, окажется больше по высоте, чем любое вложенное содержимое. При использовании того или иного изображения в качестве фона всегда нужно выбирать такое, которое будет иметь запас по высоте с расчетом на возможное переполнение содержимым, но это, к сожалению, приводит к тому, что приходится использовать крупные изображения. Кроме того, повышаются требования к скорости канала подключения пользователя.

Однако в данном случае, в отличие от решения, полностью основанного на изображениях, CSS3 за нас заботится о скруглении углов благодаря `border-radius` и прочим соответствующим свойствам. Заказчик хочет, чтобы углы имели небольшой радиус скругления, например 12 пикселей? Нет проблем: просто измените значение свойства `border-radius` на `12px`, и требуемый результат будет достигнут. Это CSS-свойство, обеспечивающее скругленные углы, представляет собой эффективный и гибкий инструмент и поддерживается браузерами Safari (версии 3 и выше), Firefox (версии 1 и выше), Opera (версии 10.5 и выше), Chrome (версии 3 и выше) и Internet Explorer 9. Компания Microsoft с таким энтузиазмом относится к поддержке этого свойства в Internet Explorer 9 (надеюсь, вы уловили здесь мой легкий сарказм), что даже создала интерактивную страницу, демонстрирующую различные эффекты, которых можно добиться с его помощью. Взглянуть на них вы сможете по следующему URL-адресу: <http://ie.microsoft.com/testdrive/html5/borderradius/default.html>.

CSS3 идет еще дальше и избавляет нас от необходимости в изображениях, обеспечивающих градиентный фон, генерируя такой эффект непосредственно в браузере. Все это пока не очень хорошо поддерживается браузерами, однако благодаря, например, свойству `linear-gradient(yellow, blue)` любой элемент можно обеспечить фоном в виде градиента, генерируемого CSS3.

Градиенты можно задавать, указывая основные цвета, традиционные шестнадцатеричные значения (например, `#BFBFBF`) или один из цветовых режимов CSS3 (подробнее на эту тему мы поговорим в главе 5). Если вы хотите, чтобы пользователи

устаревших браузеров вместо градиента видели однотонный фон (а не вообще ничего), то **CSS-код вроде того, что приведен чуть ниже, будет обеспечивать однородный цвет, если браузер не сможет обработать градиент:**

```
background-color: #42c264;
background-image: -webkit-linear-gradient(#4fec50, #42c264);
background-image: -moz-linear-gradient(#4fec50, #42c264);
background-image: -o-linear-gradient(#4fec50, #42c264);
background-image: -ms-linear-gradient(#4fec50, #42c264);
background-image: -chrome-linear-gradient(#4fec50, #42c264);
background-image: linear-gradient(#4fec50, #42c264);
```

Свойство `linear-gradient` дает указание браузеру начинать с первого значения цвета (которым в данном примере является `#4fec50`), а затем переходить ко второму значению (`#42c264`).

Как видите, в CSS-коде свойство `background-image: linear-gradient` повторяется с использованием разных префиксов, например `-webkit-`. Это позволяет различным производителям браузеров (так, например, `-moz-` относится к Mozilla Firefox, `-ms-` указывает на Microsoft Internet Explorer и т. д.) **экспериментировать с собственными** реализациями новых CSS3-свойств, прежде чем представлять готовый продукт, когда префиксы уже не будут нужны. Поскольку таблицы стилей по своей природе являются каскадными, вариант без префикса расположен в конце кода. Это подразумевает, что он будет превалировать над предшествующими объявлениями при наличии таковых.

1.8. Смотри, мама, нет изображений!

На рис. 1.12 показано, как выглядит законченная CSS3-кнопка в браузере Chrome версии 16, который мы использовали чуть ранее.

Думаю, вы согласитесь с тем, что любые отличия варианта на основе изображений от версии, полностью базирующейся на CSS, **несущественны**. Создание визуальных элементов с применением CSS3 позволяет сделать наш адаптивный веб-дизайн намного более легковесным, чем если бы мы создавали его с использованием изображений. Более того, градиенты изображений хорошо поддерживаются современными мобильными браузерами, при этом единственный недостаток заключается в отсутствии поддержки градиентов со стороны таких браузеров, как Internet Explorer версии 9 и ниже.

Что еще может предложить CSS3? До сих пор мы рассматривали весьма прозаические примеры того, как CSS3 может помочь при решении повседневных задач, связанных с разработкой. Однако теперь посмотрим, какие по-настоящему удивительные вещи позволяет делать CSS3. Запустите браузер Safari или Chrome и откройте страницу по адресу <http://www.panic.com/blog/>. К сожалению, этот веб-дизайн не является адаптивным, но для нас интерес представляют «приколотые булавками» заметки в верхней части страницы (рис. 1.13). Наведите указатель мыши на любую из них и посмотрите, как изображение при этом слегка двигается вправо-влево, а его нижняя часть как бы приподнимается. Здорово, не так ли? В прошлом подобного расширенного эффекта можно было добиться с использованием требо-

вательного к вычислительным ресурсам Flash или JavaScript. Однако в данном случае эффект был обеспечен исключительно благодаря CSS3-трансформациям. Применение CSS3 вместо JavaScript или Flash позволяет сделать анимацию более легковесной, удобной в сопровождении и, следовательно, идеально подходящей для адаптивного веб-дизайна. Пользователи браузеров, поддерживающих версию CSS3, смогут наблюдать такой эффект, в то время как остальные вместо эффекта увидят лишь статические изображения.



Рис. 1.13. Страница с анимированными заметками

Еще один замечательный пример CSS3-трансформаций можно увидеть по адресу http://demo.marcofolio.net/3d_animation_css3/. Опять-таки веб-дизайн данной страницы не является адаптивным, однако нас интересуют CSS-трюки, которые на ней задействованы. Взгляните на страницу сначала в Internet Explorer 9 или Firefox (браузер Firefox версии 9.0 не поддерживает соответствующий CSS3-модуль). А затем посмотрите на нее в Safari версии 5 и выше или Chrome версии 16 и выше. Приведенный на рис. 1.14 скриншот не позволяет в полной мере оценить анимацию, поэтому, если вы сами не взглянете на эту страницу, вам придется поверить мне на слово — демонстрируемые там эффекты хороши.

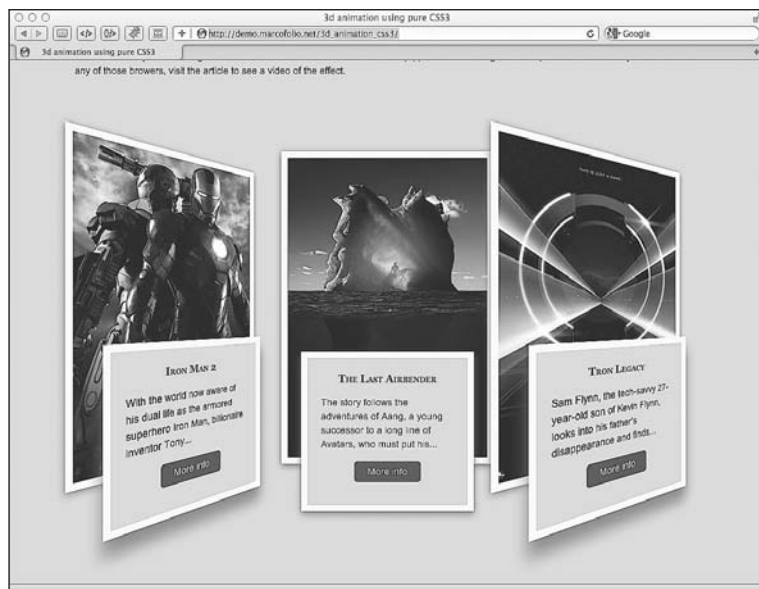


Рис. 1.14. 3D-анимация с использованием CSS3

Такие отлично выглядящие эффекты работают не только в браузерах Safari и Chrome на основе WebKit. Веб-дизайн страницы, располагающейся по адресу http://designlovr.com/examples/dynamic_stack_of_index_cards/, будет работать и в Firefox. Это еще один пример, который базируется исключительно на CSS3 (рис. 1.15).



Рис. 1.15. Создание динамического стека карточек с помощью CSS3

Безусловно, подобные эффекты не являются неотъемлемой частью любого из описанных сайтов. Это только прекрасная демонстрация прогрессивного улучшения (progressive enhancement). Пользователи браузеров, не поддерживающих эти эффекты, увидят лишь статические изображения. Однако если у пользователя установлен современный браузер, он сможет насладиться расширенными визуальными эффектами. Но надо отметить, что поддержка браузерами 3D-трансформаций CSS3 довольно ограничена. Например, широко поддерживаются такие CSS3-правила, что позволяют добавлять тени к тексту, градиенты, границы со скругленными углами, RGBA-цвет и разнообразные фоновые изображения. Они обеспечивают гибкое решение распространенных проблем, связанных с дизайном, которые долгие годы мучили и озадачивали нас.

1.9. Можно ли реализовать потенциал HTML5 и CSS3 уже сегодня?

Любой инструмент или методика должны использоваться только в том случае, если того требует создаваемое приложение. При работе над проектами мы как разработчики обычно ограничены в плане времени и ресурсов, доступных для того, чтобы сделать приложения финансово жизнеспособными.

Браузер Internet Explorer версии 7 и 8 не поддерживает новые семантические HTML5-элементы и CSS3-свойства в качестве стандарта, и, если подавляющее большинство посетителей вашего сайта будет использовать Internet Explorer 7 или 8, нет смысла концентрировать свои усилия на создании для него адаптивного веб-дизайна на основе HTML5 и CSS3. Однако это не значит, что этого нельзя сделать. Как вы увидите в главе 9, существует постоянно увеличивающийся ряд инструментов, позволяющих ставить заплатки на браузеры (по большей части — на устаревшие версии Internet Explorer), не поддерживающие функции, имеющиеся в браузерах, вышедших позднее их. Такие инструменты называются **полизаполнениями** (polyfill), поскольку используются для «заделки» брешей в устаревших браузерах. Однако наилучшая политика — использование осмысленного подхода к реализации адаптивного веб-дизайна.

По своему опыту могу сказать, что обычно я сначала ставлю перед собой следующие вопросы.

- Хочет ли заказчик охватить максимально широкий круг интернет-пользователей, количество которых постоянно растет? Если да, то подходящим выбором будет адаптивная методология.
- Хочет ли заказчик получить максимально чистую, наиболее быстро работающую и самую удобную в сопровождении кодовую базу? Если да, то подходящим выбором будет адаптивная методология.
- Понимает ли заказчик, что взаимодействие может и должно слегка отличаться при использовании разных браузеров? Если да, то подходящим выбором будет адаптивная методология.
- Хочет ли заказчик, чтобы нужный ему дизайн выглядел одинаково во всех браузерах, включая Internet Explorer версии 8 и ниже? Если да, то адаптивный веб-дизайн — не самый подходящий выбор.

- Будут ли 70 или более процентов текущих или ожидаемых посетителей сайта заказчика использовать Internet Explorer версии 8 или ниже? Если да, то адаптивный веб-дизайн будет не самым подходящим выбором.

Кроме того, если позволяет бюджет, «мобильная» версия сайта, полностью выполненная под заказ, может оказаться более подходящим вариантом, нежели созданная с использованием принципов адаптивного веб-дизайна. Для ясности отмечу, что «мобильными» сайтами я называю решения, полностью сфокусированные на мобильных устройствах и обеспечивающие разное содержимое и взаимодействие для мобильных пользователей. Не думаю, что кто-либо из сторонников методик адаптивного веб-дизайна станет спорить с тем, что вариант на основе адаптивного веб-дизайна будет подходящей заменой для «мобильного» сайта в любой ситуации.

1.10. Адаптивный веб-дизайн — это не чудодейственное средство

Хоть я и рискую вступить в противоречие с выражением «не учи ученого», хочу еще раз отметить, что адаптивный веб-дизайн с использованием HTML5 и CSS3 не является чудодейственной панацеей, которая позволяет избавиться от всех сложностей, касающихся дизайна и обеспечения содержимого. Как это всегда было в случае с веб-дизайном, специфика того или иного проекта (то есть размер бюджета, целевая демографическая аудитория и желаемый результат) должна диктовать подход к его реализации. Однако по своему опыту могу сказать, что если бюджет ограничен и/или создание «мобильного» сайта полностью под заказ нецелесообразно, то адаптивный веб-дизайн почти всегда сможет обеспечить более качественное и всестороннее пользовательское взаимодействие, чем стандартный дизайн с фиксированной шириной.

1.11. Как объяснить заказчикам, что сайты не должны выглядеть одинаково во всех браузерах

Последняя проблема, с которой нужно справиться перед тем, как приступить к адаптивному веб-дизайну, лежит скорее в психологической сфере. И в какой-то мере, возможно, является самой трудной для преодоления. Например, меня часто просят преобразовать имеющиеся графические дизайны в соответствующие стандартам веб-страницы на основе HTML/CSS и jQuery. По своему опыту могу сказать, что графическим дизайнерам редко (если я говорю «редко», то имею в виду, что это почти никогда не случается) требуется держать в уме нечто иное, кроме «настоянной версии» сайта с фиксированной шириной, в тот момент, когда они создают свои композиции дизайнов. Таким образом, моя задача заключается в том, чтобы создать совершенную до последнего пиксела визуализацию соответствующего дизайна в каждом из известных браузеров. Неудача или успех в решении этой задачи определяет в глазах моего заказчика профессионализм графического дизайнера,

да и успешность всего проекта в целом. Подобный настрой особенно часто наблюдается у клиентов с опытом работы в дизайне в сфере печатных средств массовой информации. И их мотивацию легко понять: когда заказчик утверждает композицию дизайна, она передается дизайнеру/разработчику клиентских приложений (вам или мне), который затем предпринимает меры для того, чтобы итоговый код настолько соответствовал выбранному дизайну во всех основных браузерах, насколько это представляется возможным. Что заказчик видит, то он и получит.

Однако если вы когда-нибудь пытались сделать так, чтобы тот или иной современный веб-дизайн выглядел в браузере Internet Explorer 6 и 7 точно так же, как он выглядит в актуальных, соответствующих стандартам браузерах, например в Safari, Firefox или Chrome, то вам будут известны сложности, присущие такому процессу. У меня зачастую уходит до 30 % выделенного на проект времени/бюджета на то, чтобы устранить бреши и недостатки, свойственные хилым устаревшим браузерам. Это время можно было бы потратить на внесение улучшений и оптимизацию кода в плане его объема для растущего количества пользователей, просматривающих сайты в современных браузерах, вместо того чтобы ставить заплатки и ковыряться в кодовой базе в попытке обеспечить скругленные углы, прозрачные изображения, корректно выровненные элементы формы и т. д. для сокращающейся аудитории пользователей Internet Explorer.

Жаль, но единственным «лекарством» при таком сценарии является просвещение. Заказчику необходимо объяснить, почему адаптивный веб-дизайн — целесообразный вариант, что он несет в себе и почему итоговый дизайн не будет и не должен одинаково выглядеть во всех областях просмотра и браузерах. Одни заказчики способны все это понять, а другие — нет. К сожалению, некоторые из них все равно будут хотеть, чтобы скругленные углы и отбрасываемые тени корректно выглядели и в Internet Explorer тоже!

Приступая к новому проекту, независимо от того, применим для него адаптивный веб-дизайн или нет, я разъясняю клиенту следующие моменты.

- Давая устаревшим браузерам возможность немного по-другому отображать страницы, мы делаем код более удобным в сопровождении и менее затратным при обновлении в будущем.
- Делая так, чтобы все элементы выглядели одинаково даже в устаревших браузерах (например, в Internet Explorer версии 8 и ниже), мы добавляем на сайт значительное количество изображений. Это замедляет его работу, делает его создание более дорогим, а также усложняет сопровождение.
- Более компактный код, понятный современным браузерам, означает более быстро работающий сайт. Такой сайт будет занимать более высокое место в результатах, выдаваемых поисковыми механизмами, чем тот, что работает медленно.
- Количество пользователей устаревших браузеров сокращается, в то время как пользователей современных браузеров становится все больше и больше, поэтому необходимо обеспечивать поддержку последних!
- Самое важное заключается в том, что, поддерживая современные браузеры, мы обеспечим для своих пользователей возможность наслаждаться адаптивным веб-дизайном, приспособившимся к отличающимся областям просмотра в браузерах на разных устройствах.

1.12. Резюме

Мы определились с тем, что подразумевается под понятием «адаптивный веб-дизайн», и исследовали отличные примеры адаптивных веб-дизайнов из Интернета, где задействуются инструменты и методики, о которых мы собираемся поговорить. Мы также выяснили, что нам необходимо перейти от методики, при которой центральное место занимают настольные устройства, к способам, ориентированным на более широкий круг устройств, сначала приспособивая содержимое к самым маленьким вероятным областям просмотра, а затем прогрессивно улучшая дизайн по мере перехода к более крупным областям просмотра. Взглянув на новую спецификацию HTML5, **мы установили, что можем уже сегодня с выгодой использовать** многое из того, что в ней есть, а именно новые семантические элементы разметки, позволяющие создавать страницы с применением меньшего объема кода, но с большим значением, чем это было возможно ранее.

Ключевое место в создании полностью адаптивного веб-дизайна занимает CSS3. Прежде чем мы станем использовать эту версию для добавления в наш дизайн визуальных украшений вроде градиентов, скругленных углов, теней, отбрасываемых текстом, анимаций и трансформаций, она сыграет для нас более важную роль. Используя CSS3-запросы, мы сможем нацеливать специфические CSS-правила на определенные области просмотра. В следующей главе мы всерьез начнем работать с адаптивным веб-дизайном.

2 Медиазапросы: поддержка разных областей просмотра

Как уже отмечалось в предыдущей главе, CSS3 состоит из набора «привинчиваемых» модулей. **Медиазапросы** как раз представляют собой такой CSS3-модуль. Они позволяют нацеливать специфические CSS-стили в зависимости от возможностей по отображению информации, имеющихся у того или иного устройства. Так, например, используя лишь несколько строк CSS-кода, мы можем изменить способ отображения содержимого, исходя из таких параметров, как ширина области просмотра, соотношение сторон экрана, ориентация (альбомная или книжная) и т. д.

В этой главе мы сделаем следующее:

- выясним, почему медиазапросы необходимы в адаптивных веб-дизайнах;
- посмотрим, как устроены CSS-медиазапросы;
- разберемся, какие характеристики устройств можно проверять;
- напишем первый CSS3-медиазапрос;
- нацелим правила стилей CSS на определенные области просмотра;
- выясним, как заставить медиазапросы работать на устройствах под управлением операционных систем iOS и Android.

2.1. Медиазапросы можно использовать уже сегодня

Медиазапросы уже широко применяются, а также приобрели обширную поддержку со стороны браузеров (Firefox версии 3.6 и выше, Safari версии 4 и выше, Chrome версии 4 и выше, Opera версии 9.5 и выше, Safari версии 3.2 и выше для операционной системы iOS, Opera Mobile версии 10 и выше, Android версии 2.1 и выше, Internet Explorer версии 9 и выше). Кроме того, можно без труда реализовать **исправления** (хоть и основанные на JavaScript) для таких распространенных устаревших браузеров, как Internet Explorer версий 6, 7 и 8. Если вам прямо сейчас нужны исправления для Internet Explorer версий 6, 7 и 8, то загляните в главу 9.

В общем, нет веских причин, по которым нельзя было бы использовать медиазапросы уже сегодня!



ПРИМЕЧАНИЕ

Спецификации W3C проходят процесс утверждения (если у вас будет свободный день, то изучите официальные разъяснения этого процесса по адресу <http://www.w3.org/2005/10/Process-20051014/tr>) от Working Draft (WD) (Рабочий проект) до Candidate Recommendation (CR) (Возможная рекомендация) и затем Proposed Recommendation (PR) (Предлагаемая рекомендация), прежде чем наконец спустя много лет достигнут статуса W3C Recommendation (REC) (Рекомендация W3C). Таким образом, модули с более высоким уровнем «зрелости» по сравнению с тем, что есть у других, как правило, более безопасны для использования. Например, CSS Transforms Module Level 3 (<http://www.w3.org/TR/css3-3d-transforms/>) имеет статус Working Draft с марта 2009 года, а поддержка его браузерами намного скромнее, чем модулей со статусом Candidate Recommendation, например медиазапросов.

2.2. Почему адаптивным веб-дизайном необходимы медиазапросы?

Без такого CSS3-модуля, как медиазапросы, мы не смогли бы нацеливать отдельные CSS-стили на определенные характеристики устройств, например ширину области просмотра. Если вы заглянете в W3C-спецификацию этого CSS3-модуля (<http://www.w3.org/TR/css3-mediaqueries/>), то увидите следующее официальное описание медиазапросов:

«В настоящее время HTML4 и CSS2 поддерживают медиазависимые таблицы стилей, предназначенные для разных медиатипов. Например, в документе могут использоваться шрифты семейства sans-serif, когда он отображается на экране, и шрифты семейства serif, когда он выводится на печать. Screen и print — это два медиатипа, которые были определены. Медиазапросы расширяют функциональность медиатипов, позволяя присваивать более точные метки таблицам стилей.

Медиазапрос состоит из медиатипа, а также выражений в количестве от 0 и более, которые проверяют состояние определенных медиафункций. К числу медиафункций, которые могут быть использованы в медиазапросах, относятся width, height и color. Применяя медиазапросы, можно приспособить представления к определенному диапазону устройств, не изменяя содержимое».

Синтаксис медиазапросов

Как выглядит CSS-код медиазапросов и, что более важно, как они работают?

Наберите приведенный далее код в конце любого CSS-файла, а затем откройте связанную с ним веб-страницу в браузере:

```
body {  
    background-color: grey;  
}
```

```
@media screen and (max-width: 960px) {  
  body {  
    background-color: red;  
  }  
}  
@media screen and (max-width: 768px) {  
  body {  
    background-color: orange;  
  }  
}  
@media screen and (max-width: 550px) {  
  body {  
    background-color: yellow;  
  }  
}  
@media screen and (max-width: 320px) {  
  body {  
    background-color: green;  
  }  
}
```

Теперь откройте свою веб-страницу в современном браузере (для Internet Explorer как минимум в версии 9) и измените размеры окна браузера. Фоновый цвет страницы будет изменяться в зависимости от текущих размеров области просмотра. Для ясности в приведенном чуть выше коде я использовал именованные цвета, однако обычно для указания цветов применяется шестнадцатеричный код, например #ffffff.

Проанализируем медиазапросы, чтобы понять, как их наиболее эффективно использовать.

Если вам доводилось работать с таблицами стилей CSS2, то вы уже знаете, что можно задавать тип устройства (например, screen или print), соответствующий той или иной таблице стилей, в атрибуте media тега <link>. Для этого необходимо указать ссылку, как это сделано в приведенном далее фрагменте, в тегах <head> в HTML-коде:

```
<link rel="stylesheet" type="text/css" media="screen" href="screenstyles.css">
```

Медиазапросы в основном обеспечивают возможность нацеливать стили, исходя из *возможностей* или *характеристик* устройства, а не только из его *типа*. Считайте все это вопросом, задаваемым браузеру. Если браузер выдает в ответ true, то применяются соответствующие стили. Если же он ответит false, то стили не используются. Вместо того чтобы спрашивать у браузера: «Это устройство с экраном?» — лучше воспользоваться медиазапросом. Он позволяет задать такой вопрос: «Это устройство с экраном в книжной ориентации?» Взглянем на **при-**мер:

```
<link rel="stylesheet" media="screen and (orientation: portrait)"  
href="portrait-screen.css" />
```

Сначала выражение медиазапроса спрашивает о типе («это устройство с экраном?»), а затем — о характеристике («в книжной ориентации?»). Таблица стилей `portrait-screen.css` станет загружаться, если речь будет идти о любом устройстве с экраном, имеющим книжную ориентацию, и игнорироваться для всех остальных. Логiku любого медиазапроса можно реверсировать, добавив `not` в его начало. Например, следующий код обеспечит противоположный результат, нежели наш предыдущий образец, загружая соответствующий файл для любого устройства, не имеющего экрана в книжной ориентации:

```
<link rel="stylesheet" media="not screen and (orientation: portrait)" href="portrait-screen.css">
```

Можно также связывать друг с другом разные выражения. К примеру, расширим наш первый вариант медиазапроса и введем ограничение, согласно которому соответствующий файл будет загружаться для устройств, располагающих областью просмотра шире 800 пикселей.

```
<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px)" href="800wide-portrait-screen.css" />
```

Более того, мы можем указать целый список медиазапросов. Если в результате выполнения какого-либо из них мы получим ответ `true`, то соответствующий файл будет загружен. В противном случае файл загружаться не будет. Вот пример:

```
<link rel="stylesheet" media="screen and (orientation: portrait) and (min-width: 800px), projection" href="800wide-portrait-screen.css" />
```

Здесь необходимо отметить два момента. Во-первых, медиазапросы отделены друг от друга запятыми. Во-вторых, в конце после `projection` нет `and` или комбинации «параметр/значение» в круглых скобках. Это потому, что при отсутствии таких значений медиазапрос будет охватывать все медиатипы. В нашем примере стили будут применяться для всех устройств, являющихся проекторами.

Как и существующие CSS-правила, медиазапросы позволяют условно загружать стили различными путями. До сих пор мы включали их как ссылки на CSS-файлы, которые размещались бы в области `<head></head>` нашего HTML-кода. Однако мы можем задействовать медиазапросы и в самих таблицах стилей. Например, если мы добавим следующий код в таблицу стилей, то он сделает зелеными все элементы `h1` при условии, что у определенного устройства окажется экран шириной 400 пикселей или менее:

```
@media screen and (max-device-width: 400px) {  
  h1 { color: green }  
}
```

Мы также можем использовать CSS-правило `@import` для условной загрузки таблиц стилей в нашу имеющуюся таблицу стилей. Например, приведенный далее код импортирует таблицу стилей с именем `phone.css` при условии, что определенное устройство будет иметь экран, а также располагать областью просмотра с максимальной шириной 360 пикселей:

```
@import url("phone.css") screen and (max-width:360px);
```

Помните, что применение CSS-правила `@import` приводит к увеличению количества HTTP-запросов (что, в свою очередь, отрицательно сказывается на скорости загрузки), поэтому используйте этот метод умеренно.

Что позволяют проверить медиазапросы?

При создании адаптивных веб-дизайнов наиболее часто используемые медиазапросы касаются ширины областей просмотра устройств (`width`) и ширины экранов устройств (`device-width`). По своему опыту могу сказать, что с необходимостью проверять другие характеристики я сталкивался редко. Однако на тот случай, если в этом все же возникнет необходимость, приведу список всех характеристик, которые позволяют проверять медиазапросы. Надеюсь, некоторые из них заинтересуют вас.

- `width` — ширина области просмотра.
- `height` — высота области просмотра.
- `device-width` — ширина поверхности, на которой происходит визуализация (для нас это ширина экрана устройства).
- `device-height` — высота поверхности, на которой происходит визуализация (для нас это высота экрана устройства).
- `orientation` — ориентация экрана устройства (книжная или альбомная).
- `aspect-ratio` — соотношение ширины и высоты в зависимости от ширины и высоты области просмотра. Широкоэкранный дисплей с соотношением сторон 16:9 может быть помечен как `aspect-ratio: 16/9`.
- `device-aspect-ratio` — аналогична `aspect-ratio`, однако базируется на ширине и высоте поверхности устройства, на которой происходит визуализация, а не на ширине и высоте области просмотра.
- `color` — количество бит на каждый из цветовых компонентов. Например, `min-color: 16` будет означать, что экран конкретного устройства имеет 16-битную глубину цвета.
- `color-index` — количество записей в таблице подстановки цветов. Значения должны быть числовыми и не могут быть отрицательными.
- `monochrome` — характеристика позволяет узнать, сколько бит приходится на каждый пиксел в буфере монохромных кадров. Значение должно быть числом (целым), например `monochrome: 2`, и не может быть отрицательным.
- `resolution` — может использоваться для того, чтобы узнать разрешение экрана или печати, например `min-resolution: 300dpi`. Значения также могут указываться в точках на сантиметр, например `min-resolution: 118dpcm`.
- `scan` — позволяет узнать тип развертки: прогрессивная или чересстрочная, и все это в основном относится к телевизорам. Например, нацелившись на устройства, представляющие собой HD-телевизоры с разрешением 720p (буква *p* означает *progressive*, то есть «прогрессивная развертка»), можно, указав `scan: progressive`, а на устройства, которые являются HD-телевизорами с разрешением 1080i (буква *i* здесь означает *interlaced*, то есть «чересстрочная развертка»), — указав `scan: interlace`.

- `grid` — позволяет узнать, относится ли определенное устройство к устройствам с фиксированным размером символов. Размеры букв на таком устройстве занимают одинаковую ширину и высоту и выстраиваются по заданной сетке. К подобным устройствам можно отнести терминалы, а также телефоны, которые поддерживают только один шрифт.

Все вышеуказанные характеристики за исключением `scan` и `grid` могут снабжаться префиксом `min` или `max` для задания диапазонов. Взгляните, например, на следующий фрагмент кода:

```
@import url("phone.css") screen and (min-width:200px) and (max-width:360px);
```

Здесь минимум (`min`) и максимум (`max`) были установлены в медиазапросе `width` для задания диапазона. Файл `phone.css` будет импортироваться, только когда речь будет идти об устройствах с экраном с минимальной шириной области просмотра 200 пикселей и максимальной шириной области просмотра 360 пикселей.

Использование медиазапросов для изменения дизайна

CSS-стили, располагающиеся ниже в каскадной таблице стилей, имеют больший приоритет по сравнению с эквивалентными стилями, находящимися выше (если только более высоко расположенные стили не окажутся более подходящими). Следовательно, в начале таблицы стилей мы можем задать базовые стили, применимые ко всем версиям нашего дизайна, а затем переопределить соответствующие разделы с помощью медиазапросов далее в документе. Например, можно задать навигационные ссылки как простые текстовые ссылки для версии дизайна, предназначенной для больших областей просмотра (где более высока вероятность того, что пользователи задействуют мышь), а затем медиазапросами заменить соответствующие стили, чтобы обеспечить более крупную целевую область (для нажатий пальцами) для более ограниченных по размеру областей просмотра.

Наилучший способ загрузки медиазапросов для адаптивных веб-дизайнов

Несмотря на то что современные браузеры достаточно умны для того, чтобы игнорировать не предназначенные для них целевые файлы медиазапросов, иногда это не мешает им загружать подобные файлы. Таким образом, размещение разных стилей, связанных с медиазапросами, в отдельных файлах имеет лишь небольшие преимущества (не считая личных предпочтений и/или разделения кода на блоки). Использование отдельных файлов влечет рост количества HTTP-запросов, необходимых для обработки страницы, что, в свою очередь, приводит к ее более медленной загрузке.

JavaScript-инструмент `Respond.js` (<https://github.com/scottjehl/Respond>), позволяющий наиболее быстро добавить частичную поддержку медиазапросов в Internet Explorer версии 8 и ниже, на текущий момент не способен проанализировать CSS-код, на который ссылается команда `@import`. Поэтому я рекомендую добавлять

стили, связанные с медиазапросами, в уже имеющуюся таблицу стилей. Например, вы можете просто добавить медиазапрос в существующую таблицу стилей, используя следующий синтаксис:

```
@media screen and (max-width: 768px) { ваши стили }
```

2.3. Наш первый адаптивный веб-дизайн

Не знаю, как вам, а мне не терпится приступить к адаптивному веб-дизайну! Мы разобрались в принципах медиазапросов, а теперь протестируем их и посмотрим, как они работают на практике. У меня как раз есть проект, на котором мы можем это осуществить. Но позвольте мне сделать небольшое отступление...

Я люблю кино. При этом я часто не согласен с другими (возможно, одна из причин этого кроется в том, что я целыми днями пишу код... в одиночестве!), в частности, насчет того, какой фильм можно назвать хорошим, а какой — нет. Когда оглашаются номинанты на премию «Оскар», у меня нередко возникает чувство глубокого возмущения. Я ничего не могу поделать с ощущением, что награды заслуживают другие фильмы. Мне хотелось бы запустить сайт под названием *And the winner isn't...* («И победителем не становится...»), который будет доступен для просмотра в Интернете по адресу <http://www.andthewinnerisnt.com/>. Там будут честоваться фильмы, которые должны были стать победителями, и критиковаться те, что стали ими (хотя не должны были). Кроме того, на сайте будут располагаться видеоролики, цитаты, изображения, а также будут проводиться опросы, и все это будет иллюстрировать мою правоту (я знаю, что в этом нет нужды, но так будет справедливо).

Не пугайтесь, но наш дизайн имеет фиксированную ширину

Подобно графическим дизайнерам, которых я ранее порицал за то, что они не принимают во внимание разные области просмотра, я начал создавать графический макет на основе фиксированной сетки шириной 960 пикселей. В действительности, несмотря на то что теоретически всегда лучше начинать создание дизайна, задумываясь о взаимодействии, которое будет обеспечиваться для пользователей мобильных или имеющих небольшой экран устройств, и уже оттуда строить все дальше, пройдет еще несколько лет, пока все осознают преимущества такого мышления. А до тех пор вам, скорее всего, придется брать существующие настольные дизайны и модифицировать их, чтобы сделать адаптивными. Поскольку такое положение вещей, вероятно, сохранится в обозримом будущем, мы начнем наш процесс с дизайна с фиксированной шириной. На рис. 2.1 показано, как выглядит его незавершенный макет.

Если проанализировать макет, то можно понять, что он обладает весьма простой и распространенной структурой: верхний колонтитул, навигация, врезка, содержимое и нижний колонтитул. Вероятно, именно такого рода структуру вас постоянно просят создавать заказчики.



Рис. 2.1. Макет дизайна с фиксированной шириной

В главе 4 я расскажу, почему вам следует писать разметку на HTML5. Однако пока мы не будем касаться этого вопроса, поскольку пора реализовать на практике наши новые познания о медиазапросах. Итак, сначала попробуем свои силы в применении медиазапросов, используя старую добрую **HTML4-разметку**. Без содержимого разметка структуры на HTML4 будет выглядеть следующим образом:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>And the winner isn't</title>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>

<body>

<div id="wrapper">
  <!-- верхний колонтитул и навигация -->
  <div id="header">
    <div id="navigation">
      <ul>
        <li><a href="#">Навигация1</a></li>
```



```

        <li><a href="#">Навигация2</a></li>
    </ul>
</div>
</div>
<!-- врезка -->
<div id="sidebar">
    <p>Это врезка</p>
</div>
<!-- содержимое -->
<div id="content">
    <p>Это содержимое</p>
</div>
<!-- нижний колонтитул -->
<div id="footer">
    <p>Это нижний колонтитул</p>
</div>
</div>
</body>
</html>

```

Если открыть проектный файл в Photoshop, то можно увидеть, что верхний и нижний колонтитулы имеют ширину 940 пикселей (с 10-пиксельными полями по бокам), а врезка и содержимое занимают по ширине соответственно 220 и 700 пикселей и имеют 10-пиксельные поля по бокам (рис. 2.2).

Сначала позаботимся о наших структурных блоках (header, navigation, sidebar, content и footer), используя для этого CSS. После вставки стилей сброса наш суперклассный (шутка!) CSS-код для страницы будет выглядеть следующим образом:

```

#wrapper {
    margin-right: auto;
    margin-left: auto;
    width: 960px;
}

#header {
    margin-right: 10px;
    margin-left: 10px;
    width: 940px;
    background-color: #779307;
}

#navigation ul li {
    display: inline-block;
}

#sidebar {
    margin-right: 10px;
    margin-left: 10px;
    float: left;
    background-color: #fe9c00;
}

```

```

width: 220px;
}

#content {
margin-right: 10px;
float: right;
margin-left: 10px;
width: 700px;
background-color: #dedede;
}

#footer {
margin-right: 10px;
margin-left: 10px;
clear: both;
background-color: #663300;
width: 940px;
}

```



Рис. 2.2. Проектный файл, открытый в Photoshop

Чтобы наглядно показать, как работает структура, помимо дополнительного содержимого (*без изображений*), я добавил фоновый цвет для каждого структурного раздела.



ПРИМЕЧАНИЕ

Стили сброса — это набор общих CSS-объявлений, которые сбрасывают разнообразные стили, установленные по умолчанию и используемые браузерами для обработки HTML-элементов. Они добавляются в начало основного CSS-файла для сброса стилей каждого браузера, создавая тем самым равные условия, чтобы стили, добавляемые в таблицу стилей позднее, действовали аналогичным образом в отличающихся браузерах. Не существует «идеального» набора стилей сброса, и у разработчиков имеются собственные варианты такого набора. Стили сброса, которые я использую в HTML4-документах, представляют собой комбинацию оригинальных решений Эрика Мейера (Eric Meyer) (<http://meyerweb.com/eric/tools/css/reset/>) и комплекта персональных предпочтений и трюков, о которых я узнал, изучая код, написанный другими невероятно умными людьми, например Дэном Седерхольмом (Dan Cederholm) (<http://simplebits.com>). Если вы пока еще не используете стили сброса, то добавьте стили, разработанные Эриком Мейером. Как мне кажется, существуют оптимальные отправные точки для HTML5-документов, например [normalize.css](http://necolas.github.com/normalize.css/) (<http://necolas.github.com/normalize.css/>), и мы взглянем на них в главе 4.

На рис. 2.3 показано, как базовая структура будет выглядеть в браузере с областью просмотра шире 960 пикселей.

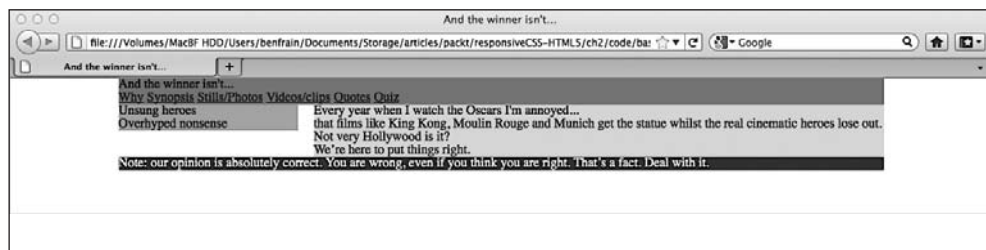


Рис. 2.3. Основа сайта в браузере с областью просмотра шире 960 пикселей

Существует множество других способов обеспечения аналогичной фиксированной структуры содержимого слева/справа с использованием CSS. У вас, несомненно, появятся собственные предпочтения в этом плане. Однако для любого из этих способов неизменным будет следующее: по мере уменьшения ширины области просмотра до менее чем 960 пикселей области содержимого справа начнут обрезать.

Адаптивные веб-дизайны: делаем изображения как можно более экономичными

Чтобы проиллюстрировать проблемы со структурой кода, я добавил эстетическую стилизацию из нашего графического файла в CSS. Поскольку в конечном счете это будет адаптивный веб-дизайн, я позаботился о нарезке фоновых изображений

наиболее экономичным путем. Например, рассмотрим флаги в верхней и нижней части дизайна. Вместо того чтобы создавать одну длинную полосу и сохранять ее как графический файл, я использовал фрагмент, включающий изображение двух флагов. Он будет повторяться по горизонтали в качестве фонового изображения в области просмотра, создавая иллюзию одной длинной полосы (независимо от того, насколько широкой она окажется). В реальном выражении это даст нам разницу в 16 Кбайт (сплошная полоса шириной 960 пикселей была бы PNG-файлом размером 20 Кбайт, в то время как фрагмент с двумя флагами будет занимать лишь 4 Кбайт) для каждой полосы. Мобильные пользователи, которые будут просматривать сайт, подключаясь к Интернету через сотовую связь, оценят такую экономию! На рис. 2.4 показано, как выглядел этот фрагмент (увеличенный на 600 %) до экспорта.

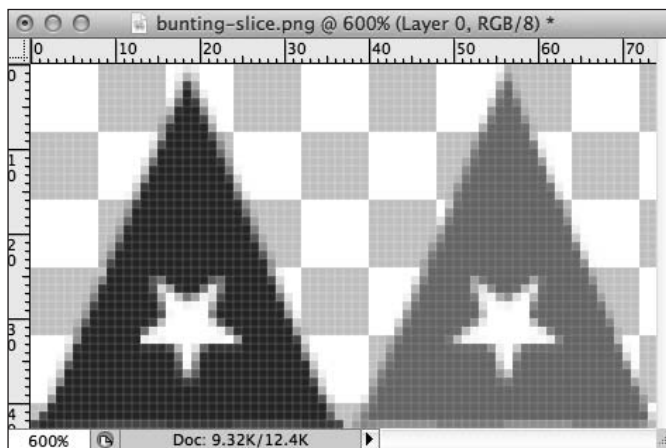


Рис. 2.4. Фрагмент полосы с двумя флагами

После того как фоновые изображения займут свое место и будет задан базовый размер шрифта, сайт And the winner isn't... в окне браузера будет выглядеть так, как показано на рис. 2.5.

Мудро подходите к стилизации, поскольку еще много чего нужно сделать. Например, цвет навигационного меню не меняется с красного на черный, главная кнопка *These should have won* (Они должны были стать победителями) в области содержимого и кнопки *Full info* (Полная информация) во врезке отсутствуют, а все шрифты сильно отличаются от тех, что есть в графическом файле. Однако все это можно исправить с помощью HTML5 и CSS3. Использование для решения этих проблем HTML5 и CSS3 вместо простой вставки файлов изображений (как мы, возможно, поступали ранее) позволит сделать сайт соответствующим нашей адаптивной цели. Помните, что нам необходимо, чтобы «накладные расходы», касающиеся кода и данных, были минимальными, благодаря чему мы сможем обеспечить пользователям с невысоким по скорости каналом подключения возможность комфортно работать с нашим сайтом.



Рис. 2.5. Страница сайта And the winner isn't...

Обрезка содержимого в меньших по размеру областях просмотра

Пока оставим в стороне эстетические проблемы и сосредоточимся на том факте, что, когда ширина области просмотра становится менее 960 пикселей, домашняя страница, над которой мы работаем, очень некрасиво обрезается (рис. 2.6).

Мы уменьшили ширину области просмотра всего лишь до 673 пикселей. Представляете, насколько плохо страница будет выглядеть на чем-нибудь вроде iPhone 3GS? Дисплей этого устройства имеет разрешение лишь 320×480 пикселей. Взгляните на рис. 2.7.



Рис. 2.6. Страница обрезается, если область просмотра становится менее 960 пикселей

Постойте-ка, странно, но все выглядит нормально. Ну, почти что... Конечно, браузер Safari для операционной системы iOS при отображении страниц автоматически использует холст шириной 980 пикселей и затем сжимает его, чтобы подогнать под соответствующую область просмотра. Нам по-прежнему придется

прибегать к увеличению, чтобы просмотреть определенные области, однако содержимое не обрезается. Что нужно сделать для того, чтобы Safari и прочие мобильные браузеры так не поступали?



Рис. 2.7. Страница нашего сайта на экране iPhone 3GS

2.4. Как сделать так, чтобы современные мобильные браузеры не изменяли автоматически размер нашей страницы

Браузеры для операционных систем Android и iOS основаны на WebKit (<http://www.webkit.org/>). Эти браузеры, а также большинство прочих (к которым, например, относится Opera Mobile) позволяют использовать специальный тег `<meta>`

с атрибутом `name="viewport"` для отмены применяемого по умолчанию трюка со сжиманием холста. Тег `<meta>` нужно просто добавить в теги `<head>` в HTML-коде. В нем допускается задать определенное значение для `width` (которое, например, можно указать в пикселах) либо для `initial-scale`, например 2.0 (удвоение фактического размера). Вот вариант тега `<meta>` с атрибутом `name="viewport"`, где заданы значения, согласно которым содержимое в окне браузера будет в два раза больше (200 %) по сравнению с фактическим размером:

```
<meta name="viewport" content="initial-scale=2.0,width=device-width" />
```

Добавим его в наш HTML-код, как это сделано в приведенном далее фрагменте:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="initial-scale=2.0,width=device-width" />
<title>And the winner isn't...</title>
```

Теперь загрузим нашу страницу в браузере Android и посмотрим, как она будет выглядеть (рис. 2.8).



Рис. 2.8. Обновленная страница, открытая в браузере Android

Как видите, это не совсем то, к чему мы стремимся, однако это хороший наглядный пример работы приведенного кода!



ГДЕ МОЖНО НАЙТИ ЭМУЛЯТОРЫ IOS И ANDROID

Несмотря на то что тестирование сайтов на реальных устройствах нельзя ничем заменить, существуют эмуляторы Android и iOS. Android-эмулятор для Windows, Linux и Mac OS X является бесплатным. Для того чтобы обзавестись им, необходимо скачать и установить комплект Android Software Development Kit (SDK), доступный по адресу <http://developer.android.com/sdk/>. Установка осуществляется из командной строки, так что это занятие не для слабонервных. iOS-симулятор доступен только для пользователей Mac OS X и является частью пакета Xcode (его можно бесплатно скачать из Mac App Store). Установив Xcode, вы сможете получить доступ к нему, используя путь ~/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications/iOSSimulator.app.

Проанализируем `ter <meta>` и разберемся, что к чему. Атрибут `name="viewport"` вполне понятен. Следующий атрибут `content="initial-scale=2.0` говорит: «масштабировать содержимое до величины двойного размера» (0.5 означало бы половину размера, а 3.0 — трехкратный размер и т. д.), в то время как выражение `width=device-width` сообщает браузеру, что ширина страницы должна быть равна `device-width`.

`Ter <meta>` также можно использовать для контроля над тем, насколько сильно пользователи смогут увеличивать и уменьшать ширину страницы. Код в приведенном далее примере позволит увеличивать ширину страницы до величины, равной трехкратной ширине экрана устройства, и уменьшать ее до половины ширины экрана устройства:

```
<meta name="viewport" content="width=device-width, maximum-scale=3,
minimum-scale=0.5" />
```

Кроме того, вы можете вообще лишить пользователей возможности осуществлять масштабирование и, несмотря на то что это важный инструмент, обеспечивающий доступность, его применение редко будет целесообразным на практике:

```
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
```

Выражение `user-scalable=no` в данном случае будет уместным.

Мы изменим значение `initial-scale` на 1.0, в результате чего мобильный браузер будет обрабатывать страницу, придавая ей размер, равный 100 % от величины своей области просмотра. Присвоение атрибуту `width` значения `device-width` приведет к тому, что при обработке наша страница получит ширину, равную 100 % ширины окна любого из совместимых мобильных браузеров. Вот `ter <meta>`, который мы будем использовать:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

Если взглянуть на страницу на экране iPad в книжной ориентации, то можно увидеть, что теперь содержимое хоть и обрезается, но не так, как если бы мы смотрели на него сквозь очки с толстыми линзами (рис. 2.9)! Именно это нам и требуется на данном этапе. Мы добились прогресса, поверьте мне!



Рис. 2.9. Вид страницы на экране iPad в книжной ориентации



ПРИМЕЧАНИЕ

Заметив, что тег `<meta>` с атрибутом `name="viewport"` используется все чаще, W3C начал принимать попытки привнести аналогичную функцию в CSS. По адресу <http://dev.w3.org/csswg/css-device-adapt/> вы сможете все узнать о новом объявлении `@viewport`. Идея заключается в том, чтобы вместо тега `<meta>` в разделе `<head>` разметки можно было указать в CSS-коде `@viewport { width: 320px; }`. В результате этого ширина окна браузера была бы задана равной 320 пикселям. Некоторые браузеры уже поддерживают такой синтаксис (например, Opera Mobile), хоть и с использованием префиксов, указывающих на их поставщиков, например `@-o-viewport { width: 320px; }`.

2.5. Подстраиваем дизайн под области просмотра с разной шириной

Разобравшись с нашей проблемой с помощью тега `<meta>` с атрибутом `name="viewport"`, мы сделали так, что теперь браузеры не будут самовольно изменять масштаб страницы, поэтому можем заняться подстройкой дизайна под области просмотра с разной шириной. Мы добавим в CSS-код медиазапрос, касающийся таких устройств, как планшетные компьютеры (например, iPad), у которых ширина области просмотра в книжной ориентации равна 768 пикселям (поскольку в альбомной ориентации она равна 1024 пикселям, при загрузке страницы в такой ориентации она будет отлично смотреться).

```
@media screen and (max-width: 768px) {  
  #wrapper {  
    width: 768px;  
  }  
  #header, #footer, #navigation {  
    width: 748px;  
  }  
}
```

Наш медиазапрос будет изменять ширину элементов `wrapper`, `header`, `footer` и `navigation`, если ширина области просмотра не будет превышать 768 пикселей. На рис. 2.10 показано, как все это будет выглядеть на экране iPad.

Этот пример сильно воодушевил меня. Содержимое теперь подгоняется под экран iPad (или любую область просмотра шириной не более 768 пикселей) без обрезки. Однако нам необходимо внести корректировки, которые будут касаться навигационной области, поскольку ссылки простираются за пределы фонового изображения, а область основного содержимого плавает под врезкой (она слишком широкая, чтобы уместиться в доступном пространстве). Внесем изменения в наш медиазапрос в CSS, как показано в следующем фрагменте кода:

```
@media screen and (max-width: 768px) {  
  #wrapper {  
    width: 768px;  
  }  
  #header, #footer, #navigation {  
    width: 748px;  
  }  
  #content, #sidebar {  
    padding-right: 10px;  
    padding-left: 10px;  
    width: 728px;  
  }  
}
```

Теперь врезка и область содержимого будут заполнять всю страницу целиком и иметь небольшие отступы по краям. Однако это не очень хороший вариант. Я хочу, чтобы первым шло содержимое, а уж затем врезка (по своей природе это второстепенная область). Я допущу здесь еще одну глупую ошибку, если для обеспечения

недостаточно привлекательного вида сайта попытаюсь применить истинную методологию адаптивного веб-дизайна.

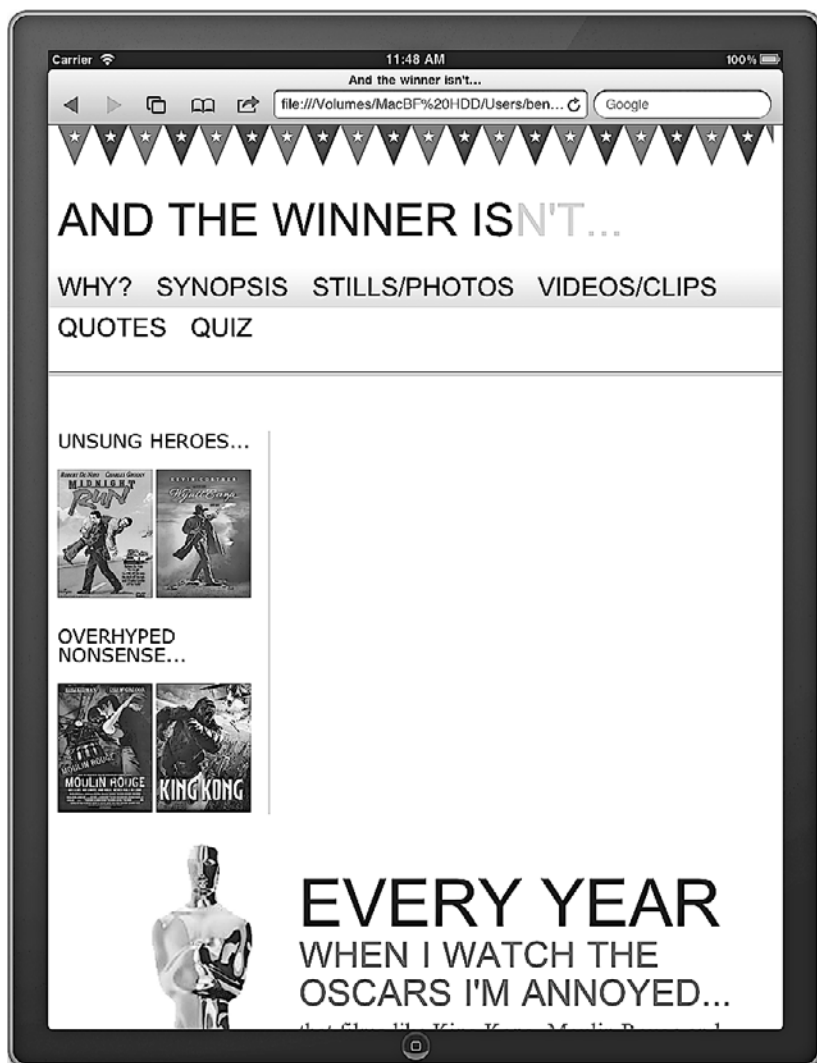


Рис. 2.10. Содержимое сайта подстраивается под область отображения

2.6. В адаптивных веб-дизайнах первым всегда должно идти содержимое

Мы хотим, чтобы многие черты нашего дизайна оставались такими, как есть при отображении на разнообразных платформах и в областях просмотра (вместо того чтобы скрывать определенные части из виду с помощью выражения `display: none`

или вроде того). Однако также важно принимать во внимание порядок, в котором располагаются объекты. На данный момент, если взглянуть на очередность расположения врезки и основного содержимого в нашей разметке, понятно, что врезка всегда будет стремиться отобразиться раньше основной информации. Ясно, что для пользователя с более ограниченной областью просмотра основное содержимое должно отображаться ранее, чем врезка, ведь в противном случае он в первую очередь увидит второстепенную информацию.

Мы также могли бы (и, возможно, нам следовало бы) изменить местоположение основного содержимого, поместив его над навигационной областью. Благодаря этому для пользователей с наименьшими областями просмотра основное содержимое будет отображаться раньше, чем что-либо другое. Однако нам бы хотелось, чтобы в большинстве случаев навигационные элементы располагались вверху страницы, поэтому будет лучше, если мы просто изменим порядок расположения врезки и основного содержимого в HTML-коде: **сделаем так, чтобы основное содержимое шло раньше врезки**. Например, взгляните на следующий код:

```
<div id="sidebar">
  <p>Это врезка</p>
</div>
<div id="content">
  <p>Это содержимое</p>
</div>
```

Вместо предыдущего кода теперь у нас будет такой:

```
<div id="content">
  <p>Это содержимое</p>
</div>
<div id="sidebar">
  <p>Это врезка</p>
</div>
```

Несмотря на то что мы внесли изменения в разметку, в более крупных областях просмотра наша страница по-прежнему выглядит абсолютно так же, как и раньше, благодаря свойствам `float: left` и `float: right`, относящимся соответственно к врезке и основному содержимому. Однако на экране iPad основное содержимое теперь идет первым, а второстепенное (врезка) — вторым.

Итак, наша разметка структурирована в правильном порядке. Кроме того, я позаботился о добавлении и изменении дополнительных стилей, характерных для областей просмотра шириной 768 пикселей. Вот как теперь будет выглядеть соответствующий медиазапрос:

```
@media screen and (max-width: 768px) {
  #wrapper, #header, #footer, #navigation {
    width: 768px;
    margin: 0px;
  }
  #logo {
    text-align: center;
  }
  #navigation {
    text-align: center;
  }
}
```

```

    background-image: none;
    border-top-color: #bfbfbf;
    border-top-style: double;
    border-top-width: 4px;
    padding-top: 20px;
}
#navigation ul li a {
    background-color: #dedede;
    line-height: 60px;
    font-size: 40px;
}
#content, #sidebar {
    margin-top: 20px;
    padding-right: 10px;
    padding-left: 10px;
    width: 728px;
}
.oscarMain {
    margin-right: 30px;
    margin-top: 0px;
    width: 150px;
    height: 394px;
}
#sidebar {
    border-right: none;
    border-top: 2px solid #e8e8e8;
    padding-top: 20px;
    margin-bottom: 20px;
}
.sideBlock {
    width: 46%;
    float: left;
}
.overHyped {
    margin-top: 0px;
    margin-left: 50px;
}
}

```

Помните, что добавленные здесь стили начнут задействоваться, только когда речь будет идти об устройствах с областями просмотра шириной 768 пикселей или менее. А для более крупных областей просмотра они будут игнорироваться. Кроме того, поскольку эти стили идут в самом конце, они будут иметь больший приоритет в соответствующих ситуациях. Таким образом, более крупные области просмотра будут выглядеть так же, как и раньше. На устройствах с областями просмотра шириной 768 пикселей все будет выглядеть так, как показано на рис. 2.11.

Разумеется, здесь мы не собираемся завоевывать какие-либо награды за дизайн, однако с помощью всего лишь нескольких строк **CSS-кода в медиазапросе мы создали совершенно новый макет для областей просмотра с другой шириной. Как именно мы это сделали?**



Рис. 2.11. Страница проектируемого сайта в области просмотра шириной 768 пикселей

Прежде всего мы сбросили все области содержимого до полной ширины, указанной в медиазапросе, как показано в следующем фрагменте кода:

```
#wrapper, #header, #footer, #navigation {
    width: 768px;
    margin: 0px;
}
```

Затем нам оставалось лишь добавить стили для изменения эстетических свойств элементов. Например, приведенный далее фрагмент кода изменяет размер навигационной области, макет и фон, чтобы пользователям планшетных компьютеров

(или любым пользователям с областью просмотра шириной 768 пикселей или менее) было легче выбрать тот или иной навигационный элемент:

```
#navigation {  
  text-align: center;  
  background-image: none;  
  border-top-color: #bfbfbf;  
  border-top-style: double;  
  border-top-width: 4px;  
  padding-top: 20px;  
}  
#navigation ul li a {  
  background-color: #dedede;  
  line-height: 60px;  
  font-size: 40px;  
}
```

Теперь точно такое же содержимое, что мы видели раньше, отображается с использованием другого макета в зависимости от размеров текущей области просмотра. Медиазапросы — это отличная штука, не так ли? Устроим вечеринку. Пока вы сходите за шампанским, я посмотрю, как наша страница выглядит на экране моего iPhone... Вы можете увидеть это на рис. 2.12.

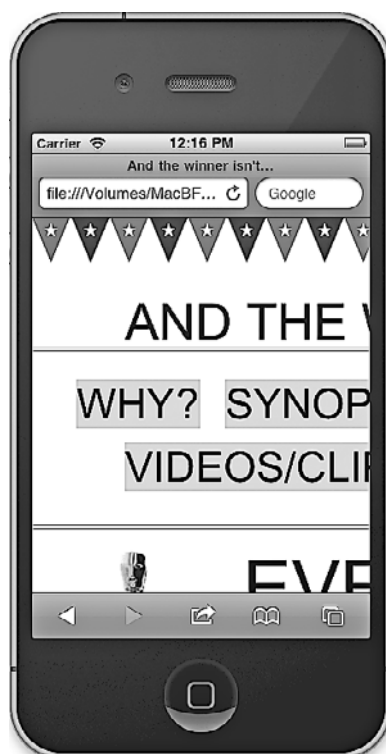


Рис. 2.12. Проектируемая страница на экране iPhone

2.7. Медиазапросы — это лишь часть решения

Ох... лучше положить лед обратно в морозилку. Наша работа явно еще очень далека от завершения. Созданная нами страница выглядит просто ужасно на моем iPhone в области просмотра шириной 320 пикселей. Наш медиазапрос делает именно то, что должен, применяя стили в зависимости от характеристик конкретного устройства. Однако проблема заключается в том, что он охватывает очень узкий диапазон областей просмотра. На экране любого устройства с областью просмотра шириной менее 768 пикселей содержимое будет обрезаться, как и на экране с областью просмотра шириной от 768 до 960 пикселей, поскольку в этом случае начнет задействоваться версия CSS-стилей, не связанных с медиазапросами. А как мы уже знаем, они не применяются, когда ширина области просмотра оказывается менее 960 пикселей (здесь я хватаюсь за голову и тяжело вздыхаю).

Нам необходим «резиновый» макет. Использование одних лишь медиазапросов для изменения дизайна — прекрасное решение, если у нас имеется конкретное, известное целевое устройство. Мы уже видели, насколько легко приспособить дизайн, чтобы он корректно отображался на экране iPad. Однако у такой стратегии есть серьезный недостаток, который заключается в том, что она не ориентирована на будущее, то есть может потерять свою актуальность. Сейчас при изменении размера нашей области просмотра дизайн достигает точки, когда в дело вступают соответствующие медиазапросы и конфигурация нашего макета изменяется. Однако затем он остается статичным до тех пор, пока не будет достигнута следующая контрольная точка, связанная с областью просмотра. Нам требуется нечто более подходящее, чем данное решение.

Добавление CSS-стилей, характерных для каждого конкретного изменения размеров области просмотра, не делает поправку на будущие устройства, а по-настоящему хорошим является тот дизайн, в который закладываются функции, ориентированные на будущее. На данном этапе наше решение является неполноценным. У нас получился скорее «настраиваемый» дизайн, нежели действительно адаптивный, к которому мы стремимся. Нам необходимо, чтобы дизайн проявлял гибкость до достижения следующей контрольной точки. Для этого нам потребуются перейти от жесткого, фиксированного макета к «резиновому».

2.8. Резюме

В этой главе мы выяснили, что собой представляют CSS3-медиазапросы, как включать их в CSS-файлы и как они могут помочь нам в стремлении создать адаптивный веб-дизайн. Мы также узнали, как заставить современные мобильные браузеры обрабатывать веб-страницы в такой же манере, как это делают их настольные аналоги, и вкратце поговорили о необходимости принимать во внимание принцип «сначала содержимое» при структурировании разметки. Кроме того, мы рассмотрели

способ, как сократить объем данных, наиболее экономичным путем используя в дизайне изображения.

Мы также выяснили, что медиазапросы могут обеспечивать лишь «настраиваемый» веб-дизайн, а не по-настоящему адаптивный. Медиазапросы — это существенный компонент адаптивного веб-дизайна, однако не менее важным является «резиновый» макет, позволяющий дизайну проявлять гибкость между контрольными точками, обрабатываемыми медиазапросами. В следующей главе мы поговорим о создании гибкой основы для нашего макета, пытаясь сделать плавными переходы между контрольными точками наших медиазапросов.

3 Использование «резиновых» макетов

Когда я впервые занялся созданием сайтов в конце 1990-х годов, макеты страниц строились на основании таблиц. В большинстве случаев экранное пространство разбивалось на фрагменты, задаваемые в процентах. Например, колонка слева, включающая навигационные элементы, могла занимать 20 % пространства, а область основного содержимого — 80 %. Такого большого разнообразия в размерах браузерных областей просмотра, какое мы наблюдаем сегодня, тогда не было, поэтому макеты нормально работали и масштабировались в рамках ограниченного диапазона областей просмотра. Никого не беспокоило, что на одном экране предложения выглядели немного иначе, чем на другом. Однако, когда на смену пришли дизайны на основе CSS, тот или иной веб-дизайн стал более единообразно отображаться на экранах разных устройств, подражая печатным изданиям. В результате этого перехода для многих людей (включая меня) пропорциональные макеты на долгие годы утратили значение в угоду их жестким, основанным на пикселах собратьям.

Как и все великие идеи и решения, пропорциональные макеты снова вернулись. Мода на малолитражные автомобили, волосы с перманентной завивкой (еще бы!) и расклешенные джинсы со временем возвращалась. Теперь пришло время возродиться и пропорциональным макетам.

В этой главе мы сделаем следующее:

- выясним, почему для адаптивных веб-дизайнов необходимы пропорциональные макеты;
- преобразуем показатели ширины элементов, основанные на пикселах, в пропорциональные процентные значения;
- преобразуем размеры верстки, основанные на пикселах, в эквивалентные значения в единицах em;
- разберемся, как определить контекст для любого элемента;
- узнаем, как заставить изображения гибко масштабироваться;
- научимся обеспечивать разные изображения для разных по размеру экранов;
- выясним, как медиазапросы могут работать с «резиновыми» изображениями и макетами;
- создадим адаптивный макет с нуля с использованием сеточной системы CSS.

3.1. Фиксированные макеты не ориентированы на будущее

Как я уже говорил, со времен дизайнов на основе таблиц мне редко требовалось прибегать к использованию пропорциональных макетов. Клиенты обычно просили меня написать HTML-разметку и CSS-код, которые лучше всего подходили для того или иного дизайна с шириной, почти всегда составлявшей 950–1000 пикселей. Если когда-либо при создании макета для него задавалась пропорциональная ширина (например, 90 %), то жалобы пользователей не заставляли себя ждать: «На моем экране все выглядит не так». Веб-страницы с фиксированными размерами на основе пикселей были самым простым решением.

Даже в более поздние времена, когда медиазапросы стали применяться для создания настраиваемых версий макетов, характерных для некоторых популярных устройств, например iPad и iPhone (что мы делали в главе 2), размеры по-прежнему могли основываться на пикселях, поскольку разработчики знали, какой именно в этом случае будет область просмотра. Однако, хотя многим из них нравилась возможность выставлять счета клиентам каждый раз, когда последним требовалось внести изменения в сайты для адаптации к современным новейшим устройствам, этот подход к разработке веб-страниц нельзя было назвать ориентированным на будущее. С ростом разнообразия областей просмотра потребовался способ адаптации веб-страниц для незнакомых устройств.

3.2. Почему для адаптивных веб-дизайнов необходимы пропорциональные макеты

Несмотря на то что медиазапросы — это невероятно мощный инструмент, у них имеются кое-какие ограничения. Любой дизайн с фиксированной шириной, для адаптации которого к разным областям просмотра используются лишь одни медиазапросы, будет просто «перескакивать» от одного набора CSS-правил, применяемых в медиазапросах, к другому без линейной прогрессии между ними. В главе 2 вы видели, что если ширина области просмотра попадала в промежуток между определенными значениями фиксированной ширины в наших медиазапросах (что может случаться с будущими незнакомыми устройствами и их областями просмотра), то дизайн требовал прибегать к горизонтальной прокрутке в браузере. Вместо этого мы хотим создать дизайн, который будет проявлять гибкость и хорошо смотреться в областях просмотра любых размеров, а не только в тех, что указаны в медиазапросе. Я перейду к главному, пропустив незначительные подробности. Нам необходимо преобразовать наш фиксированный, основанный на пикселях макет в «резиновый» пропорциональный. В результате элементы смогут масштабироваться относительно области просмотра, пока тот или иной медиазапрос не изменит стилизацию.

**ОБЪЕДИНЕНИЕ ПРОПОРЦИОНАЛЬНОГО МАКЕТА И МЕДИАЗАПРОСОВ**

Ранее я уже упоминал статью Итана Маркотта на тему адаптивного веб-дизайна, расположенную на сайте A List Apart (<http://www.alistapart.com/articles/responsive-web-design/>). Несмотря на то что инструменты, которые он задействовал («резиновый» макет и «резиновые» изображения, а также медиазапросы), не были новыми, его подход к использованию и объединению идей в одну согласованную методологию отличается новизной. Для многих людей, работающих в сфере веб-дизайна, его статья стала источником новых возможностей. Более того, описанные варианты разработки веб-страниц предложили лучшее из двух областей: способ создать «резиновый» гибкий дизайн на основе пропорционального макета, одновременно имея возможность ограничить гибкость элементов с помощью медиазапросов. Такое объединение образует ядро адаптивного веб-дизайна, создавая нечто действительно большее, чем простое соединение их частей.

3.3. Преобразование дизайна на основе фиксированного макета в дизайн на основе пропорционального макета

В обозримом будущем любая композиция дизайна, с которой вы столкнетесь или которую станете создавать, скорее всего, будет иметь фиксированные размеры. В настоящее время мы измеряем размеры, поля и т. д. в пикселях в графических файлах, открытых в Photoshop, Fireworks и пр. Затем мы переносим эти размеры непосредственно в CSS. То же самое касается размеров текста. Мы щелкаем на текстовом элементе в нашем любимом редакторе изображений, смотрим на размер шрифта, после чего вводим его (опять-таки зачастую измеряя в пикселях) в соответствующее CSS-правило. Как же нам преобразовать фиксированные размеры в пропорциональные?

Формула для запоминания

Возможно, я становлюсь слишком большим фанатом Итана Маркотта, однако на данном этапе для меня важно сделать еще один большой реверанс в его адрес (вероятно, это должен быть поклон, возможно, даже с преклонением колена). Мистер Маркотт внес вклад в замечательную книгу Дэна Седерхольма *Handcrafted CSS* («CSS ручной работы»), написав главу о «резиновых» сетках. В ней он привел простую и непротиворечивую формулу, позволяющую преобразовать пиксельные показатели фиксированной ширины в пропорциональные процентные значения:

ширина цели × ширина контекста = результат.

Немного напоминает уравнение? Не беспокойтесь, при создании адаптивных веб-дизайнов эта формула быстро станет вашим новым лучшим другом. Вместо того чтобы дальше говорить о теории, применим эту формулу на практике для преобразования текущего макета с фиксированными размерами, используемого на сайте And the winner isn't..., в «резиновый» макет с размерами на основе процентных значений.

Если помните, в главе 2 мы установили, что базовая разметочная структура нашего сайта выглядит так:

```
<div id="wrapper">
  <!-- верхний колонтитул и навигация -->
  <div id="header">
    <div id="navigation">
      <ul>
        <li><a href="#">navigation1</a></li>
        <li><a href="#">navigation2</a></li>
      </ul>
    </div>
  </div>
  <!-- врезка -->
  <div id="sidebar">
    <p>Это врезка</p>
  </div>
  <!-- содержимое -->
  <div id="content">
    <p>Это содержимое</p>
  </div>
  <!-- нижний колонтитул -->
  <div id="footer">
    <p>Это нижний колонтитул</p>
  </div>
</div>
```

Содержимое было добавлено позднее, однако здесь важно отметить CSS-код, который мы используем для задания ширины основных структурных элементов (header, navigation, sidebar, content и footer). Обратите внимание, что я не стал приводить многие из правил стилизации, чтобы мы смогли сосредоточиться на структуре:

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 960px;
}

#header {
  margin-right: 10px;
  margin-left: 10px;
  width: 940px;
}

#navigation
  padding-bottom: 25px;
  margin-top: 26px;
  margin-left: -10px;
  padding-right: 10px;
  padding-left: 10px;
```

```
width: 940px;
}

#navigation ul li {
  display: inline-block;
}

#content {
  margin-top: 58px;
  margin-right: 10px;
  float: right;
  width: 698px;
}

#sidebar {
  border-right-color: #e8e8e8;
  border-right-style: solid;
  border-right-width: 2px;
  margin-top: 58px;
  padding-right: 10px;
  margin-right: 10px;
  margin-left: 10px;
  float: left;
  width: 220px;
}

#footer {
  float: left;
  margin-top: 20px;
  margin-right: 10px;
  margin-left: 10px;
  clear: both;
  width: 940px;
}
```

На текущий момент все значения заданы в пикселах. Начнем с самого дальнего элемента и по порядку заменим показатели ширины всех элементов на пропорциональные процентные значения, используя формулу *ширина цели × ширина контекста = результат*.

Все содержимое располагается в элементе `<div>` с идентификатором `#wrapper`. Из приводившегося чуть ранее CSS-кода видно, что для его свойств `margin-right` и `margin-left` задано значение `auto`, а для `width` — значение `960px`. Как нам определить для самого дальнего элемента `<div>`, какой процент от ширины области просмотра должна составлять его ширина?

Задание контекста для пропорциональных элементов

Нам необходимо, чтобы что-то удерживалось на месте и стало контекстом для всех пропорциональных элементов (`content`, `sidebar`, `footer` и т. д.), которые согласно замыслу будут присутствовать в нашем дизайне. Следовательно, нам нужно задать

пропорциональное значение для ширины, которую будет иметь `<div>` с идентификатором `#wrapper` по отношению к ширине области просмотра. Зададим значение 96% и посмотрим, что будет. Вот измененное правило для элемента `<div>` с идентификатором `#wrapper`:

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 96%; /* Удерживание самого дальнего элемента <div> */
}
```

А вот как все это будет выглядеть в окне браузера (рис. 3.1).



Рис. 3.1. Страница, для которой задано пропорциональное значение для ширины

Пока все хорошо! Значение 96% замечательно работает в данном случае, хотя мы могли выбрать 100% или 90% — любое из них, которое показалось бы нам подходящим,

и обеспечить наиболее приятный с эстетической точки зрения дизайн в области просмотра.

Теперь переход от фиксированного макета к пропорциональному будет немного сложнее по мере нашего продвижения вперед. Сначала взглянем на верхний колонтитул. Вспоминаем формулу *ширина цели × ширина контекста = результат*. Наш элемент `<div>` с идентификатором `#header` (цель) располагается в `<div>` с идентификатором `#wrapper` (контекст). Следовательно, нужно взять значение ширины нашего элемента `<div>` с идентификатором `#header` (цель), равное 940 пикселям, и разделить его на значение ширины `<div>` с идентификатором `#wrapper`, которое равно 960 пикселям, в результате чего мы получим 0.979166667. Мы можем преобразовать этот результат в проценты, переместив точку, отделяющую десятичную дробь от целого числа, на два знака вправо, и получить процентное значение ширины, равное 97.9166667%. Добавим его в наш CSS-код:

```
#header {  
    margin-right: 10px;  
    margin-left: 10px;  
    width: 97.9166667%; /* 940 × 960 */  
}
```

Поскольку элементы `<div>` с идентификаторами `#navigation` и `#footer` имеют одинаковую объявленную ширину, мы можем заменить пиксельные значения их ширины на аналогичные значения в процентах.

И наконец, прежде чем мы снова взглянем на нашу страницу в браузере, займемся элементами `<div>` с идентификаторами `#content` и `#sidebar`. Поскольку ширина контекста осталась прежней (960 пикселей), нам нужно лишь разделить значение ширины нашей цели на показатель ширины контекста. На текущий момент ширина нашего элемента `<div>` с идентификатором `#content` составляет 698 пикселей, поэтому, если разделить это число на 960, получится значение 0.727083333. Переместим точку, отделяющую десятичную дробь от целого числа, на два знака вправо и получим 72.7083333% — это ширина элемента `<div>` с идентификатором `#content` в процентном выражении. Ширина нашей врезки на данный момент равна 220 пикселям, однако также необходимо принимать во внимание границу шириной 2 пиксела. Я не хочу, чтобы ширина границы справа увеличивалась или уменьшалась пропорционально, то есть она должна остаться 2 пиксела. Для этого мне нужно вычесть эту величину из значения ширины врезки. Таким образом, для нашей врезки я вычел 2 пиксела из значения ее ширины, а затем выполнил вычисление по приводившейся ранее формуле. Я разделил ширину цели (теперь она равна 218 пикселям) на ширину контекста (960 пикселей) и получил 0.227083333. Если переместить точку, отделяющую десятичную дробь от целого числа, на два знака вправо, то получится значение ширины врезки в процентах, равное 22.7083333%. После преобразования всех значений ширины в пикселях в показатели, выраженные в процентах, наш CSS-код будет выглядеть так:

```
#wrapper {  
    margin-right: auto;  
    margin-left: auto;
```

```
width: 96%; /* Удержание самого дальнего элемента <div> */
}

#header {
    margin-right: 10px;
    margin-left: 10px;
    width: 97.9166667%; /* 940 × 960 */
}

#navigation {
    padding-bottom: 25px;
    margin-top: 26px;
    margin-left: -10px;
    padding-right: 10px;
    padding-left: 10px;
    width: 72.7083333%; /* 698 × 960 */
}

#navigation ul li {
    display: inline-block;
}

#content {
    margin-top: 58px;
    margin-right: 10px;
    float: right;
    width: 72.7083333%; /* 698 × 960 */
}

#sidebar {
    border-right-color: #e8e8e8;
    border-right-style: solid;
    border-right-width: 2px;
    margin-top: 58px;
    margin-right: 10px;
    margin-left: 10px;
    float: left;
    width: 22.7083333%; /* 218 × 960 */
}

#footer {
    float: left;
    margin-top: 20px;
    margin-right: 10px;
    margin-left: 10px;
    clear: both;
    width: 97.9166667%; /* 940 × 960 */
}
```

На рис. 3.2 показано, как в результате этого наша страница будет выглядеть в Firefox с областью просмотра шириной примерно 1000 пикселей.



Рис. 3.2. Измененная страница в браузере Firefox

Пока все хорошо. Теперь пойдем дальше и заменим все значения, равные 10 пикселям, которые были использованы в коде для задания отступов и полей, на эквивалентные им пропорциональные значения, задействуя прежнюю формулу *ширина цели × ширина контекста = результат*. Поскольку для всех 10-пиксельных значений имеет место один и тот же контекст с шириной, равной 960 пикселям, в процентном выражении ширина составит 1.0416667% (10 × 960).



А ПОЧЕМУ БЫ ПРОСТО НЕ ОКРУГЛЯТЬ ТАКИЕ ЧИСЛА?

Некоторые критики методологии адаптивного веб-дизайна (например, загляните на страницу по адресу <http://tripleodeon.com/2010/10/not-a-mobile-web-merely-a-320px-wide-one/>) утверждают, что вводить в таблицы стилей числа вроде .550724638em просто глупо. Вы можете задаться вопросом: «Почему бы не округлять такие числа до более удобных для восприятия?»

Контраргумент заключается в том, что такие числа являются более точными ответами на задаваемые вопросы. Если предоставить браузеру наиболее точный ответ, он сможет отобразить этот ответ в наиболее точной форме. Если в школе вы бодрствовали более чем на двух уроках математики, то, я уверен, слышали о золотом сечении (http://en.wikipedia.org/wiki/Golden_ratio). Это математическое отношение, которое встречается и используется почти в каждой из известных нам дисциплин и выражается примерно как 1:1,61803398874989 (если вы хотите узнать, какие 10 000 десятичных знаков следуют после запятой в данной пропорции, то загляните по адресу <http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/phi10000dps.txt>). Этот показатель никоим образом нельзя назвать лаконичным, однако он весьма важен. Если в случае с золотым сечением допускаются столь точные измерения, то я склонен полагать, что это возможно и в создаваемых нами дизайнах.

Все по-прежнему отлично выглядит в области просмотра с теми же размерами, что и раньше. Однако навигационная область ведет себя не так, как следует. Если немного увеличить размеры области просмотра, то ссылки будут занимать уже две строки (рис. 3.3).



Рис. 3.3. При увеличении области просмотра ссылки переносятся на новую строку

Более того, если расширить область просмотра, то поля между ссылками не увеличатся пропорционально. Взглянем на CSS-код, ассоциированный с навигацией, и попробуем выяснить причину:

```
#navigation {
  padding-bottom: 25px;
  margin-top: 26px;
  margin-left: -1.0416667%; /* 10 × 960 */
  padding-right: 1.0416667%; /* 10 × 960 */
  padding-left: 1.0416667%; /* 10 × 960 */
  width: 97.9166667%; /* 940 × 960 */
  background-repeat: repeat-x;
  background-image: url(../img/atwiNavBg.png);
  border-bottom-color: #bfbfbf;
  border-bottom-style: double; border-bottom-width: 4px;
}

#navigation ul li {
  display: inline-block;
}

#navigation ul li a {
  height: 42px;
  line-height: 42px;
  margin-right: 25px;
  text-decoration: none;
  text-transform: uppercase;
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;
  font-size: 27px;
  color: black;
}
```

Похоже, в нашем третьем правиле `#navigation ul li` по-прежнему задано поле с использованием пиксельного значения, которое в данном случае равно 25. Исправим это, прибегнув к нашей проверенной формуле. Поскольку элемент `<div>` с идентификатором `#navigation` имеет ширину 940 пикселей, результат будет равен 2.6595745%. Таким образом, мы изменим правило так, как показано далее:

```
#navigation ul li a {
  height: 42px;
  line-height: 42px;
  margin-right: 2.6595745%; /* 25 × 940 */
  text-decoration: none;
  text-transform: uppercase;
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;
  font-size: 27px;
  color: black;
}
```

Это было довольно просто! Теперь проверим, все ли хорошо выглядит в браузере (рис. 3.4)...



Рис. 3.4. Страница после изменения ширины правого поля

Постойте, это не совсем то, к чему мы стремились. Хорошо, ссылки больше не растягиваются на две строки, однако четко видно, что пропорциональных полей между ними нет. Навигационные ссылки выглядят как одно длинное слово, которое нельзя найти в словаре...

Всегда важно помнить о контексте

Если снова вспомнить нашу формулу (*ширина цели × ширина контекста = результат*), то можно понять причину возникшей проблемы. Здесь она заключается в контексте. Вот соответствующая разметка:

```
<div id="navigation">
  <ul>
    <li><a href="#">Why?</a></li>
```

```

<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>

```

Как вы можете видеть, ссылки `` заключены в теги ``. Они представляют собой контекст для наших пропорциональных полей. Взглянув на CSS-код для тегов ``, можно увидеть, что там не задано никаких значений ширины:

```
#navigation ul li { display: inline-block; }
```

Как это часто бывает, оказывается, решение данной проблемы возможно разными способами. Мы могли бы добавить конкретное значение ширины в теги ``, однако оно должно быть либо значением фиксированной ширины в пикселах, либо значением, выраженным в процентах от ширины элемента-контейнера (`<div>` с идентификатором `#navigation`), но ни одно из них не обеспечит гибкости для текста, который в конечном счете будет там располагаться.

Вместо этого мы могли бы откорректировать CSS-код для тегов ``, изменив значение `display` с `inline-block` на `inline`:

```
#navigation ul li {
  display: inline;
}
```

Выбор в пользу `display: inline;` (в результате этого элементы `` перестанут вести себя как блочные элементы) также приведет к тому, что строка навигации будет обрабатываться горизонтально в более ранних версиях Internet Explorer (версиях 6 и 7), имеющих проблемы с `inline-block`. Однако я поклонник значения `inline-block`, поскольку оно обеспечивает больший контроль над полями и отступами в современных браузерах. Я лучше оставляю для тегов `` свойство `display: inline-block;` (и, возможно, позднее добавлю переопределяющий стиль для Internet Explorer версии 6 и 7), а также перемещу правило для задания полей на основе процентов из тега `<a>` (который не имеет явного контекста) в блок-контейнер ``. Правила с внесенными изменениями теперь будут выглядеть так:

```
#navigation ul li {
  display: inline-block;
  margin-right: 2.6595745%; /* 25 × 940 */
}

#navigation ul li a {
  height: 42px;
  line-height: 42px;
  text-decoration: none;
  text-transform: uppercase;
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;
  font-size: 27px;
  color: black;
}
```

На рис. 3.5 показано, как наша страница будет выглядеть в браузере с областью просмотра шириной 1200 пикселей.



Рис. 3.5. Исправленная страница в области просмотра шириной 1200 пикселей

Итак, ситуация с навигацией постепенно улучшается, однако у нас все еще остается проблема — ссылки начинают растягиваться на две строки при уменьшении ширины области просмотра вплоть до того, как она окажется менее 768 пикселей, и тогда начинает работать медиазапрос, написанный нами в главе 2. Он переопределяет текущие стили, относящиеся к навигации. Прежде чем мы приступим к решению проблемы с навигационными ссылками, преобразуем все показатели размеров верстки из пиксельных значений в величины, выраженные в пропорциональных единицах em. В результате мы взглянем на еще одну очевидную, но часто игнорируемую проблему и сделаем так, чтобы наши изображения масштабировались пропорционально дизайну.

3.4. Использование единиц em вместо пикселей для задания размеров верстки

В прошлом веб-дизайнеры для задания размеров верстки в основном использовали единицы em вместо пикселей, поскольку более ранние версии Internet Explorer были не способны изменять масштаб текста, величина шрифта которого задана в пикселях. Современные браузеры уже могут изменять масштаб текста на экране, даже если его размер был указан в пикселях. Так почему же использование единиц em вместо пикселей по-прежнему предпочтительнее? На то есть две очевидные причины: во-первых, любой, кто до сих пор использует **Internet Explorer 6**, автоматически получит возможность изменять масштаб текста, а во-вторых, это значительно облегчит жизнь вам как разработчику. Размер в em зависит от размера соответствующего контекста. Если мы зададим для тега `<body>` размер шрифта 100% и стилизуем остальную верстку с использованием em, то все эти шрифты будут подвергаться воздействию со стороны начального объявления. В результате, например, если после того, как мы зададим весь необходимый шрифтовой набор, наш клиент попросит сделать все шрифты немного больше, нам потребуется изменить лишь размер шрифта для `<body>`, а все остальные значения верстки изменятся соответственно этому.

Используя нашу прежнюю формулу *ширина цели × ширина контекста = результат*, я преобразую все размеры шрифтов на основе пикселей в значения, выраженные в em. Вам стоит знать, что все современные настольные браузеры используют 16px в качестве размера шрифта по умолчанию (если только другое значение не будет задано явным образом). Следовательно, изначальное применение любого из приведенных далее правил к тегу `<body>` обеспечит одинаковый результат:

```
font-size: 100%;  
font-size: 16px;  
font-size: 1em;
```

Например, первое значение размера шрифта на основе пикселей в нашей таблице стилей определяет величину символов в названии сайта And the winner isn't..., указанном вверху слева:

```
#logo {  
  display: block;  
  padding-top: 75px;  
  color: #0d0c0c;  
  text-transform: uppercase;  
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;  
  font-size: 48px;  
}
```

```
#logo span { color: #dfdada; }
```

Следовательно, $48 \times 16 = 3$. Таким образом, наш стиль изменится:

```
#logo {  
  display: block;  
  padding-top: 75px;  
  color: #0d0c0c;  
  text-transform: uppercase;  
  font-family: Arial, "Lucida Grande", Verdana, sans-serif;  
  font-size: 3em; /* 48 × 16 = 3 */  
}
```

Аналогичную логику вы можете применить и далее в коде. Если на каком-то этапе что-то начнет «барахлить», вероятно, это происходит из-за того, что контекст для вашей цели изменился. Например, взгляните на элемент `<h1>` в разметке нашей страницы:

```
<h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
```

Наш новый CSS-код на основе `em` будет выглядеть так:

```
#content h1 {  
  font-family: Arial, Helvetica, Verdana, sans-serif;  
  text-transform: uppercase;  
  font-size: 4.3125em; } /* 69 × 16 */  
  
#content h1 span {  
  display: block;  
  line-height: 1.052631579em; /* 40 × 38 */  
  color: #757474;  
  font-size: .550724638em; /* 38 × 69 */  
}
```

Как вы можете здесь видеть, размер шрифта (который был равен 38 пикселям) элемента `` зависит от размера шрифта родительского элемента (который был равен 69 пикселям). Более того, для свойства `line-height` (имевшего значение 40px) значение задается в зависимости от размера шрифта как такового (который был равен 38 пикселям).



ЧТО ЖЕ ТАКОЕ EM?

Термин «`em`» определяет способ выражения буквы М в письменной форме и произносится как есть. Исторически сложилось так, что буква М стала использоваться для установления размера шрифта из-за того, что является самой широкой из всех букв. В настоящее время `em` как единица измерения определяет соотношение ширины/высоты конкретной буквы и размера конкретного шрифта в пунктах.

Итак, теперь наша структура изменяет свои размеры и мы преобразовали значения на основе пикселей в показатели, выраженные в `em`. Однако нам еще нужно выяснить, как масштабировать изображения по мере изменения размеров области просмотра, так что сейчас и займемся рассмотрением этой темы.

3.5. «Резиновые» изображения

Заставить изображения масштабироваться пропорционально «резиновому» дизайну для современных браузеров (включая **Internet Explorer версии 7 и выше**) не составляет особого труда. Для этого нужно лишь объявить в CSS следующее:

```
img {  
    max-width: 100%;  
}
```

В результате все изображения будут масштабироваться вплоть до 100 % ширины их элемента-контейнера. Более того, аналогичный атрибут и свойство могут быть применены к другим мультимедиа. Например:

```
img,object,video,embed {  
    max-width: 100%;  
}
```

Они тоже будут масштабироваться, не считая нескольких примечательных исключений вроде видео в теге `<iframe>` с сайта YouTube. О том, как с ними справиться, мы поговорим в главе 4. А сейчас сосредоточимся на изображениях, поскольку принцип будет тем же, что и раньше, независимо от мультимедиа.

При использовании этого подхода необходимо учитывать несколько важных моментов. Во-первых, он предполагает упреждающее планирование: добавляемые изображения должны быть достаточно велики для того, чтобы масштабироваться соразмерно крупным областям просмотра. Это приводит нас к следующему, возможно, более важному моменту. Независимо от того, каким будет размер области просмотра или устройство, используемое для обозрения сайта, все равно придется загружать большие изображения, даже если в области просмотра на определенных устройствах пользователям потребуется увидеть то или иное изображение размером всего лишь 25 % от его фактической величины. В некоторых случаях проблема заключается в скорости канала подключения пользователя, поэтому мы вскоре вернемся ко второму из описанных выше моментов. А пока позаботимся о том, чтобы наши изображения масштабировались.

Как сделать так, чтобы изображения масштабировались относительно области просмотра

Представьте, что в нашей врезке размещаются постеры двух замечательных картин и двух фильмов, которые можно назвать полной чепухой (здесь мы не будем дискутировать на эту тему). Соответствующая разметка на данный момент выглядит так:

```
<!-- врезка -->  
<div id="sidebar">  
    <div class="sideBlock unSung">
```

```

<h4>Unsung heroes...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
<div class="sideBlock overHyped">
<h4>Overhyped nonsense...</h4>
<a href="#"></a>
<a href="#"></a>
</div>
</div>

```

Несмотря на то что мы добавили объявление `max-width: 100%` в элемент `img` в CSS-коде, ничего не изменилось и изображения не масштабируются по мере расширения области просмотра (рис. 3.6).



Рис. 3.6. После внесенных изменений изображения по-прежнему не масштабируются

Причина здесь заключается в том, что я явным образом указал ширину и высоту изображений в разметке:

```

```

Еще одна мальчишеская ошибка! Необходимо откорректировать разметку, ассоциированную с изображениями, удалив атрибуты `height` и `width`:

```

```

Посмотрим, что это нам даст, обновив страницу в окне браузера (рис. 3.7).



Рис. 3.7. Изображения на странице увеличились

Что ж, внесенные исправления определенно сработали! Однако возникла другая проблема. Поскольку изображения масштабируются с целью заполнить 100 % ширины их элемента-контейнера, каждое заполняет всю врезку по ширине. Как и всегда, решить эту проблему можно разными способами...

Приоритетные правила для конкретных изображений

Можно было бы добавить класс для каждого изображения, как показано в приведенном далее фрагменте кода:

```

```

Затем мы могли бы задать особое правило для ширины. Однако вместо этого оставим нашу разметку как есть и используем CSS-приоритеты для переопределения

существующего правила `max-width` другим, более приоритетным правилом для изображений во врезке:

```
img {
  max-width: 100%;
}

.sideBlock img {
  max-width: 45%;
}
```

На рис. 3.8 показано, как наша страница теперь будет выглядеть в браузере.



Рис. 3.8. Обновленная страница

Подобное использование CSS-приоритетов также позволяет нам получить еще больший контроль над шириной всех остальных изображений и мультимедиа. Кроме того, из главы 5 вы узнаете, как новые мощные CSS3-селекторы дают возможность нацеливаться практически на любой элемент без дополнительной разметки или применения JavaScript-фреймворков вроде библиотеки jQuery для выполнения этой тяжелой работы.

Для каждого из изображений, расположенных по два во врезке, я решил задать ширину величиной 45 %, поскольку знаю, что позднее потребуете добавить не-

большое поле между рисунками, благодаря чему получится два изображения, ширина которых в сумме будет составлять 90 %, то есть у нас останется небольшое свободное пространство (10 %).

Теперь, когда с рисунками во врезке все в порядке, удалим атрибуты `width` и `height` из разметки, описывающей изображение статуэтки «Оскар». Однако если не задать для этого рисунка значения пропорциональной ширины, он не будет масштабироваться. Я откорректировал соответствующий CSS-код и указал величину пропорциональной ширины, воспользовавшись для ее вычисления старой доброй и проверенной формулой *ширина цели × ширина контекста = результат*.

```
.oscarMain {
  float: left;
  margin-top: -28px;
  width: 28.9398281%; /* 698 × 202 */
}
```

«Притормаживаем» «резиновые» изображения

Итак, теперь наши изображения соответствующим образом масштабируются при увеличении и уменьшении области просмотра. Однако если область просмотра увеличится настолько, что при масштабировании рисунка будут превышены его естественные размеры, то все станет выглядеть очень некрасиво. Взгляните на изображение статуэтки «Оскар» в области просмотра шириной больше 1900 пикселей (рис. 3.9).



Рис. 3.9. Страница нашего сайта в области просмотра шириной более 1900 пикселей

Фактическая ширина рисунка `oscar.png` составляет 202 пикселя. Однако если ширина области просмотра окажется более 1900 пикселей и этот рисунок будет пропорционально отмасштабирован, то при отображении его ширина составит

выше 300 пикселей. Мы можем легко «притормозить» рисунок, определив еще одно, более приоритетное правило:

```
.oscarMain {
  float: left;
  margin-top: -28px;
  width: 28.9398281%; /* 698 × 202 */
  max-width: 202px;
}
```

В результате изображение `oscar.png` сможет масштабироваться благодаря общему правилу, которое указано для него, однако ширина рисунка никогда не будет выходить за пределы значения, заданного для более приоритетного свойства `max-width` в приведенном выше коде. Вот как наша страница будет выглядеть после применения такого правила (рис. 3.10).



Рис. 3.10. Рисунок теперь отображается корректно

Невероятно универсальное свойство `max-width`

Чтобы предотвратить неограниченное увеличение разных объектов, можно также задать свойство `max-width` для всего элемента `<div>` с идентификатором `#wrapper`, как показано далее:

```
#wrapper {
  margin-right: auto;
  margin-left: auto;
  width: 96%; /* Удерживание самого дальнего элемента <div> */
  max-width: 1414px;
}
```


Это означает, что дизайн будет масштабироваться до ширины, составляющей 96 % от ширины области просмотра, но никогда не превысит 1414 пикселей (я выбрал значение 1414px, поскольку в большинстве современных браузеров оно приводит к тому, что видимая полоса из флагов обрезается таким образом, что в ее конце остается целый флаг, а не его половинка). На рис. 3.11 показано, как наша страница будет выглядеть в области просмотра шириной примерно 1900 пикселей.



Рис. 3.11. Обновленная страница в области просмотра шириной около 1900 пикселей

Ясно, что все это лишь варианты. Однако они подтверждают универсальность «резиновой» сетки, а также то, что мы можем управлять потоком, используя лишь несколько более приоритетных объявлений.

3.6. Обеспечение разных по величине изображений для разных по размеру экранов

Итак, размеры наших изображений изменяются так, как нужно, и теперь мы знаем, как ограничить размеры конкретных рисунков при их выводе на экран, если посчитаем это необходимым. Однако ранее в этой главе мы затронули проблему, связанную с масштабированием картинок. Физические размеры рисунков должны быть больше размеров при отображении, чтобы они хорошо смотрелись при воспроизведении на разных по величине экранах. В противном случае они будут выглядеть ужасно. Из-за этого размер файла рисунка почти всегда бывает больше, чем нужно для обеспечения его нормального отображения на экранах предполагаемой величины.

Многие люди бьются над решением этой проблемы, пытаясь обеспечить меньшие по размеру изображения для небольших экранов. Первым примечательным примером стало решение под названием Responsive Images (http://filamentgroup.com/lab/responsive_images_experimenting_with_context_aware_image_sizing/). Однако недавно я перешел на Adaptive Images от Мэтта Уилкокса (Matt Wilcox) (<http://adaptive-images.com>). Решение от Filament Group требовало внесения изменений в разметку, связанную с изображениями, а решение от Мэтта этого не требует и автоматически генерирует рисунки с измененными (меньшими) размерами, беря за основу полноразмерные изображения, уже указанные в разметке. Таким образом, это решение позволяет изменять размеры рисунков и при необходимости поставлять их пользователям на основании ряда контрольных точек, связанных с размерами экранов. Посмотрим, как устанавливается Adaptive Images.

Установка Adaptive Images

Решение Adaptive Images требует наличия Apache 2, PHP 5.x и GD Lib. Таким образом, чтобы увидеть его преимущества, вам потребуется разрабатывать сайт на соответствующем сервере. Скачайте ZIP-файл (рис. 3.12).

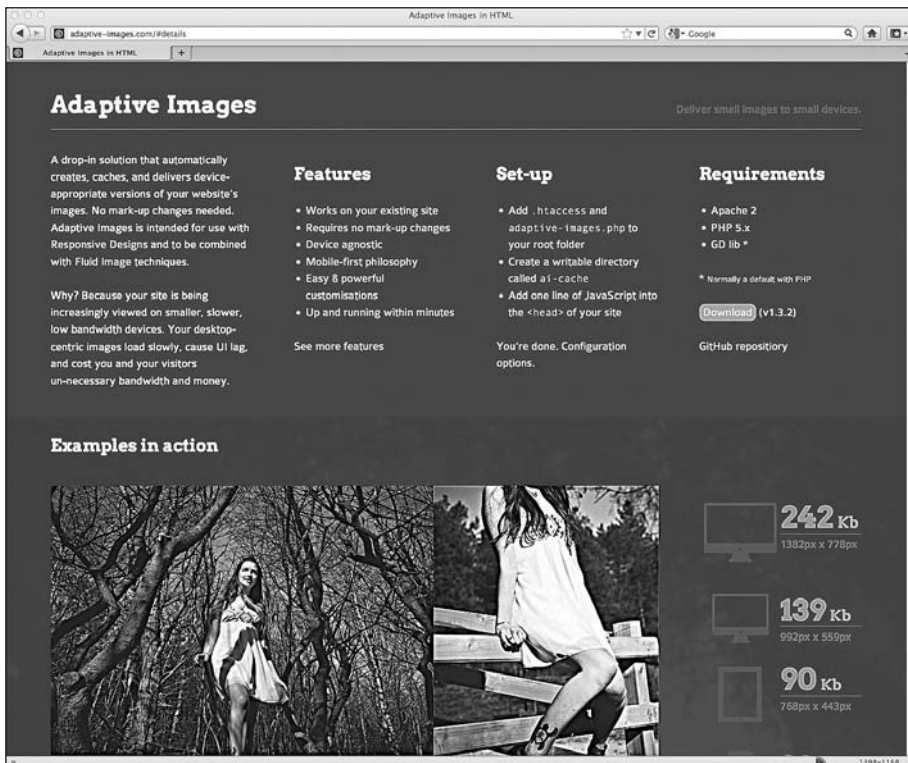


Рис. 3.12. Страница, с которой можно загрузить Adaptive Images

Разархивируйте ZIP-файл и скопируйте файлы `adaptive-images.php` и `.htaccess` в корневой каталог своего сайта. Если у вас в корневом каталоге уже есть файл `.htaccess`, не перезаписывайте его. Вместо этого прочтите дополнительную информацию в файле `instructions.htm`, который включен в архив.

Теперь создайте папку с именем `ai-cache` в корневом каталоге своего сайта (рис. 3.13).

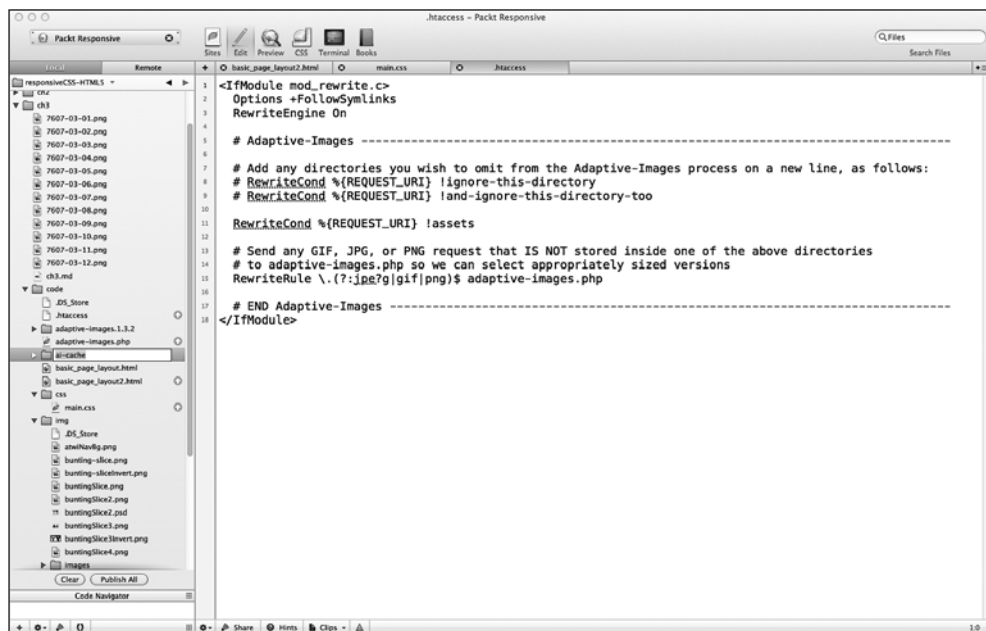


Рис. 3.13. Создание папки `ai-cache`

Используя свой любимый FTP-клиент, задайте разрешения на запись 777 (рис. 3.14).

Теперь скопируйте приведенный далее JavaScript-код в тег `<head>` каждой страницы, которая нуждается в использовании Adaptive Images:

```
<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+';
path='/';</script>
```

Если вы не используете HTML5 (мы перейдем к нему в следующей главе), но хотите, чтобы ваша страница прошла валидацию, потребуется добавить атрибут `type`. В результате сценарий должен будет выглядеть так:

```
<script type="text/javascript">document.cookie='resolution='+Math. max(screen.
width,screen.height)+'; path='/';</script>
```

Важно, чтобы JavaScript-код располагался внутри тега `<head>` (желательно — первая часть сценария), поскольку он должен запускаться раньше, чем закончится

загрузка страницы, и до того, как будут запрошены какие-либо изображения. Вот JavaScript-код, добавленный в раздел `<head>` сайта, над которым мы работаем:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0" />
<title>And the winner isn't...</title>
<script type="text/javascript">document.cookie='resolution='+Math. max(screen.
width,screen.height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
```

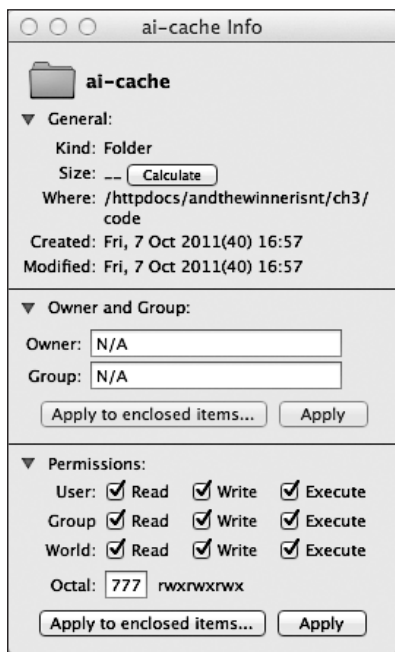


Рис. 3.14. Установка разрешения на запись

Размещение фоновых изображений в другом месте

Раньше я обычно размещал все свои изображения (используемые как в качестве фоновых элементов CSS, так и встроенных изображений, вставленных в разметку) в одной папке с именем, например, `images` или `img`. Однако если вы применяете Adaptive Images, то изображения, используемые через CSS в качестве фоновых (или любые другие изображения, размеры которых должны оставаться без изме-

нений), рекомендуется размещать в другом каталоге. **Adaptive Images по умолчанию** определяет папку с именем **assets** для хранения изображений, размеры которых не должны изменяться. Следовательно, если вы хотите, чтобы какие-либо рисунки не меняли своих размеров, добавляйте их в эту папку. Если вы решите использовать папку с другим именем (или несколько папок), то внесите следующие изменения в файл `.htaccess`:

```
<IfModule mod_rewrite.c>
  Options +FollowSymlinks
  RewriteEngine On
  # Adaptive-Images -----

  RewriteCond %{REQUEST_URI} !assets
  RewriteCond %{REQUEST_URI} !bkg

  # Send any GIF, JPG, or PNG request that IS NOT stored inside one of the above
  # directories
  # to adaptive-images.php so we can select appropriately sized versions
  RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php

  # END Adaptive-Images -----
</IfModule>
```

В примере мы указали, что не хотим, чтобы изображения, располагающиеся в каталоге **assets** или **bkg**, адаптировались. И наоборот, если вы хотите, чтобы адаптировались только изображения, расположенные в определенных папках, то не указывайте восклицательный знак в правиле. Например, если бы я захотел, чтобы адаптировались только изображения в подпапке моего сайта, носящей имя **andthewinnerisnt**, то мне потребовалось бы отредактировать файл `.htaccess` следующим образом:

```
<IfModule mod_rewrite.c>
  Options +FollowSymlinks
  RewriteEngine On

  # Adaptive-Images -----

  RewriteCond %{REQUEST_URI} andthewinnerisnt

  # Send any GIF, JPG, or PNG request that IS NOT stored inside one of the above
  # directories
  # to adaptive-images.php so we can select appropriately sized versions
  RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php

  # END Adaptive-Images -----
</IfModule>
```

Вот и все. Наиболее легкий способ проверить, что все работает, — вставить большое изображение на страницу, а затем зайти на нее со смартфона. Если вы

взгляните на содержимое своей папки `ai-cache` в FTP-клиенте, то увидите файлы и каталоги в папках, связанных с контрольными точками, например `480` (рис. 3.15).

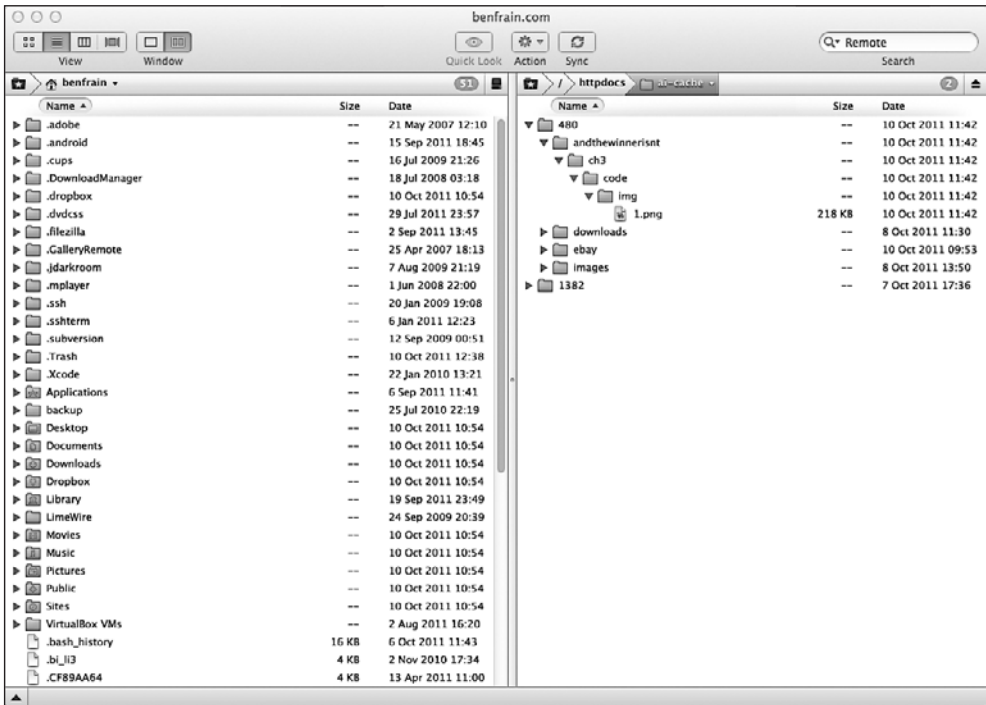


Рис. 3.15. Содержимое папки `ai-cache`

Работа Adaptive Images не ограничивается статичными сайтами. Это решение может использоваться наряду с системами управления содержимым, кроме того, существуют обходные пути для ситуаций, когда нельзя применять JavaScript. Благодаря Adaptive Images мы можем обеспечивать абсолютно разные по размеру изображения в зависимости от величины экранов устройств, снижая требования к скорости канала подключения для устройств, пользователи которых не увидят никаких преимуществ в том, что мы будем по умолчанию предоставлять им полно-размерные изображения.

3.7. Где «резиновые» сетки и медиазапросы объединяются друг с другом

Если помните, ранее в этой главе отмечалось, что наши навигационные ссылки начинают растягиваться на две строки, когда область просмотра становится определенной ширины. Мы можем решить эту проблему, используя медиазапросы. Если ссылки умещаются в одну строку при ширине области просмотра 1060 пик-

слов, однако начинают растягиваться на две строки, когда область просмотра становится шириной 768 пикселей (при этом вступает в действие написанный нами ранее медиазапрос), то зададим дополнительные стили шрифтов для применения в тот момент, когда показатель ширины области просмотра будет находиться в промежутке между упомянутыми чуть выше значениями:

```
@media screen and (min-width: 1001px) and (max-width: 1080px) {  
#navigation ul li a { font-size: 1.4em; }  
}  
@media screen and (min-width: 805px) and (max-width: 1000px) {  
#navigation ul li a { font-size: 1.25em; }  
}  
@media screen and (min-width: 769px) and (max-width: 804px) {  
#navigation ul li a { font-size: 1.1em; }  
}
```

Как видите, мы сделали так, что размер шрифта будет изменяться в зависимости от ширины области просмотра и в результате ссылки всегда будут умещаться в одну строку в диапазоне значений ширины области просмотра от 769 пикселей до бесконечности. Это еще одно доказательство симбиоза медиазапросов и «резиновых» макетов: медиазапросы сглаживают шероховатости в «резиновых» макетах, а «резиновые» макеты облегчают переход от одного набора определенных стилей в том или ином медиазапросе к другому.

3.8. Сеточные системы CSS

Сеточные системы/фреймворки CSS — это потенциально спорный предмет разговора. Одни дизайнеры рекомендуют их, а другие ругают. Чтобы минимизировать количество писем с угрозами и оскорблениями в мой адрес, я скажу, что занимаю нейтральную позицию в этом вопросе. Хотя мне понятно, почему некоторые разработчики считают их лишними и неудобными из-за необходимости писать дополнительный код, я все же высоко ценю фреймворки за то, что они позволяют быстро создать прототип макета.

Вот несколько фреймворков CSS, которые предлагают разную степень «адаптивной» поддержки:

- Semantic (<http://semantic.gs>);
- Skeleton (<http://getskeleton.com>);
- Less Framework (<http://lessframework.com>);
- 1140 CSS Grid (<http://cssgrid.net>);
- Columnal (<http://www.columnal.com>).

Среди перечисленных я предпочитаю сеточную систему Columnal, поскольку она наряду с медиазапросами включает встроенную «резиновую» сетку, а также задействует те же CSS-классы, что и 960.gs — популярная сеточная система с фиксированной шириной, с которой хорошо знакомо большинство разработчиков и дизайнеров.



АЛФА, ОМЕГА И ДРУГИЕ ОБЩИЕ СЕТОЧНЫЕ КЛАССЫ

Многие сеточные системы CSS задействуют CSS-классы для решения повседневных задач, связанных с макетами. Классы с именами `row` и `container` не требуют пояснений, однако существует и множество других классов, поэтому всегда проверяйте документацию ко всем сеточным системам на предмет любых классов, которые могут облегчить вам жизнь. Например, к числу прочих типичных классов, используемых в сеточных системах CSS, относятся `alpha` и `omega` — для первого и последнего элементов в строке соответственно (эти классы убирают отступы и поля), а также `.col_x`, где `x` — количество колонок, на которое должен простирается соответствующий элемент (например, `.col_6` означает шесть колонок).

Быстрое создание сайта с использованием сеточной системы. Допустим, мы еще не создавали нашу «резиновую» сетку и не писали медиазапросов. Нам предоставили композитный PSD-файл с дизайном домашней страницы сайта *And the winner isn't...* и сказали как можно быстрее подготовить базовую макетную структуру с использованием HTML и CSS. Посмотрим, может ли сеточная система Columnal помочь нам решить эту задачу.

Взглянув на исходный PSD-файл, можно легко понять, что макет базировался бы на 16 колонках. Однако сеточная система Columnal поддерживает не более 12 колонок, поэтому наложим на дизайн из PSD-файла 12 колонок вместо первоначальных 16 (рис. 3.16).

Скачав ZIP-файл Columnal и разархивировав его, мы сделаем дубликат уже имеющейся страницы, а затем укажем в разделе `<head>` ссылку на `columnal.css` вместо `main.css`. Для создания визуальной структуры с использованием Columnal очень важно добавить правильные классы. Вот как выглядит полная разметка нашей страницы на текущий момент:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0" />
<title>And the winner isn't...</title>
<script type="text/javascript">document.cookie='resolution='+Math. max(screen.
width.screen.height)+'; path='/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
  <!-- верхний колонтитул и навигация -->
  <div id="header">
    <div id="logo">And the winner is<span>n't...</span></div>
    <div id="navigation">
      <ul>
        <li><a href="#">Why?</a></li>
```




Рис. 3.16. Наложение сетки

```

<li><a href="#">Synopsis</a></li>
<li><a href="#">Stills/Photos</a></li>
<li><a href="#">Videos/clips</a></li>
<li><a href="#">Quotes</a></li>
<li><a href="#">Quiz</a></li>
</ul>
</div>
</div>
<!-- содержимое -->
<div id="content">
  
  <h1>Every year <span>when I watch the Oscars I'm annoyed...</ span></h1>
  <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the
  real cinematic heroes lose out. Not very Hollywood is it?</p>
  <p>We're here to put things right. </p>

```

```

<a href="#">these should have won &raquo;</a>
</div>
<!-- врезка -->
<div id="sidebar">
  <div class="sideBlock unsung">
    <h4>Unsung heroes...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
  <div class="sideBlock overHyped">
    <h4>Overhyped nonsense...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
</div>
<!-- нижний колонтитул -->
<div id="footer">
  <p>Note: our opinion is absolutely correct. You are wrong, even if you think you
are right. That's a fact. Deal with it.</p>
</div>
</div>
</body>
</html>

```

Первым делом нам необходимо указать, что наш элемент `<div>` с идентификатором `#wrapper` является контейнером для всех элементов, поэтому добавим для него класс `.container`:

```
<div id="wrapper" class="container">
```

Двигаясь далее вниз по разметке страницы, мы видим, что фраза **AND THE WINNER ISN'T...** занимает первую строку. Следовательно, для этого элемента мы добавим класс `.row`:

```
<div id="header" class="row">
```

Наш логотип, хотя это всего лишь текст, простирается на 12 колонок. Следовательно, для него мы добавим `.col_12`:

```
<div id="logo" class="col_12">And the winner is<span>n't...</span></div>
```

Следующую строку занимает элемент `<div>` с идентификатором `navigation`, так что мы добавим для него класс `.row`:

```
<div id="navigation" class="row">
```

Далее снова добавляются классы `.row` и `.col_x` там, где это необходимо. Мы пропустим этот этап, поскольку от повторения описанного процесса вас может начать клонить в сон. Вместо этого я приведу разметку уже со всеми внесенными изменениями.

Надо отметить, что пришлось также переместить изображение статуэтки «Оскар» и расположить его в отдельной колонке. Кроме того, нужно было заключить в раздел `<div>` с `class="row"` наши элементы `#content` и `#sidebar`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0" />
<title>And the winner isn't...</title>
<script type="text/javascript">document.cookie='resolution='+Math.
max(screen.width,screen.height)+'; path=/'</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
<link href="css/custom.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper" class="container">
  <!-- верхний колонтитул и навигация -->
  <div id="header" class="row">
    <div id="logo" class="col_12">And the winner is<span>n't...</span></div>
    <div id="navigation" class="row">
      <ul>
        <li><a href="#">Why?</a></li>
        <li><a href="#">Synopsis</a></li>
        <li><a href="#">Stills/Photos</a></li>
        <li><a href="#">Videos/clips</a></li>
        <li><a href="#">Quotes</a></li>
        <li><a href="#">Quiz</a></li>
      </ul>
    </div>
  </div>
  <div class="row">
    <!-- содержимое -->
    <div id="content" class="col_9 alpha omega">
      
      <div class="col_6 omega">
        <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
        <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst
the real cinematic heroes lose out. Not very Hollywood is it?</p>
        <p>We're here to put things right. </p>
        <a href="#">these should have won &raquo;</a>
      </div>
    </div>
    <!-- врезка -->
    <div id="sidebar" class="col_3">
      <div class="sideBlock unSung">
        <h4>Unsung heroes...</h4>
      </div>
    </div>
  </div>
</div>
</body>
</html>
```

```

    <a href="#"></a>
    <a href="#"></a>
  </div>
  <div class="sideBlock overHyped">
    <h4>Overhyped nonsense...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
</div>
</div>
<!-- нижний колонтитул -->
<div id="footer" class="row">
  <p>Note: our opinion is absolutely correct. You are wrong, even if you think you
are right. That's a fact. Deal with it.</p>
</div>
</div>
</body>
</html>

```

Кроме того, необходимо было добавить в файл `custom.css` дополнительные стили. Содержимое этого файла теперь выглядит следующим образом:

```

#navigation ul li {
  display: inline-block;
}

#content {
  float: right;
}

#sidebar {
  float: left;
}

.sideBlock {
  width: 100%;
}

.sideBlock img {
  max-width: 45%;
  float: left;
}

.footer {
  float: left;
}

```

Внеся все описанные ранее изменения и быстро взглянув на страницу в окне браузера, мы увидим, что наша базовая структура выглядит надлежащим образом и масштабируется пропорционально ширине области просмотра в браузере (рис. 3.17).

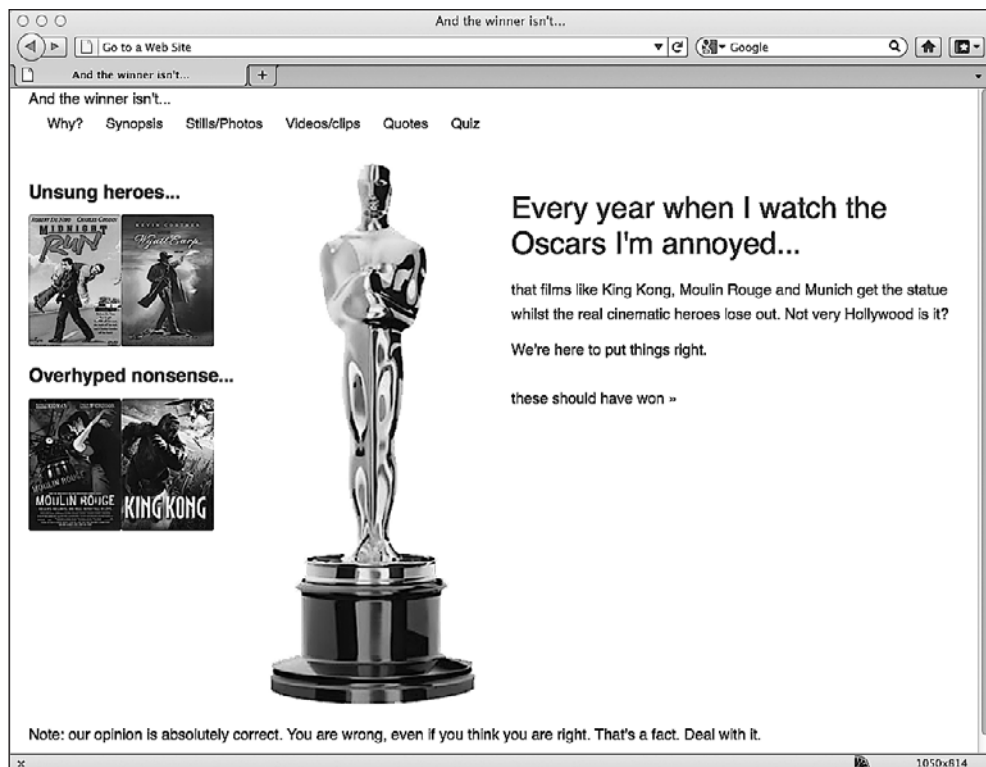


Рис. 3.17. Результат внесения всех изменений

Нам, конечно же, все еще предстоит проделать массу работы (я знаю, что это еще мягко сказано), однако если требуется быстрый способ создания базовой адаптивной структуры, то сеточные системы CSS вроде Columnal заслуживают внимания.

3.9. Резюме

В этой главе мы выяснили, как преобразовать жесткую структуру с размерами на основе пикселей в гибкую структуру, размеры которой основаны на процентных значениях. Мы также научились использовать единицы `em` вместо пикселей для задания более гибких шрифтов. Кроме того, мы разобрались, как сделать так, чтобы изображения реагировали на изменения и гибко масштабировались. Мы реализовали серверное решение для обеспечения пользователей абсолютно разными по величине изображениями в зависимости от размеров экранов применяемых ими устройств. И наконец, мы поэкспериментировали с адаптивной сеточной системой CSS, позволяющей быстро прототипировать адаптивные макеты с минимальной затратой сил.

Однако до сих пор мы исследовали особенности адаптивного веб-дизайна, используя для разметки HTML 4.01. В главе 1 было сказано об эффекте экономии от применения HTML5. Это особенно важно и актуально для адаптивных веб-дизайнов, где принцип «сперва проектируем для мобильной платформы» сводится к созданию настолько компактного, быстро работающего и семантически значимого кода, насколько это возможно.

В следующей главе мы применим HTML5 и модифицируем нашу разметку, чтобы воспользоваться преимуществами самой свежей и выдающейся версии спецификации HTML.

4 HTML5 для адаптивных веб-дизайнов

HTML5 развился из проекта Web Applications 1.0, инициатором которого выступила группа **Web Hypertext Application Technology Working Group (WHATWG)**, а позднее к работе над ним подключился W3C. Впоследствии большие части спецификации HTML5 были посвящены веб-приложениям. Если вы не занимаетесь созданием веб-приложений, то это не значит, что в HTML5 не найдется множества функций, которыми вы могли бы воспользоваться (на самом деле вы даже должны это сделать), приступая к созданию адаптивного веб-дизайна. Таким образом, в то время как одни HTML5-функции имеют непосредственное отношение к созданию хороших адаптивных веб-страниц (в основе которых, например, лежит более компактный код), другие находятся за пределами сферы адаптивного веб-дизайна.

HTML5 также предусматривает специальные инструменты для обработки форм и вводимых пользователями данных. Эти инструменты позволяют в значительной степени снизить применение таких требовательных к вычислительным ресурсам технологий, как JavaScript, для выполнения действий вроде валидации форм. Однако мы отдельно взглянем на формы в главе 8, а в этой главе поговорим о следующем:

- какие части HTML5 можно использовать уже сегодня;
- как писать HTML5-страницы;
- каков эффект экономии от применения HTML5;
- какие HTML-параметры стали устаревшими;
- какие новые семантические HTML5-элементы появились;
- как применять Web Accessibility Initiative-Accessible Rich Internet Applications (WAI-ARIA) ради расширенной семантики и содействия пользователям вспомогательных технологий;
- что такое вложение мультимедиа;
- как применять адаптивное HTML5-видео, а также видео в `<iframe>`;
- как сделать сайт доступным в автономном режиме.

4.1. Какие части HTML5 можно использовать уже сегодня?

Хотя полная спецификация HTML5 пока не утверждена, большинство новых HTML5-функций в той или иной степени уже поддерживаются современными браузерами, включая Apple Safari, Google Chrome, Opera, Mozilla Firefox и даже Internet Explorer 9! Конечно, маловероятен тот факт, что все в текущем черновике HTML5-спецификации доживет до достижения ею статуса W3C Recommendation (REC), однако в ней имеется масса новых функций, которые можно использовать уже сегодня.

Большинство сайтов может быть написано на HTML5

В настоящее время, если меня попросят создать сайт, я по умолчанию буду использовать для написания разметки HTML5, а не HTML 4.01. Всего несколько лет назад ситуация была противоположной, однако сейчас нет убедительных причин для того, чтобы не писать разметку сайтов на HTML5. Все современные браузеры без проблем «понимают» общие HTML5-параметры (новые структурные элементы, теги `<video>` и `<audio>`), а устаревшие версии Internet Explorer допустимо снабдить **полизаполнениями**, чтобы устранить возможные недостатки.



ЧТО ТАКОЕ ПОЛИЗАПОЛНЕНИЯ?

Термин «полизаполнение» (polyfill) был придуман Реми Шарпом (Remy Sharp) как намек на заполнение брешей в устаревших браузерах с использованием Polyfilla (название шпатлевки в США). Таким образом, полизаполнение — это JavaScript-прослойка, которая эффективно имитирует новые функции в устаревших браузерах. Однако важно учитывать, что полизаполнения «утяжеляют» код. Таким образом, простая возможность добавить три сценария-полизаполнения, чтобы Internet Explorer 6 обрабатывал ваш сайт так же, как и любой другой браузер, вовсе не означает, что вы непременно должны прибегать к ней!

Полизаполнения, прослойки и Modernizr

Как правило, устаревшие версии Internet Explorer (до версии 9) «не понимают» новые семантические HTML5-элементы. Однако не так давно Шерд Висшер (Sjoerd Visscher) выяснил, что если создавать элементы, изначально используя JavaScript, то Internet Explorer будет способен распознать и стилизовать их соответствующим образом. Вооружившись этими знаниями, специалист по JavaScript Реми Шарп создал небольшой специальный сценарий (<http://remysharp.com/2009/01/07/html5-enabling-script/>), который при включении в HTML5-страницу волшебным образом обеспечивает поддержку HTML5-элементов в устаревших версиях Internet Explorer.

Наверняка HTML5-первопроходцы в течение длительного времени применяли бы в своей разметке этот сценарий, чтобы обеспечить для пользователей

Internet Explorer версий 6, 7 и 8 взаимодействие, сопоставимое с тем, что имеет место в современных браузерах. Однако прогресс шагнул далеко вперед. Сейчас у нас есть новичок, который делает все описанное чуть выше, а также многое другое. Он называется Modernizr (<http://www.modernizr.com>) и заслуживает вашего внимания, если вы создаете страницы на HTML5. Помимо обеспечения поддержки структурных HTML5-элементов в устаревших версиях Internet Explorer, он позволяет условно загружать дополнительные полизаполнения, CSS- и JavaScript-файлы, беря за основу результаты тестов браузеров на предмет поддерживаемых ими функций.

Таким образом, поскольку веских доводов в пользу того, чтобы не использовать HTML5, мало, приступим к написанию разметки на HTML5.



НУЖЕН КРАТЧАЙШИЙ ПУТЬ К ОТЛИЧНОМУ HTML5-КОДУ? РАССМОТРИТЕ ВОЗМОЖНОСТЬ ИСПОЛЬЗОВАНИЯ HTML5 BOILERPLATE

Если время не терпит и вам необходима хорошая отправная точка для вашего проекта, то рассмотрите возможность использования HTML5 Boilerplate (<http://html5boilerplate.com/>). Это готовый HTML5-файл, включающий важные стили (как, например, упоминавшийся ранее `normalize.css`), полизаполнения и инструменты вроде Modernizr. Кроме того, в него входит компоновочный инструмент, который автоматически соединяет CSS- и JavaScript-файлы, а также удаляет комментарии для создания производственного кода. Очень рекомендую!

4.2. Как писать HTML5-страницы

Откройте уже имеющуюся у вас веб-страницу. Вероятно, первые несколько строк ее разметки будут выглядеть следующим образом:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Удалите этот фрагмент кода и замените его следующим:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
```

Сохраните изменения в документе, и в результате у вас должна получиться HTML5-страница, которая сможет пройти валидацию в валидаторе W3C (<http://validator.w3.org/>).

Не беспокойтесь, на этом текущая глава не заканчивается! Это предварительное упражнение призвано лишь продемонстрировать гибкость HTML5. Это эволюционное, а не революционное изменение разметки, которую вы уже пишете. Мы можем использовать его для того, чтобы «зарядить» разметку, которую уже умеем писать.

Итак, что же мы в действительности сделали чуть раньше? Прежде всего мы указали новое объявление типа документа HTML5:

```
<!DOCTYPE html>
```

Если вам нравится писать код в нижнем регистре, то `<!doctype html>` будет ничуть не хуже. Разницы никакой.



ОБЪЯВЛЕНИЕ ТИПА ДОКУМЕНТА HTML5 — ПОЧЕМУ ОНО ТАКОЕ КОРОТКОЕ?

Объявление типа документа HTML5 `<!DOCTYPE html>` такое короткое потому, что лишь указывает браузерам обрабатывать страницы в «стандартном режиме». Такой наиболее эффективный подход к написанию синтаксиса преобладает в значительной части HTML5.

После объявления типа документа мы открыли тег `<html>`, указали язык, а затем открыли раздел `<head>`:

```
<html lang="en">
<head>
```



ВЫ ГОВОРИТЕ ПО-НЕМЕЦКИ?

Согласно спецификациям W3C (<http://dev.w3.org/html5/spec/Overview.html#attr-lang>) атрибут `lang` определяет первичный язык для содержимого конкретного элемента и любых атрибутов этого элемента, которые содержат текст. Если вы пишете страницы не на английском, то вам потребуется указать соответствующий язык. Например, для японского языка тег `<html>` будет выглядеть как `<html lang="ja">`. Полный перечень языков можно увидеть по адресу <http://www.iana.org/assignments/languagesubtag-registry>.

И наконец, мы указали кодировку символов. Для этого мы использовали элемент без содержимого, который не требует закрывающего тега:

```
<meta charset=utf-8>
```

Если только у вас не будет веской причины указать другую кодировку символов, то ею почти всегда будет UTF-8¹.

Эффект экономии от использования HTML5

У меня в школе был весьма придирчивый (но все же очень хороший) учитель математики. Случалось, что он не приходил на занятия, и тогда весь класс облегченно вздыхал, поскольку те, кто подменял «Мистера Придирчивость», обычно оказывались покладистыми и добродушными людьми, которые ладили с нами без криков и постоянной нервотрепки. Они не настаивали на тишине, пока мы занимались делом, и не очень беспокоились насчет того, насколько аккуратно выглядели наши решения задач в тетрадях. Для них были важны лишь ответы. Если бы

¹ Для русскоязычных сайтов подходящей будет кодировка windows-1251. — *Примеч. ред.*

HTML5 был учителем математики, то его можно было представить вот таким добродушным временным заместителем постоянного учителя. Я поясню эту причудливую аналогию...

Если вы обратите внимание на то, как пишете код, то заметите, что в основном используете нижний регистр, заключаете значения атрибутов в кавычки и объявляете `type` для сценариев и таблиц стилей. Например, вы могли бы указать ссылку на таблицу стилей так:

```
<link href="CSS/main.css" rel="stylesheet" type="text/css" />
```

HTML5 не требует такой детальности и позволяет указать все это следующим образом:

```
<link href=CSS/main.css rel=stylesheet >
```

Знаю, знаю. Мне это тоже кажется странным. Нет закрывающего тега/слеша, кавычек вокруг значений атрибутов и объявления `type`. Однако добродушному HTML5 все равно. Второй пример будет таким же валидным, как и первый.

Подобный нестрогий синтаксис допускается во всем документе, а не только в отношении привязываемых CSS-файлов и JavaScript-элементов. Например, если хотите, можете указать `<div>` следующим образом:

```
<div id=wrapper>
```

Это абсолютно валидный HTML5-код. Аналогичным образом дело обстоит и со вставкой изображений:

```
<img SRC=frontCarousel.png alt=frontCarousel>
```

Это тоже валидный HTML5-код. Никакого закрывающего тега/слеша, кавычек и комбинации символов в верхнем и нижнем регистрах. Вы даже можете пренебречь открывающим тегом `<head>`, и ваша страница все равно сможет пройти валидацию. Что обо всем этом «сказал» бы XHTML 1.0!

Разумный подход к написанию HTML5-разметки

Несмотря на то что при работе с нашими адаптивными веб-страницами и дизайнами мы стремимся придерживаться принципа «сперва проектируем для мобильной платформы», я не могу перестать думать о создании наиболее оптимальной разметки (здесь я руководствовался стандартами разметки XHTML 1.0, которые подразумевают XML-синтаксис). Да, мы можем сделать наши страницы чуть меньше по объему благодаря эффекту экономии, который достигается при использовании HTML5, однако, откровенно говоря, если потребуется, то я лучше уберу из дизайна то или иное изображение, чтобы добиться экономии!

Мне кажется, что ради удобочитаемости кода все же стоит печатать дополнительные символы (слеши в конце и кавычки вокруг значений атрибутов). Поэтому при написании HTML5-документов я стремлюсь выбирать нечто среднее между старомодным стилем создания разметки (который обеспечивает валидную разметку с точки зрения HTML5, хотя валидаторы/инструменты для проверки соответствия

могут выдавать предупреждающие сообщения) и эффектом экономии, который дает HTML5. Например, если говорить о ссылке на таблицу стилей CSS, которая приводилась чуть ранее, то для нее я бы написал следующий код:

```
<link href="CSS/main.css" rel="stylesheet"/>
```

Я сохранил закрывающий тег и кавычки, но исключил атрибут `type`. Здесь важно подчеркнуть, что вы можете сами определить, какой именно уровень вас устраивает. HTML5 не станет кричать на вас, демонстрировать вашу разметку перед всем классом и ставить вас в угол за то, что она не проходит валидацию.

Приветствуем великий тег `<a>`

Еще одна особенность HTML5, которая обеспечивает реальную экономию, состоит в том, что теперь мы наконец-то можем заключать в тег `<a>` множественные элементы. Раньше, если вы хотели, чтобы ваша разметка смогла пройти валидацию, приходилось заключать каждый элемент в его собственный тег `<a>`. Например, взгляните на следующий фрагмент кода:

```
<h2><a href="index.html">The home page</a></h2>
<p><a href="index.html">This paragraph also links to the home page</a></p>
<a href="index.html"></a>
```

Однако мы можем убрать все индивидуальные теги `<a>` и взамен расположить группу элементов так, как показано в приведенном далее фрагменте кода:

```
<a href="index.html">
<h2>The home page</h2>
<p>This paragraph also links to the home page</p>

</a>
```

Однако следует иметь в виду, что, как вполне понятно, нельзя заключать один тег `<a>` в другой тег `<a>`, а также нельзя заключать `<form>` в тег `<a>`.

Устаревшие HTML-функции

Существуют атрибуты HTML, к использованию которых вы, возможно, привыкли, но теперь они считаются устаревшими в HTML5. Важно осознавать, что есть два «лагеря» устаревших параметров в HTML5: соответствующие и не соответствующие требованиям. Соответствующие требованиям элементы по-прежнему смогут работать, однако валидаторы будут генерировать предупреждающие сообщения. Вам следует избегать их применения на практике, если это возможно, однако небеса не рухнут, если вы все же добавите их в разметку. В определенных браузерах параметры, не соответствующие требованиям, будут обработаны, однако их использование считается весьма скверной практикой и может иметь отрицательные последствия!

Примером устаревшего, но соответствующего требованиям параметра является атрибут `border`, который указывается в теге ``. Исторически он применялся для

того, чтобы вокруг изображений не выводилась рамка синего цвета, если они вложены в ссылку. Например, взгляните на следующий код:

```

```

Вместо этого для обеспечения аналогичного эффекта я рекомендую вам использовать CSS.

Устаревших и не соответствующих требованиям параметров довольно много. Признаюсь, мне не доводилось использовать многие из них (а с некоторыми я даже ни разу не сталкивался!). Возможно, в вашем случае все сложится похожим образом. Однако если вам интересно, то вы сможете найти полный список устаревших и не соответствующих требованиям элементов и атрибутов по адресу <http://dev.w3.org/html5/spec/Overview.html#non-conforming-features>. К числу примечательных не соответствующих требованиям элементов относятся `strike`, `center`, `font`, `acronym`, `frame` и `frameset`.

4.3. Новые семантические HTML5-элементы

В имеющемся у меня словаре семантике дается следующее определение: *«раздел языкознания, изучающий содержание, информацию, передаваемые языковыми единицами»*. В нашем случае семантика — это процесс придания значения нашей разметке. Почему это важно? Я рад, что вы спросили. Взгляните на структуру текущей разметки сайта And the winner isn't...:

```
<body>
<div id="wrapper">
  <div id="header">
    <div id="logo"></div>
    <div id="navigation">
      <ul>
        <li><a href="#">Why?</a></li>
      </ul>
    </div>
  </div>
  <!-- содержимое -->
  <div id="content">
    </div>
    <!-- врезка -->
    <div id="sidebar">
    </div>
    <!-- нижний колонтитул -->
    <div id="footer">
    </div>
  </div>
</body>
```

Большинство специалистов, занимающихся написанием разметки, заметят общие соглашения по идентификационным именам используемых элементов `<div>` — `header`, `content`, `sidebar` и т. д. Однако, взглянув на сам код, любой

пользовательский агент (браузер, экранный диктор, поисковый робот и др.) не сможет наверняка сказать, в чем заключается назначение каждого раздела `<div>`. HTML5 стремится решить эту проблему с помощью новых семантических элементов.

Элемент `<section>`

Элемент `<section>` используется для определения универсального раздела документа или приложения. Например, вы можете захотеть разбить на части имеющееся у вас содержимое; один раздел будет служить для контактных данных, другой — для новостных лент и т. д. Важно понимать, что элемент `<section>` не предназначен для использования в целях стилизации. Если вам потребуется заключить тот или иной элемент во что-то просто для стилизации, то продолжайте использовать `<div>`, как делали это раньше.



СОВЕТ

По адресу <http://dev.w3.org/html5/spec/Overview.html#thesection-element> вы сможете узнать, что говорится в спецификации HTML5 консорциума W3C об элементе `<section>`.

Элемент `<nav>`

Элемент `<nav>` используется для определения главных навигационных блоков — ссылок на другие страницы или части в рамках конкретной страницы. Этот элемент не предназначен для использования лишь в нижних колонтитулах (хотя может) и т. п., где группы ссылок на другие страницы являются обычным делом.



СОВЕТ

По адресу <http://dev.w3.org/html5/spec/Overview.html#the-navelement> вы сможете узнать, что говорится в спецификации HTML5 консорциума W3C об элементе `<nav>`.

Элемент `<article>`

Элемент `<article>` вместе с `<section>` могут легко привести к путанице. Мне, в частности, пришлось прочитать, а затем перечитать спецификации, касающиеся каждого из этих элементов, прежде чем до меня все дошло. Элемент `<article>` используется для обособленной части содержимого. При структурировании страницы задайтесь вопросом: можно ли будет взять все содержимое целиком, которое вы намереваетесь поместить в тег `<article>`, и вставить его в другой сайт, чтобы при этом оно осталось абсолютно понятным? Здесь можно рассуждать и по-другому: будет ли содержимое, расположенное в `<article>`, в действительности представлять собой отдельную статью в RSS-ленте? Наглядным примером содержимого, которое следует заключать в элемент `<article>`, является блог-пост. Знайте, что при вложении элементов `<article>` предполагается, что такие вложенные элементы будут связаны с внешним `<article>`.

**СОВЕТ**

По адресу <http://dev.w3.org/html5/spec/Overview.html#thearticle-element> вы сможете узнать, что говорится в спецификации HTML5 консорциума W3C об элементе `<article>`.

Элемент `<aside>`

Элемент `<aside>` используется для второстепенного по значению контента. На практике я часто применяю его для добавления врезок (когда они включают соответствующее содержимое). Он также считается подходящим для размещения цитат, рекламы и групп навигационных элементов (вроде списков ссылок на другие блогги и т. д.).

**СОВЕТ**

Что еще говорится в спецификации HTML5 консорциума W3C об элементе `<aside>`, вы сможете узнать по адресу <http://dev.w3.org/html5/spec/Overview.html#theaside-element>.

Элемент `<hgroup>`

Если у вас имеется несколько заголовков, слоганов и подзаголовков в `<h1>`, `<h2>`, `<h3>` и последующих тегах, то рассмотрите возможность заключить их в тег `<hgroup>`. Сделав это, вы скроете вторичные элементы от алгоритма определения иерархической структуры HTML5, в результате чего только первый заголовочный элемент в `<hgroup>` будет «участвовать» в иерархической структуре документа.

Алгоритм определения иерархической структуры HTML5. HTML5 позволяет каждому контейнеру иметь свою отдельную иерархическую структуру. Это означает, что больше не нужно постоянно думать о том, на каком уровне тега `<header>` вы находитесь. Например, в блоге я могу сделать так, чтобы для размещения названий моих постов использовался тег `<h1>`, как и для размещения названия самого блога. В качестве примера взгляните на следующую структуру:

```
<hgroup>
  <h1>Ben's blog</h1>
  <h2>All about what I do</h2>
</hgroup>
<article>
  <header>
    <hgroup>
      <h1>A post about something</h1>
      <h2>Trust me this is a great read</h2>
      <h3>No, not really</h3>
      <p>See. Told you.</p>
    </hgroup>
  </header>
</article>
```

Несмотря на наличие нескольких заголовков `<h1>` и `<h2>`, иерархическая структура по-прежнему будет выглядеть так:

- Ben's blog
 - A post about something

Вам не нужно отслеживать тег `<header>`, который требуется использовать. Вы можете просто выбрать любой уровень тега `<header>` в каждом из разделов, и алгоритм определения иерархической структуры HTML5 выполнит соответствующее упорядочение.

Можете проверить иерархическую структуру своих документов с помощью специальных инструментов, располагающихся по следующим адресам:

- <http://gsnedders.html5.org/outliner/>;
- <http://hoyois.github.com/html5outliner/>.

На рис. 4.1 показана страница HTML5 Outliner.

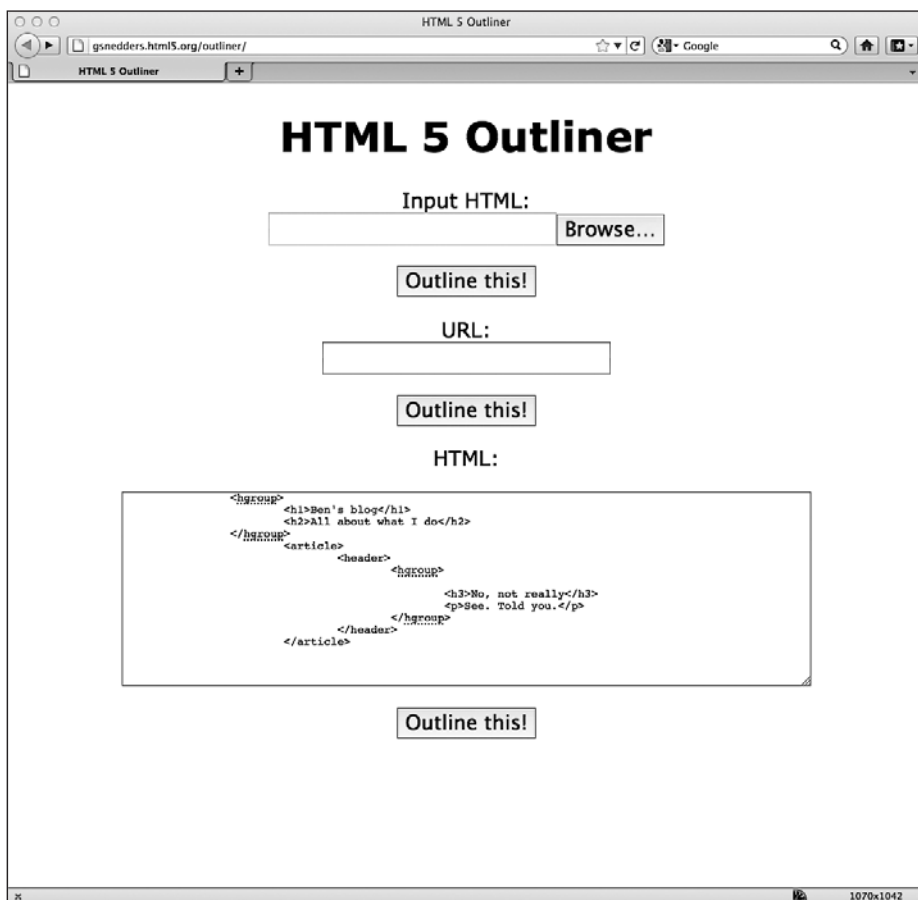


Рис. 4.1. Страница HTML5 Outliner

**СОВЕТ**

Что еще говорится в спецификации HTML5 консорциума W3C об элементе `<hgroup>`, вы сможете узнать по адресу <http://dev.w3.org/html5/spec/Overview.html#thehgroup-element>.

Элемент `<header>`

Элемент `<header>` не задействуется при выполнении алгоритма определения иерархической структуры, поэтому не может использоваться для секционирования содержимого. Вместо этого его следует применять для введения в содержимое. На практике элемент `<header>` может использоваться в «шапке» верхнего колонтитула сайта, а также для введения в другое содержимое, например то, что располагается в элементе `<article>`.

**СОВЕТ**

По адресу <http://dev.w3.org/html5/spec/Overview.html#theheader-element> вы сможете узнать, что говорится в спецификации HTML5 консорциума W3C об элементе `<header>`.

Элемент `<footer>`

Как и `<header>`, элемент `<footer>` не задействуется при выполнении алгоритма определения иерархической структуры, поэтому не может использоваться для секционирования содержимого. Вместо этого его следует применять для размещения информации о разделе, в котором он располагается. Элемент `<footer>` может содержать, например, ссылки на другие документы или информацию об авторских правах. Как и `<header>`, он может неоднократно задействоваться в рамках страницы, если потребуется. Скажем, его можно использовать для размещения нижнего колонтитула блога, а также нижнего колонтитула в `<article>` блог-поста. Однако в спецификации HTML5 отмечается, что контактные данные автора того или иного блог-поста следует заключать в элемент `<address>`.

**СОВЕТ**

По адресу <http://dev.w3.org/html5/spec/Overview.html#thefooter-element> вы сможете узнать, что говорится в спецификации HTML5 консорциума W3C об элементе `<footer>`.

Элемент `<address>`

Элемент `<address>` предназначен конкретно для размещения контактных данных для своего ближайшего предшествующего элемента `<article>` или `<body>`. Чтобы не запутаться, помните, что элемент `<address>` **не используется** для размещения почтовых адресов и т. п., если только они в действительности не являются контактными

адресами, которые относятся к соответствующему содержанию. Вместо этого почтовые адреса и прочие произвольные контактные данные следует заключать в старые добрые теги `<p>`.

**СОВЕТ**

По адресу <http://dev.w3.org/html5/spec/Overview.html#theaddress-element> вы сможете узнать, что говорится в спецификации HTML5 консорциума W3C об элементе `<address>`.

4.4. Практическое использование структурных элементов HTML5

Взглянем на примеры использования этих новых элементов. Думаю, элементы `<header>`, `<nav>` и `<footer>` говорят сами за себя, поэтому для начала возьмем текущую разметку домашней страницы сайта And the winner isn't... и изменим области `<header>`, `<nav>` и `<footer>` (см. выделенные области в приведенном далее фрагменте кода):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initial-scale=1.0"
/>
<title>And the winner isn't...</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen. height)+';
path='/';</script>
<link href="css/main.css" rel="stylesheet" />
</head>
<body>
<div id="wrapper">
  <!-- верхний колонтитул и навигация -->
  <header>
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav>
      <ul>
        <li><a href="#">Why?</a></li>
        <li><a href="#">Synopsis</a></li>
        <li><a href="#">Stills/Photos</a></li>
        <li><a href="#">Videos/clips</a></li>
        <li><a href="#">Quotes</a></li>
        <li><a href="#">Quiz</a></li>
      </ul>
    </nav>
  </header>
  <!-- содержимое -->
```

```

<div id="content">
  
  <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
  <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the
real cinematic heroes lose out. Not very Hollywood is it?</p>
<p>We're here to put things right. </p>
  <a href="#">these should have won &raquo;</a>
</div>
<!-- врезка -->
<div id="sidebar">
  <div class="sideBlock unsung">
    <h4>Unsung heroes...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
  <div class="sideBlock overHyped">
    <h4>Overhyped nonsense...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
</div>
<!-- нижний колонтитул -->
<footer>
  <p>Note: our opinion is absolutely correct. You are wrong, even if you think you
are right. That's a fact. Deal with it.</p>
</footer>
</div>
</body>
</html>

```

Однако, как вы уже видели ранее, когда в рамках страницы используются `<article>` и `<section>`, их может быть больше чем по одному на страницу. Каждый элемент `<article>` или `<section>` может включать свой собственный `<header>`, `<footer>` и `<nav>`. Например, если мы добавим в разметку элемент `<article>`, то код будет выглядеть следующим образом:

```

<body>
<div id="wrapper">
  <!-- верхний колонтитул и навигация -->
  <header>
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav>
      <ul>
        <li><a href="#">Why?</a></li>
      </ul>
    </nav>
  </header>
  <!-- содержимое -->

```

```

<div id="content">
  <article>
    <header>An article about HTML5</header>
    <nav>
      <a href="1.html">related link 1</a>
      <a href="2.html">related link 2</a>
    </nav>
    <p>это содержимое статьи</p>
    <footer>This was an article by Ben Frain</footer>
  </article>

```

Как видно, мы используем `<header>`, `<nav>` и `<footer>` как для страницы, так и для статьи, которая на ней располагается.

Изменим нашу область `<sidebar>`. Вот что у нас имеется на текущий момент в разметке на HTML 4.01:

```

<!-- врезка -->
<div id="sidebar">
  <div class="sideBlock unsung">
    <h4>Unsung heroes...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
  <div class="sideBlock overHyped">
    <h4>Overhyped nonsense...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
</div>

```

Содержимое нашей врезки, несомненно, является второстепенным по отношению к основному содержимому, поэтому первым делом удалим `<div id="sidebar">` и заменим его элементом `<aside>`:

```

<!-- врезка -->
<aside>
  <div class="sideBlock unsung">
    <h4>Unsung heroes...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
  <div class="sideBlock overHyped">
    <h4>Overhyped nonsense...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
</aside>

```

Отлично! Однако взглянем на страницу в браузере (рис. 4.2). Неудивительно, если после этого у вас вырвется бранное словечко...

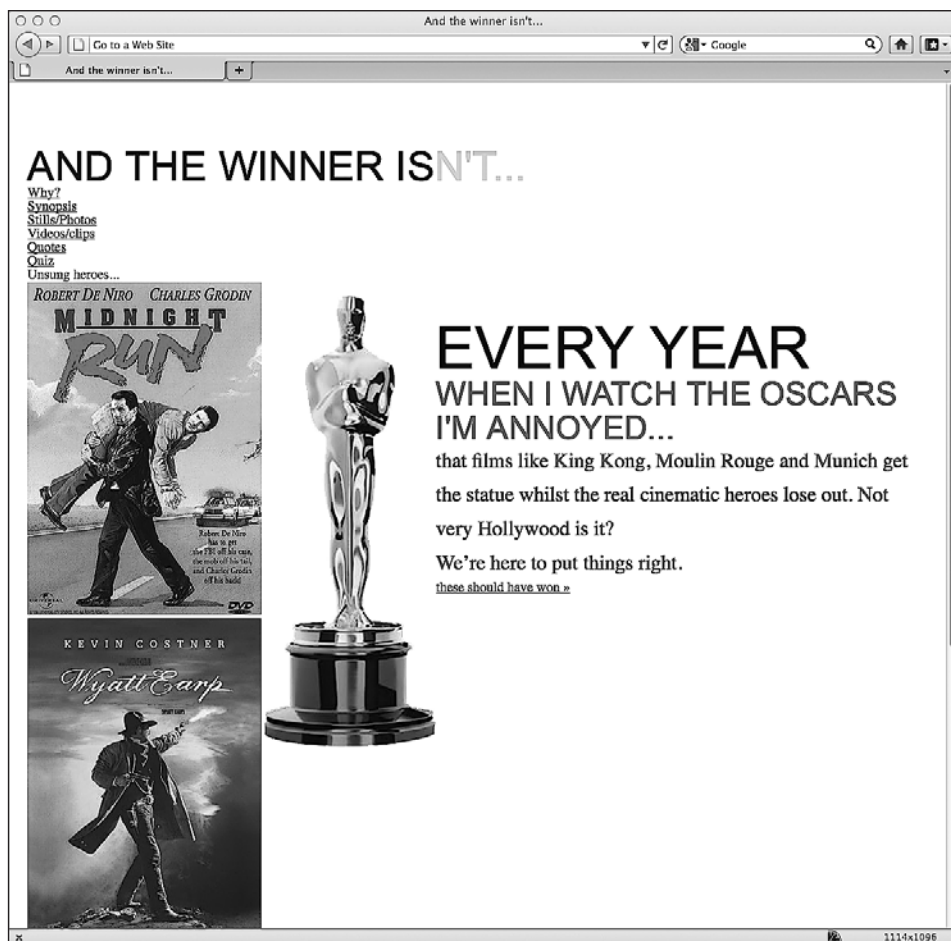


Рис. 4.2. Страница с внесенными изменениями

Получается, мы сделали один шаг вперед и два назад! Причина этого заключается в том, что мы не внесли изменения в CSS-код в соответствии с новыми элементами. Сделаем это перед тем, как двигаться дальше. Необходимо изменить все ссылки на `#header`, написав их как `header`, все ссылки на `#navigation`, написав их как `nav`, и все ссылки на `#footer`, написав их как `footer`. Например, первое CSS-правило, относящееся к верхнему колонтитулу, изменится с такого:

```
#header {
  background-position: 0 top;
  background-repeat: repeat-x;
  background-image: url(../img/buntingSlice3Invert.png);
  margin-right: 1.0416667%; /* 10×960 */
  margin-left: 1.0416667%; /* 10×960 */
  width: 97.9166667%; /* 940×960 */
}
```

на такое:

```
header {
  background-position: 0 top;
  background-repeat: repeat-x;
  background-image: url(../img/buntingSlice3Invert.png);
  margin-right: 1.0416667%; /* 10 × 960 */
  margin-left: 1.0416667%; /* 10 × 960 */
  width: 97.9166667%; /* 940 × 960 */
}
```

Это было очень легко сделать для верхнего и нижнего колонтитулов, а также навигации, поскольку идентификаторы были такими же, как элементы, на которые мы их заменяли, — мы просто убрали символ #. С врезкой дело обстоит немного по-другому: нам необходимо изменить ссылки с #sidebar на aside. Однако здесь поможет функция поиска и замены в вашем любимом редакторе кода. Чтобы было ясно, правила вроде такого:

```
#sidebar {
}
```

изменяться следующим образом:

```
aside {
}
```

Даже если вы написали очень большую таблицу стилей CSS, замена всех идентификаторов HTML 4.01 HTML5-элементами будет довольно безболезненной процедурой.



БУДЬТЕ БДИТЕЛЬНЫ В ОТНОШЕНИИ МНОЖЕСТВЕННЫХ ЭЛЕМЕНТОВ В HTML5

Знайте, что в HTML5 в рамках одной страницы разрешено использовать множественные элементы <header>, <footer> и <aside>, поэтому может потребоваться написать более специфические стили для отдельных экземпляров.

После изменения соответствующим образом всех стилей для сайта And the winner isn't... наша страница приобретет надлежащий вид (рис. 4.3).

Теперь, хотя мы и сообщаем пользователям, какой раздел страницы является врезкой (элемент <aside>), у нас имеется два разных раздела: **UNSUNG HEROES** и **OVERHYPED NONSENSE**. Следовательно, для семантического определения этих областей потребуется внести дополнительные изменения в наш код:

```
<!-- врезка -->
<aside>
  <section>
    <div class="sideBlock unsung">
      <h4>Unsung heroes...</h4>
      <a href="#"></a>
      <a href="#"></a>
```

```

</div>
</section>
<section>
  <div class="sideBlock overHyped">
    <h4>Overhyped nonsense...</h4>
    <a href="#"></a>
    <a href="#"></a>
  </div>
</section>
</aside>

```



Рис. 4.3. Страница выглядит так, как задумывалось

Важно помнить, что элемент `<section>` не предназначен для стилизации, а служит для идентификации определенного фрагмента содержимого. Обычно у разделов также имеются заголовки, что нам очень подходит. Из-за алгоритма определения иерархической структуры HTML5 мы также можем заменить теги `<h4>` тегами `<h1>`, и он все равно будет выдавать точную структуру нашего документа (рис. 4.4).

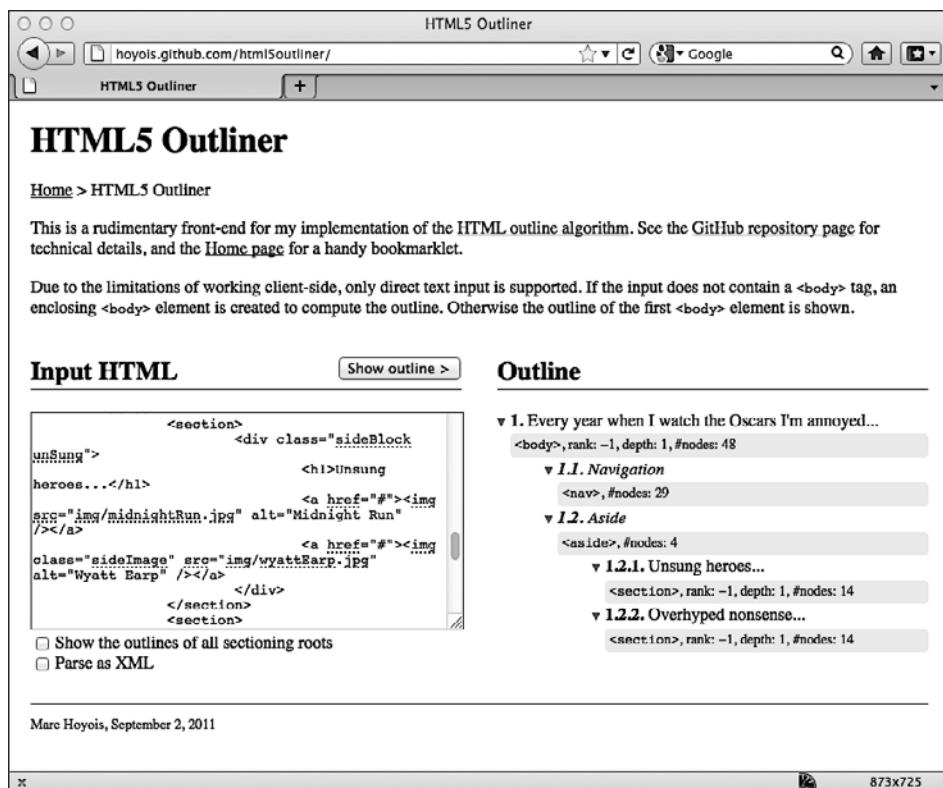


Рис. 4.4. Отображение структуры документа инструментом HTML5 Outliner

Как насчет основного содержимого сайта? Вас может удивить то, что нет определенного элемента для разметки основного содержимого страниц. Однако логика подсказывает, что, если есть способ отделить все второстепенное, то оставшееся должно быть основным содержимым страницы.

4.5. HTML5-семантика уровня текста

Помимо рассмотренных нами структурных элементов, в HTML5 пересмотрены элементы, которые раньше назывались **строчными**. Теперь в HTML5-спецификации они называются **семантическими элементами уровня текста** (<http://dev.w3.org/html5/spec/Overview.html#text-level-semantics>). Рассмотрим несколько распространенных примеров.

Элемент ****

Несмотря на то что мы часто использовали элемент **** для стилизации, в действительности его применение означало «выделить данный текст полужирным шрифтом». Теперь его можно официально использовать просто как средство сти-

лизации в CSS, поскольку в HTML5-спецификации он описывается следующим образом:

«...включает текст, не несущий дополнительного значения и не подразумевающий другую интонацию, например ключевые слова в резюме документа, названия товаров в обзоре или вступительная часть статьи».

Элемент ``

Ладно, признаю, я тоже часто использовал `` лишь как средство стилизации. Мне нужно исправиться, поскольку в HTML5 говорится, что этот элемент предназначен для того, чтобы:

«...подчеркнуть свое содержимое».

Следовательно, если только вы на самом деле не хотите подчеркнуть содержимое этого элемента, то используйте тег `` или, в соответствующих случаях, `<i>`.

Элемент `<i>`

В HTML5-спецификации элемент `<i>` описывается так:

«...включает текст, произносимый другим тоном или на другой лад либо с иным отступлением от обычного текста в манере, указывающей на его особое качество».

Достаточно сказать, что элемент `<i>` служит не только для того, чтобы выделить курсивом какой-нибудь текст.

Применение семантики уровня текста в нашей разметке

Взглянем на текущую разметку, касающуюся области основного содержимого нашей домашней страницы, и посмотрим, можно ли сделать ее более семантически значимой для пользовательских агентов. Вот что у нас имеется на данный момент:

```
<!-- содержимое -->
<div id="content">
  
  <h1>Every year <span>when I watch the Oscars I'm annoyed...</span></h1>
  <p>that films like King Kong, Moulin Rouge and Munich get the statue whilst the
  real cinematic heroes lose out. Not very Hollywood is it?</p>
  <p>We're here to put things right. </p>
  <a href="#">these should have won &raquo;</a>
</div>
```

Мы определенно можем здесь кое-что улучшить. Прежде всего тег `` в нашем заголовке `<h1>` лишен смысла в данном контексте. Поскольку мы пытаемся подчеркнуть соответствующий текст, стилизовав его, сделаем это в нашем коде:

```
<h1>Every year <em>when I watch the Oscars I'm annoyed...</em></h1>
```

Снова взглянем на нашу начальную композицию (рис. 4.5).



Рис. 4.5. Страница со стилизованным заголовком

Нам также необходимо по-другому выделить названия фильмов, однако они не должны подразумевать другую интонацию. Похоже, для этого идеально подойдет тег ``:

```
<p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and  
<b>Munich</b> get the statue whilst the real cinematic heroes lose out. Not very  
Hollywood is it?</p>
```



СТИЛИЗАЦИЯ, ПО УМОЛЧАНИЮ ПРИМЕНЯЕМАЯ К СЕМАНТИЧЕСКИМ ЭЛЕМЕНТАМ УРОВНЯ ТЕКСТА

При использовании элемента `` большинство браузеров по-прежнему устанавливают полужирное начертание для заключенного в него текста, поэтому, в зависимости от вашей ситуации, вам может потребоваться модифицировать определяемый по умолчанию стиль в соответствующем CSS-коде.

И наконец, я не шучу, когда говорю: «We're here to put things right» («Мы здесь для того, чтобы исправить текущее положение дел»), я не паясничаю, а хочу, чтобы пользовательские агенты знали это! Поэтому наконец заключим эту фразу в тег `<i>`. Вы можете заявить, что вместо него здесь следует использовать тег ``. В данном случае этот вариант тоже отлично подошел бы, однако я собираюсь использовать `<i>`. Так-то вот! В результате код будет выглядеть следующим образом:

```
<p><i>We're here to put things right.</i></p>
```

Подобно тому, как это происходит с текстом в теге ``, браузеры по умолчанию выделяют курсивом текст в теге `<i>`, поэтому вносите соответствующие исправления там, где это будет необходимо.

Итак, мы добавили семантику уровня текста в имеющееся у нас содержимое, чтобы сделать разметку семантически более значимой. В HTML5 есть множество других семантических тегов уровня текста. Их полный перечень можно найти, заглянув в соответствующий раздел HTML5-спецификации по адресу: <http://dev.w3.org/html5/spec/Overview.html#text-level-semantic>.

Однако, приложив еще немного усилий, мы можем пойти еще дальше и обеспечить дополнительное значение для пользователей, нуждающихся в специальных технологиях.

4.6. Обеспечение большей доступности для вашего сайта с помощью WAI-ARIA

Цель WAI-ARIA — сделать динамическое содержимое страниц доступным. WAI-ARIA предусматривает способы описания ролей, состояний и свойств для пользовательских виджетов (динамических частей веб-приложений), чтобы их смогли распознать и задействовать пользователи вспомогательных технологий.

Например, если экранный виджет отображает постоянно изменяющийся курс акций, то как незрячий пользователь, зашедший на соответствующую страницу, сможет узнавать об этих изменениях? WAI-ARIA пытается решить эту проблему. Полная реализация ARIA лежит вне рамок этой книги (всю информацию вы сможете найти по адресу <http://www.w3.org/WAI/intro/aria>). Однако существуют очень легко реализуемые части ARIA, которые мы можем применять для улучшения доступности любого написанного на HTML5 сайта для пользователей вспомогательных технологий.

Когда вам требуется создать сайт для заказчика, часто бывает так, что времени/денег на внедрение поддержки доступности не предусматривается (печально, но зачастую об этом люди вообще не задумываются). Однако можно прибегнуть к **ролям ориентиров ARIA**, чтобы устранить некоторые бросающиеся в глаза недостатки семантики HTML и позволить экранным дикторам, поддерживающим WAI-ARIA, легко переключаться между разными областями экрана.

Роли ориентиров ARIA. Реализация ролей ориентиров ARIA нехарактерна для адаптивных веб-дизайнов. Однако, поскольку довольно легко внедрить частичную

поддержку этой спецификации (итоговый код сможет пройти валидацию на предмет соответствия требованиям HTML5 без дополнительных усилий с вашей стороны), едва ли имеет смысл включать ее в любую веб-страницу, которую вы будете писать на HTML5 начиная с этого дня и далее. Но хватит слов! Посмотрим, как все это работает.

Взгляните на нашу новую навигационную область на HTML5:

```
<nav>
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</nav>
```

Мы можем сделать так, что совместимые с WAI-ARIA экранные дикторы смогут легко переходить в эту область. Для этого добавим для нее атрибут `role`, как показано в приведенном далее фрагменте кода:

```
<nav role="navigation">
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</nav>
```

Насколько все просто? Существуют роли ориентиров для следующих частей структуры документа.

- `application` — применяется для определения области веб-приложения.
- `banner` — используется для определения области, имеющей ширину сайта (а не документа). Например, такой области, как верхний колонтитул или логотип сайта.
- `complementary` — применяется для определения области, являющейся дополнительной по отношению к основному разделу страницы. На нашем сайте *And the winner isn't...* дополнительными будут считаться области **UNSUNG HEROES** и **OVERHYPED NONSENSE**.
- `contentinfo` — ее следует использовать, когда дело касается информации об основном содержимом. Например, для вывода информации об авторских правах в нижнем колонтитуле страницы.

- `form` — как вы уже догадались, эта роль относится к формам! Однако если форма, о которой идет речь, предназначена для поиска, то вместо роли `form` следует использовать `search`.
- `main` — применяется для определения основного содержимого страницы.
- `navigation` — используется для определения навигационных ссылок текущего документа или связанных документов.
- `search` — применяется для определения области, которая выполняет поиск.



ПОДРОБНЕЕ ОБ ARIA

ARIA не ограничивается лишь ролями ориентиров. Если вы решите пойти еще дальше, то полный перечень ролей с кратким описанием их предназначения найдете по адресу http://www.w3.org/TR/wai-aria/roles#role_definitions.

Перейдем к расширению нашей текущей HTML5-версии разметки сайта *And the winner isn't...* с использованием соответствующих ролей ориентиров ARIA:

```
<body>
<div id="wrapper">
  <!-- верхний колонтитул и навигация -->
  <header role="banner">
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav role="navigation">
      <ul>
        <li><a href="#">Why?</a></li>
        <li><a href="#">Synopsis</a></li>
        <li><a href="#">Stills/Photos</a></li>
        <li><a href="#">Videos/clips</a></li>
        <li><a href="#">Quotes</a></li>
        <li><a href="#">Quiz</a></li>
      </ul>
    </nav>
  </header>
  <!-- содержимое -->
  <div id="content" role="main">
    
    <h1>Every year <em>when I watch the Oscars I'm annoyed...</em></h1>
    <p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and
    <b>Munich</b> get the statue whilst the real cinematic heroes lose out. Not very
    Hollywood is it?</p>
    <p><i>We're here to put things right.</i></p>
    <a href="#">these should have won &raquo;quote;</a>
  </div>
  <!-- врезка -->
  <aside>
```

```

<section role="complementary">
  <div class="sideBlock unsung">
    <h1>Unsung heroes...</h1>
    <a href="#"></a>
    <a href="#"></a>
    </div>
  </section>
  <section role="complementary">
    <div class="sideBlock overHyped">
      <h1>Overhyped nonsense...</h1>
      <a href="#"></a>
      <a href="#"></a>
    </div>
  </section>
</aside>
<!-- нижний колонтитул -->
<footer role="contentinfo">
  <p>Note: our opinion is absolutely correct. You are wrong, even if you think you
are right. That's a fact. Deal with it.</p>
</footer>
</div>
</body>

```



ПРОТЕСТИРУЙТЕ СВОИ ДИЗАЙНЫ БЕСПЛАТНО С ПОМОЩЬЮ NONVISUAL DESKTOP ACCESS (NVDA)

Если вы разрабатываете сайты на платформе Windows и желаете протестировать свои улучшенные благодаря ARIA дизайны с использованием экранного диктора, то можете сделать это бесплатно с помощью NVDA. Это решение доступно по адресу <http://www.nvda-project.org/>.

Надеюсь, краткое введение в WAI-ARIA показало, насколько легко внедрить частичную поддержку этой спецификации для пользователей вспомогательных технологий, и вы задумаетесь о том, чтобы задействовать **WAI-ARIA** для улучшения своего следующего HTML5-проекта.



СТИЛИЗАЦИЯ РОЛЕЙ ARIA

Как и любые атрибуты, роли ARIA можно стилизовать напрямую, используя селектор атрибутов. Например, вы можете добавить CSS-правило для роли `navigation`, напечатав следующий код: `nav[role="navigation"] {}`.

4.7. Вложение мультимедиа

Термин «HTML5» впервые вошел в лексикон многих людей, когда компания Apple отказалась внедрять поддержку Flash в свои iOS-устройства. Технология Flash заняла лидирующее положение на рынке в качестве излюбленного плагина для

воспроизведения видео в браузерах. Однако вместо использования этой проприетарной технологии от Adobe компания Apple решила положиться на HTML5 как на средство, позволяющее справиться с обработкой мультимедиа. Несмотря на то что технология HTML5 в любом случае успешно продвигалась в этой области, ее публичная поддержка со стороны Apple серьезно поспособствовала тому, что ее медиainструменты стали активнее использоваться большим количеством разработчиков.

Как вы, возможно, уже догадались, Internet Explorer версии 8 и ниже не поддерживает HTML5-видео и аудио. Однако для устаревших версий браузера от Microsoft можно легко обеспечить обходные пути в виде резервных вариантов, о которых мы вскоре поговорим. Большинство прочих браузеров, которые можно назвать современными (Firefox версии 3.5 и выше, Chrome версии 4 и выше, Safari версии 4, Opera версии 10.5 и выше, Internet Explorer версии 9 и выше, iOS версии 3.2 и выше, Opera Mobile версии 11 и выше, Android версии 2.3 и выше), отлично справляются с HTML5-видео и аудио.

4.8. Добавление видео и аудио с использованием HTML5

Буду с вами честен. Я всегда считал добавление мультимедиа на веб-страницы с использованием HTML 4.01 настоящей мукой. Это не сложный, а просто запутанный процесс. HTML5 делает все значительно проще. Необходимый для этого синтаксис очень похож на тот, что используется для добавления изображений:

```
<video src="myVideo.ogg"></video>
```

Это глоток свежего воздуха для большинства веб-дизайнеров! Вместо того чтобы писать объемный код, необходимый для включения видео в страницу, можно поручить тегу `<video></video>` (или тегу `<audio></audio>` для аудио) всю тяжелую работу. Можно также вставить примечание между открывающим и закрывающим тегами для уведомления пользователей на тот случай, если они будут применять несовместимые с HTML5 браузеры. Кроме того, имеются дополнительные атрибуты, например `height` и `width`. Добавим их:

```
<video src="video/myVideo.mp4" width="640" height="480">What, do you mean you don't understand HTML5?</video>
```

Теперь, если мы вставим приведенный чуть выше фрагмент кода в разметку нашей страницы, а затем взглянем на нее в браузере Safari, то увидим, что элементы управления воспроизведением отсутствуют. Чтобы они отобразились по умолчанию, нам необходимо добавить атрибут `controls`. Можно также добавить атрибут `autoplay` (но делать это все же не рекомендуется, поскольку никому не нравится, когда воспроизведение видео запускается автоматически). Добавление названных атрибутов демонстрируется в следующем фрагменте кода:

```
<video src="video/myVideo.mp4" width="640" height="480" controls autoplay>What, do you mean you don't understand HTML5?</video>
```

Результат применения приведенного чуть выше фрагмента кода показан на рис. 4.6.



Рис. 4.6. На нашей странице появилась область воспроизведения с загруженным видео

К прочим атрибутам относится `preload` для управления предварительной загрузкой мультимедиа (разработчики, впервые использовавшие HTML5, должны обратить внимание на то, что атрибут `preload` пришел на смену `autobuffer`), `loop` для повторного воспроизведения видео и `poster` для определения кадра видео, который будет показан до начала воспроизведения. Последний атрибут окажется полезен, если в воспроизведении видео возможна задержка. Чтобы использовать тот или иной атрибут, нужно просто добавить его в соответствующий тег. Вот пример включения всех упоминавшихся чуть ранее атрибутов:

```
<video src="video/myVideo.mp4" width="640" height="480" controls autoplay
preload="auto" loop poster="myVideoPoster.jpg">What, do you mean you don't understand
HTML5?</video>
```


Обеспечение альтернативных файлов-источников

Первоначальная HTML5-спецификация предусматривала поддержку всеми браузерами прямого воспроизведения (без плагинов) видео и аудио в контейнерах Ogg. Однако из-за споров между участниками рабочей группы HTML5 требование по поводу поддержки Ogg (включая Theora-видео и Vorbis-аудио) в качестве базового стандарта было удалено из более поздних версий HTML5-спецификации. По этой причине сейчас одни браузеры поддерживают воспроизведение одного набора видео- и аудиофайлов, а другие — другого. Например, Safari поддерживает использование в сочетании с элементами `<video>` и `<audio>` только медиа MP4/H.264/AAC, в то время как Firefox и Opera поддерживают лишь Ogg и WebM.

К счастью, есть способ обеспечить поддержку разных форматов в одном теге. Однако это не избавит нас от необходимости создавать разные версии нашего мультимедиа. Хотя мы все и держим пальцы скрещенными в надежде на то, что текущая ситуация со временем разрешится, между тем, вооружившись разными версиями нашего файла, мы можем разметить `<video>` следующим образом:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
  <source src="video/myVideo.ogv" type="video/ogg">
  <source src="video/myVideo.mp4" type="video/mp4">
  What, do you mean you don't understand HTML5?
</video>
```

Если браузер поддерживает воспроизведение Ogg, то он будет использовать соответствующий файл; в противном случае браузер перейдет к следующему тегу `<source>`.

Резервные варианты для устаревших браузеров

Использование тега `<source>` позволяет предусмотреть при необходимости несколько резервных вариантов. Например, наряду с обеспечением версий MP4 и Ogg мы, если потребуется включить подходящий резервный вариант для Internet Explorer версии 8 и ниже, можем добавить в качестве альтернативы Flash-видео. Более того, если у определенного пользователя не окажется подходящей технологии воспроизведения, мы можем предусмотреть для него ссылки на скачивание самих файлов:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
  <source src="video/myVideo.mp4" type="video/mp4">
  <source src="video/myVideo.ogv" type="video/ogg">
  <object width="640" height="480" type="application/x-shockwaveflash"
data="myFlashVideo.SWF">
    <param name="movie" value="myFlashVideo.swf" />
```

```

<param name="flashvars" value="controlbar=over&image=myVideoPoster.
jpg&file=video/myVideo.mp4" />

</object>
<p> <b>Download Video:</b>
MP4 Format: <a href="myVideo.mp4">"MP4"</a>
Ogg Format: <a href="myVideo.ogv">"Ogg"</a>
</p>
</video>

```

Теги `<audio>` и `<video>` работают почти одинаково

Тег `<audio>` работает по тому же принципу и с теми же атрибутами, что и `<video>`, не поддерживая лишь атрибуты `width`, `height` и `poster`. Да, вы можете использовать теги `<video>` и `<audio>` почти как взаимозаменяемые. Основное различие между ними заключается в том, что `<audio>` не имеет области воспроизведения для видимого содержимого.

4.9. Адаптивное видео

Вы уже видели, что обеспечение поддержки устаревших браузеров приводит к увеличению объема кода. То, что с тегом `<video>` поначалу было одной или двумя строками, в итоге превратилось в десять и более строк (и дополнительный Flash-файл) лишь для того, чтобы осчастливить устаревшие версии **Internet Explorer**! Что касается меня, то я обычно воздерживаюсь от применения Flash-видео в качестве резервного варианта, чтобы итоговый код занимал меньший объем памяти, однако у каждого сценария есть свои отличия.

Теперь единственная проблема с нашим прекрасным внедрением HTML5-видео заключается в том, что оно не является адаптивным. Так и есть. Мы проделали столько тяжелой работы, однако наш веб-дизайн нельзя назвать адаптивным. Взгляните на рис. 4.7 и постарайтесь всеми силами сдержать слезы.

К счастью, для вложенного HTML5-видео эту проблему легко решить. Просто удалите из разметки все атрибуты `height` и `width` (например, удалите `width="640" height="480"`) и добавьте такой CSS-код:

```
video { max-width: 100%; height: auto; }
```

Несмотря на то что такой способ отлично работает с файлами, которые могут размещаться локально, он не решает проблемы с видео, вложенным в `<iframe>` (привет **YouTube**, **Vimeo** и др.). Приведенный далее код добавляет на нашу страницу трейлер к фильму «Успеть до полуночи» (**Midnight Run**) с **YouTube**:

```
<iframe width="960" height="720" src="http://www.youtube.com/embed/B1_N28DA3gY"
frameborder="0" allowfullscreen></iframe>
```



Рис. 4.7. При уменьшении размера области просмотра часть видео обрезается

Несмотря на CSS-правило, которое я приводил ранее, вот что получится в итоге (рис. 4.8).

Я уверен, что Роберт Де Ниро был бы не очень рад увидеть такое! Эту проблему можно решить несколькими способами, однако самый простой из них, с которым мне доводилось сталкиваться, заключается в использовании небольшого jQuery-плагина под названием **FitVids**. Посмотрим, насколько просто использовать этот плагин, добавив его в код сайта And the winner isn't...

Прежде всего нам потребуется JavaScript-библиотека jQuery. Добавим ссылку на нее в элемент `<head>`. В данном случае я использую версию из **Content Delivery Network (CDN)**.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/
jquery.min.js"></script>
```

Скачайте плагин FitVids с сайта <http://fitvidsjs.com/> (дополнительную информацию о нем можно найти по адресу <http://daverupert.com/2011/09/responsive-video-embeds-with-fitvids/>).

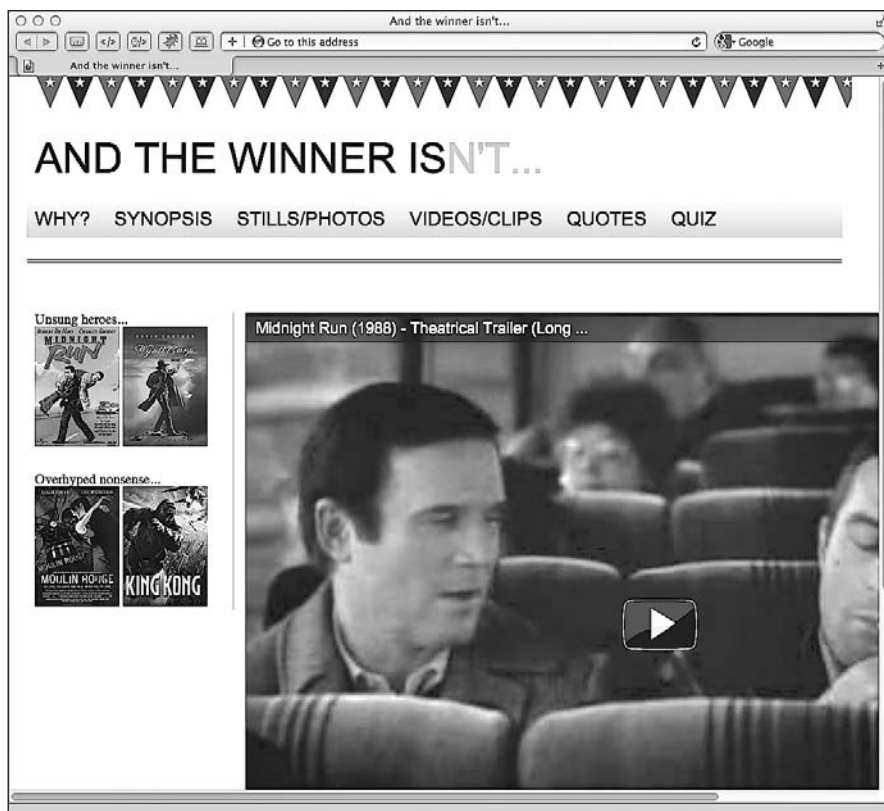


Рис. 4.8. Видео по-прежнему обрезается

Теперь сохраните JavaScript-файл **FitVids** в подходящей папке (свою я творчески назвал **js**), а затем добавьте ссылку на него в элемент `<head>`:

```
<script src="js/fitvids.js"></script>
```

И наконец, остается лишь использовать jQuery и нацелиться на определенный элемент, содержащий наше видео с YouTube. В данном примере я добавил трейлер к фильму «Успеть до полуночи» (Midnight Run) с YouTube в элемент `<div>` с идентификатором `#content`:

```
<script>
$(document).ready(function(){
    // Нацельтесь на свой .container, .wrapper, .post и т.д.
    $("#content").fitVids();
});
</script>
```

Вот и все. Благодаря jQuery-плагину **FitVids** теперь у нас имеется полностью адаптивное видео с YouTube (рис. 4.9).

Уф! Все исправлено. Это должно поспособствовать сохранению моих дружеских отношений с Робертом!



Рис. 4.9. Наша страница с видео стала адаптивной

4.10. Автономные веб-приложения

Несмотря на обилие в HTML5 функций, которые не особо помогают нам в стремлении к адаптивности (например, API-интерфейс Geolocation), автономные веб-приложения потенциально способны нам в этом помочь. Поскольку мы знаем, что количество мобильных пользователей, которые, вероятно, будут посещать наши сайты, постоянно растет, нужно обеспечить для них возможность просматривать содержимое сайтов даже при отсутствии подключения к Интернету. Автономные приложения HTML5 предназначены именно для этого.

Очевидно, что функциональность в виде возможности работы в автономном режиме как нельзя лучше подходит для веб-приложений. Представьте онлайнное веб-приложение для создания заметок. Пользователь может успеть написать только половину заметки до того, как разорвется соединение его мобильного телефона

с Интернетом. Благодаря автономным веб-приложениям HTML5 пользователи, столкнувшиеся с такой ситуацией, смогут продолжить писать свои заметки, а данные можно будет отправить позднее, как только восстановится соединение с Интернетом.

Замечательная особенность автономных веб-приложений HTML5 состоит в том, что их легко конфигурировать и использовать. Здесь мы задействуем их общим путем — для создания автономной версии нашего сайта. Это значит, что если пользователь захочет взглянуть на сайт при отсутствии подключения к Интернету, то он сможет сделать это.

Вкратце об автономных веб-приложениях

Автономные веб-приложения работают следующим образом: каждая страница, которая должна быть доступна в автономном режиме, указывает на текстовый файл с расширением `.manifest`. В нем содержится перечень всех ресурсов (HTML, изображения, JavaScript и т. д.), необходимых странице для того, чтобы она была доступна в автономном режиме. Браузеры с поддержкой автономных веб-приложений HTML5 (Firefox версии 3 и выше, Chrome версии 4 и выше, Safari версии 4 и выше, Opera версии 10.6 и выше, iOS версии 3.2 и выше, Opera Mobile версии 11 и выше, Android версии 2.1 и выше, Internet Explorer версии 10 и выше) считывают файл с расширением `.manifest`, загружают приведенные в нем ресурсы и кэшируют их локально на тот случай, если соединение с Интернетом будет разорвано. Все просто, не так ли? Сделаем это...

Делаем веб-страницы доступными в автономном режиме

В открывающем теге `<html>` мы указываем на файл с расширением `.manifest`:

```
<html lang="en" manifest="/offline.manifest">
```

Можете присвоить этому файлу любое имя по своему усмотрению, однако рекомендуется, чтобы у него было файловое расширение `.manifest`.



СОВЕТ

Вам потребуется добавить атрибут `manifest="/offline.manifest"` в тег `<html>` каждой страницы, которая должна быть доступна в автономном режиме.

Если вашим веб-сервером является Apache, то вам, скорее всего, потребуется добавить в файл `.htaccess` следующую строку:

```
AddType text/cache-manifest .manifest
```

В результате этого файл получит корректный тип MIME, которым является `text/cache-manifest`.

Пока открыт файл `.htaccess`, добавьте в него следующие строки:

```
<Files offline.manifest>
  ExpiresActive On
  ExpiresDefault "access"
</Files>
```

Этим вы сделаете так, что браузер больше не будет кэшировать кэш. Да, вы все правильно прочитали. Поскольку файл `offline.manifest` является статичным, браузер по умолчанию будет его кэшировать. Таким образом, предыдущий код дает команду серверу «сказать» браузеру, чтобы он этого не делал!

Теперь необходимо написать файл `offline.manifest`. Он будет информировать браузер о том, какие файлы следует сделать доступными в автономном режиме. Вот содержимое файла `offline.manifest` для сайта *And the winner isn't...*:

```
CACHE MANIFEST
#версия 1

CACHE:
basic_page_layout_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png

NETWORK:
*

FALLBACK:
/ /offline.html
```

Понятие файла манифеста

Файл манифеста должен начинаться с `CACHE MANIFEST`. Следующая строка представляет собой всего лишь комментарий, в котором указывается номер версии файла манифеста. Вскоре мы поговорим об этом подробнее.

В разделе `CACHE`: приводится перечень файлов, которые должны быть доступны в автономном режиме. Они должны соответствовать тем файлам, что упоминаются в `offline.manifest`, поэтому вам может потребоваться изменить пути в зависимости от того, какие ресурсы следует кэшировать. При необходимости также можно использовать абсолютные URL-адреса.

В разделе `NETWORK`: приводится список всех ресурсов, которые не должны кэшироваться. Считайте его «белым онлайн-списком». Все, что в нем перечислено,

всегда будет проходить мимо кэша при наличии сетевого соединения. Если вы хотите сделать содержимое своего сайта доступным там, где возможно подключение к Интернету (вместо того чтобы обращаться исключительно к автономному кэшу), то в этом вам поможет символ *. Он называется **подстановочным флагом белого онлайн-списка**.

В разделе FALLBACK: для определения URL-шаблона используется символ /. По сути, здесь задается вопрос: «Эта страница в кэше?» Если выяснится, что страница там, то отлично — она будет отображена. В противном случае пользователь увидит указанный файл — offline.html.

Автоматическое добавление страниц в кэш

В зависимости от обстоятельств возможно применение еще более легкого способа конфигурирования файла offline.manifest. Любая страница, указывающая на этот файл (как вы помните, для этого необходимо добавить manifest="/offline.manifest" в открывающий тег <html>), будет автоматически добавляться в кэш, когда пользователь посетит ее. Благодаря такому подходу каждая страница вашего сайта, на которую заходит пользователь, будет добавляться в его кэш, чтобы он смог снова посетить ее в автономном режиме. Вот как должно выглядеть содержимое файла манифеста:

```
CACHE MANIFEST
# Манифест кэша, версия 1
FALLBACK:
/ /offline.html
NETWORK:
*
```

При выборе этого подхода следует иметь в виду, что загружаться и кэшироваться будет только **HTML-код посещаемой страницы**. Однако этого не будет происходить с изображениями/JavaScript-кодом и прочими ресурсами, которые она может содержать или с которыми может быть связана. Если они важны для вас, то укажите их в CACHE:, как уже описывалось ранее в разделе «Понятие файла манифеста».

О комментарии с указанием номера версии

При внесении изменений в сайт вам придется так или иначе изменить файл offline.manifest и заново выгрузить его. В результате этого сервер сможет предоставить новую версию файла браузеру, который затем извлечет новые версии других соответствующих файлов и снова начнет автономный процесс. Я следую примеру Ника Пилгрима (Nick Pilgrim) (из отличной книги *Dive into HTML5* («Погружение в HTML5»)) и добавляю в верхнюю часть файла offline.manifest комментарий с указанием номера версии, который будет увеличиваться с каждым внесением изменений:

```
# Манифест кэша, версия 1
```


Просмотр сайта в автономном режиме

Теперь пришло время протестировать наше творение. Откройте страницу в браузере, совместимом с автономными веб-приложениями (рис. 4.10). Одни браузеры будут выдавать предупреждение насчет автономного режима (например, Firefox — обратите внимание на расположенную вверху строку), в то время как другие, к примеру Chrome, никак о нем не упомянут.

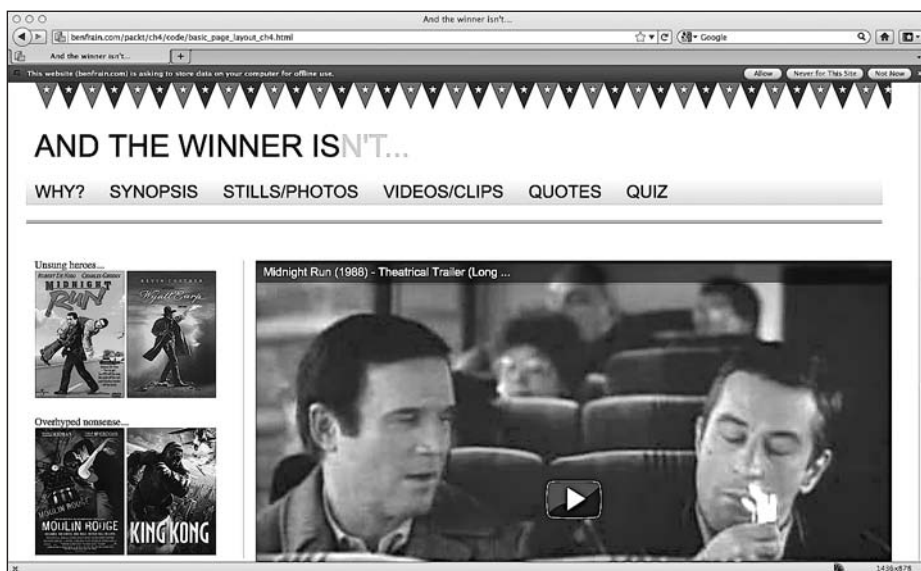


Рис. 4.10. Страница сайта And the winner isn't...

А теперь вырубите Интернет (ну, то есть отключите WiFi — просто это звучит не так драматично, как «вырубите») и обновите страницу в браузере. Следует надеяться, что после этого она будет выглядеть так же, как и при наличии соединения с Интернетом.

Устранение неполадок с автономными веб-приложениями

Когда у меня возникают проблемы с тем, чтобы заставить сайты корректно работать в автономном режиме, для устранения неполадок я предпочитаю использовать браузер Chrome (рис. 4.11). Встроенные в него инструменты разработчика включают удобный раздел Console (Консоль) (чтобы открыть его, щелкните на значке с изображением гаечного ключа справа от адресной строки, а затем выберите Tools ▸ Developer Tools (Инструменты ▸ Инструменты разработчика) и перейдите на вкладку Console (Консоль)). В этом разделе можно узнать об успехах и неудачах в работе автономного кэша и часто отмечается, что вы делаете неправильно. По своему опыту могу сказать, что обычно проблемы связаны с путями, например, для страниц не указано корректное местоположение файла манифеста.

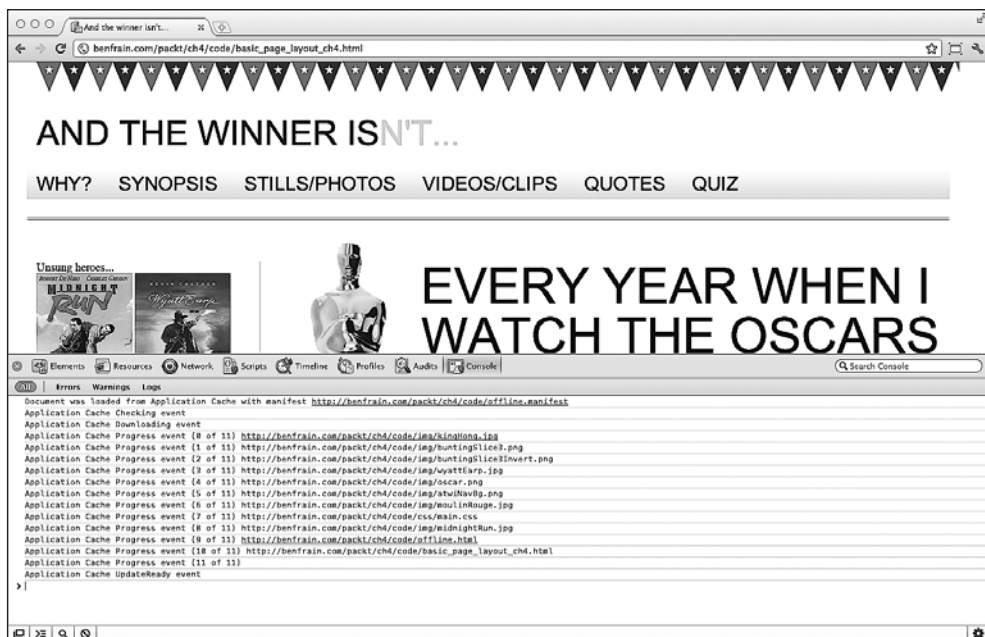


Рис. 4.11. Проверка работы сайта в браузере Chrome

Полную спецификацию автономных веб-приложений вы сможете отыскать по следующему адресу: <http://dev.w3.org/html5/spec/Overview.html#offline>.

4.11. Резюме

В этой главе мы рассмотрели все, начиная с основ создания страниц, которые смогут пройти валидацию на предмет соответствия требованиям HTML5, и заканчивая обеспечением работы страниц в автономном режиме, когда у пользователей нет возможности установить соединение с Интернетом. Кроме того, мы поговорили о вложении мультимедиа (в частности, видео) в разметку, а также о том, как адаптировать его к разным по размеру областям просмотра. Мы также рассмотрели особенности создания семантически насыщенного и значимого кода и способы оказания помощи пользователям, нуждающимся во вспомогательных технологиях. Однако наш сайт по-прежнему не лишен некоторых серьезных недостатков. Попросту говоря, он выглядит довольно захудало. Текст на нем не стилизован, и полностью отсутствуют такие элементы, как кнопки, которые были видимыми в оригинальной композиции. До сих пор мы совершенно обоснованно избегали использования изображений для решения этих проблем. Изображения нам просто не нужны! В последующих главах мы воспользуемся мощью и гибкостью CSS3 для создания быстро загружающегося и удобного в сопровождении адаптивного дизайна.

5 CSS3: селекторы, типографика и цветовые режимы

В главе 1 отмечалось, что количество людей, получающих доступ к сайтам по телекоммуникационным сетям мобильной связи, постоянно увеличивается. Скорость нынешних телекоммуникационных сетей сильно варьируется, и нам необходимо принимать во внимание быстроту передачи информации по каналам и, следовательно, время загрузки создаваемых сайтов. Раньше нам приходилось учитывать время, которое уходило бы на загрузку наших страниц вместе с изображениями и мультимедиа, при использовании модема, поддерживающего максимальную скорость передачи данных 56 Кбит/с.

В настоящее время мы сталкиваемся с похожими проблемами, касающимися времени загрузки. Процентные правила основанных на таблицах макетов снова становятся актуальными, равно как и необходимость пересмотреть каждую часть мультимедиа и «тяжеловесного» содержимого, которое мы добавляем на страницы. Хотя теперь устройства стали мобильными, быстрота, с которой они загружают содержимое, и то, во что это обходится (скорость и цена), сопоставимы с теми, что имели место в прошлом. Новое — это хорошо забытое старое! К счастью, CSS3 дает возможность в значительной степени снизить нашу зависимость от изображений для визуального оформления, давая нам инструменты, позволяющие создавать красивые сайты, которые к тому же очень быстро загружаются. Нам предстоит многое рассмотреть. В главе 6 мы поговорим о специфических CSS3-методиках, включая те, что позволяют создавать тени, отбрасываемые текстом и блочными элементами, а также градиенты и фоны, а в главе 7 взглянем на CSS3-анимации, трансформации и переходы.

В этой главе мы изучим следующие основы CSS3:

- функции, которые CSS3 предлагает разработчикам клиентских приложений;
- легко реализуемые и полезные CSS3-трюки (множественные колонки и перенос слов);
- анатомию CSS-правил;
- префиксы поставщиков и способы их применения;
- новые CSS3-селекторы и особенности их работы;
- пользовательскую типографику с использованием `@font-face`;
- применение цветовых режимов RGB и HSL с альфа-прозрачностью.

5.1. Что CSS3 предлагает разработчикам клиентских приложений

В прошлом мы либо делали ставку на то, что пользователи смирятся с длительным временем загрузки ради прекрасного дизайна (впрочем, на самом деле они не хотят делать этого!), либо убирали изображения, зачастую идя на компромисс в том, что касается идеального, с нашей точки зрения, дизайна ради обеспечения удобства пользования. CSS3 во многих отношениях сводит на нет необходимость идти на компромисс. Благодаря лишь нескольким строкам кода (и без каких-либо изображений!) CSS3 позволяет создавать экранные элементы, например, со скругленными углами, фоновые градиенты, тени, отбрасываемые текстом и блочными элементами, а также добавлять пользовательскую типографику и множественные фоновые изображения (ладно, согласен, в этом случае все же придется прибегнуть к использованию рисунков). Если этого мало, то замечу, что значительная часть базового взаимодействия, для обеспечения которого мы ранее полагались на JavaScript, например анимации, запускающиеся при наведении указателя мыши, может обеспечиваться исключительно посредством CSS3.

В CSS3 есть множество замечательных функций и инструментов, которые дают эффект экономии и поднимут наш сайт от уровня всего лишь «обычного сайта, сделанного адаптивным» до адаптивного сайта, созданного с ориентиром на будущее. Используя CSS3, мы сделаем так, что наш адаптивный веб-дизайн будет быстрее загружаться, требовать меньше ресурсов и станет намного более легким в сопровождении и внесении изменений в дальнейшем. Прежде чем мы приступим к рассмотрению всего этого, разберемся с очевидной, но зачастую игнорируемой проблемой.

Поддержка CSS3 в Internet Explorer версии от 6 до 8

За небольшим исключением (например, правило @font-face), некоторые из новых CSS3-модулей поддерживаются устаревшими версиями Internet Explorer (6, 7 и 8). Следует ли вам использовать CSS3 в своем дизайне? Как и всегда в веб-разработке, ответ звучит так: «Все зависит от обстоятельств».

Что касается меня, то в настоящее время я принципиально использую CSS3 для улучшения сайтов, а не для обеспечения важной функциональности. Меня полностью устраивает то, что элементы немного по-разному выглядят в разных браузерах. Я считаю, что это должно устраивать вас и ваших заказчиков. Возможно, вам будет полезно еще раз прочитать раздел «Как объяснить заказчикам, что сайты не должны выглядеть одинаково во всех браузерах» в главе 1. Какие части дизайна являются крайне необходимыми для того, чтобы он работал, а какие — для того, чтобы он выглядел надлежащим образом, каждый разработчик решает сам. Однако не стоит забывать, что существует много полизаполнений, позволяющих внедрять CSS3-функциональность в устаревшие версии Internet Explorer. Особенности применения таких полизаполнений более подробно рассматриваются в главе 9.



ПРИМЕЧАНИЕ

Полный перечень функций CSS 2.1 и CSS3, поддерживаемых в разных версиях Internet Explorer, вы сможете отыскать по следующему адресу: <http://msdn.microsoft.com/en-us/library/cc351024%28v=vs.85%29.aspx>.

Использование CSS3 для дизайна и разработки страниц в браузере

Не могу говорить за вас, однако я считаю переделывание изображений утомительным занятием. Вы знаете, какого рода высказывания заказчиков я здесь подразумеваю: «А можно сделать эти углы немного более скругленными?» или «Можно ли сделать градиент сверху немного темнее?» Как только мы послушно внесем необходимые изменения, зачастую слышим неизбежное: «О нет, раньше все было лучше. Вы можете вернуть все назад?» Этот процесс «хождения» туда-сюда, конечно же, необходим, поскольку нам, в конце концов, требуется тем или иным образом менять дизайн, чтобы посмотреть, как он работает. Однако CSS3 позволяет сделать большую часть всего этого лишь за секунды в коде, а не за минуты в графическом редакторе.

5.2. Анатомия CSS-правил

Прежде чем приступать к исследованию возможностей CSS3, во избежание путаницы определимся с терминологией, которую мы будем использовать для описания CSS-правил. Взгляните на следующий пример:

```
.round {  
  border-radius: 10px;  
}
```

Это правило состоит из **селектора** (.round), после которого идет **объявление** (border-radius: 10px;). Объявление дополнительно определяется **свойством** (border-radius;) и **значением** (10px;). Вы рады, что мы одинаково все понимаем? Отлично, продолжим.

5.3. Префиксы поставщиков и их использование

Если вести речь о спецификациях CSS3-модулей, то либо их еще предстоит утвердить (чем занимается консорциум W3C), либо полностью реализовать все предлагаемые ими функции в браузерах, из-за чего поставщики браузеров используют для тестирования новых «экспериментальных» CSS-параметров **префиксы поставщиков**. Несмотря на то что это помогает разработчикам браузеров реализовывать новые CSS3-модули, нам, как занимающимся написанием CSS3-кода, это немного усложняет жизнь. Взгляните на приведенный далее код для создания скругленного угла:

```
.round{  
  -khtml-border-radius: 10px; /* Konqueror */  
  -rim-border-radius: 10px; /* RIM */  
  -ms-border-radius: 10px; /* Microsoft */  
  -o-border-radius: 10px; /* Opera */
```

```
-moz-border-radius: 10px; /* Mozilla (например, Firefox) */  
-webkit-border-radius: 10px; /* Webkit (например, Safari и Chrome) */  
border-radius: 10px; /* W3C */  
}
```

В этом примере вы можете видеть свойства с префиксами поставщиков (причем это далеко не исчерпывающий список), каждое из которых обладает собственным уникальным префиксом, например `-webkit-`, что означает основанные на WebKit браузеры, или `-ms-` — префикс такого поставщика, как Microsoft, подразумевающий Internet Explorer, и т. д. Из-за особенностей работы CSS браузер будет двигаться вниз по таблице стилей строка за строкой, применяя соответствующие свойства и игнорируя те, которые не может распознать.

Кроме того, соответствующие свойства, расположенные далее в таблице стилей, имеют больший приоритет, чем предшествующие им. Благодаря такому каскадированию мы можем указать сначала наши свойства с префиксами поставщиков, а затем надлежащую (но, возможно, еще ожидающую своей реализации) версию без префикса, точно зная, что, когда она будет полностью принята, браузеры станут задействовать надлежащую версию вместо экспериментальной, ориентированной на определенный браузер и указанной до нее.



ВЫРЕЗАННЫЕ ФРАГМЕНТЫ КОДА И JAVASCRIPT-РЕШЕНИЯ ДЛЯ БЫСТРОГО ДОБАВЛЕНИЯ ПРЕФИКСОВ CSS3

Возможно, вам покажется удобным сохранять вырезанные фрагменты кода распространенных CSS3-правил, которые содержат все необходимые свойства с префиксами поставщиков. Таким образом, вы сможете просто вставлять их без необходимости заново печатать каждый раз. Во многих программах для редактирования кода (или интегрированных средах разработки (IDE — Integrated Development Environment), как их часто называют) имеется функция сохранения вырезанных сегментов кода и доступа к ним, которая при использовании CSS3 позволяет сэкономить массу времени. Кроме того, существуют JavaScript-решения, которые автоматически добавляют префиксы в CSS-файлы, и по адресу <http://leaverou.github.com/prefixfree/> вы сможете отыскать прекрасное решение такого рода под названием `-prefix-free`.

Кроме того, допустимо указывать все версии префиксов поставщиков того или иного свойства. Однако в реальности так поступают лишь немногие разработчики. Вместо этого они либо нацеливаются на браузеры, которые считают наиболее часто используемыми, либо проверяют, какие именно браузеры поддерживают определенное свойство, прежде чем писать правило. Например, вы можете предпочесть написать так:

```
.round{  
  -moz-border-radius: 10px; /* Mozilla (например, Firefox) */  
  -webkit-border-radius: 10px; /* Webkit (например, Safari и Chrome) */  
  border-radius: 10px; /* W3C */  
}
```

В этом примере охватываются Firefox, Chrome, Safari и любые браузеры, которые полностью реализуют данное правило.

Я знаю, что вы думаете: «А разве указание версий одного и того же свойства, но с разными префиксами поставщиков не приведет к “раздутию” кода?» Что ж, при-

ведет, но в небольшой степени. Независимо от того, сколько свойств с префиксами мы добавим, это все равно будет более быстрое, изящное и надежное решение, чем использование изображений.

Прежде чем приступить к работе над сайтом, будет разумным взглянуть на текущую статистику использования браузеров. Так вы быстрее поймете, для каких браузеров вам придется обеспечить специфическую поддержку. Например, если вы ограничены во времени и бюджете, то, возможно, решите пренебречь префиксами поставщиков для браузеров, доля использования каждого из которых для просмотра вашего сайта составит менее 3 % от общего количества. Как и всегда, вам потребуется принять решение, исходя из нескольких переменных факторов.

Итак, мы знаем, что собой представляют префиксы и как применять их в правилах. Теперь взглянем на небольшие, легко реализуемые и полезные CSS3-трюки.



КОГДА ИМЕННО МОЖНО ИСПОЛЬЗОВАТЬ ОПРЕДЕЛЕННЫЕ CSS3- И HTML5-ФУНКЦИИ?

Поскольку мы все больше и больше углубляемся в CSS3, я очень рекомендую вам посетить сайт <http://caniuse.com>. Он поможет, если вам когда-нибудь потребует узнать текущий уровень поддержки браузерами определенной CSS3- или HTML5-функции (рис. 5.1). Помимо отображения версий браузеров, поддерживающих определенную функцию (которые можно выяснить, введя название этой функции), на этом веб-ресурсе приводится самая свежая глобальная статистика использования с сайта <http://gs.statcounter.com>.

When can I use... Suggestions Feed Twitter 652 115

Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers.

Latest update: IE10 Platform Preview 3 released (September 14, 2011)

Ads by Google Browsers HTML5 CSS3 Desktop Support HTML5 Encoding

Search: media

1 result found

Index Tables

Compatibility tables Browser comparison

Show options

Supported Not supported Partially supported Support unknown

CSS3 Media Queries - Candidate Recommendation

Method of applying styles based on media information. Includes things like page and device dimensions

Resources: IE demo page with information Demo page for page width

Global user stats*: Support: 63.09% Partial support: 0.03% Total: 63.12%

Show all versions	IE	Firefox	Safari	Chrome	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
3 versions back	6.0	3.6	3.2	10.0	10.6				
2 versions back	7.0	4.0	4.0	11.0	11.0	3.2		10.0	2.1
Previous version	8.0	5.0	5.0	12.0	11.1	4.0-4.1		11.0	2.2
Current		6.0		13.0	11.5	4.2-4.3	5.0-6.0	11.1	2.3
Near future	9.0	7.0	5.1	14.0	12.0				3.0
Farther future	10.0	8.0	6.0	15.0	12.1				

Note: Incomplete support by older webkit browsers refers to only acknowledging different media rules on page reload

Feedback

Рис. 5.1. Страница сайта When can I use...

5.4. Легко реализуемые и полезные CSS3-трюки

В повседневной работе одни новые CSS3-функции я использую постоянно, а в других у меня никогда не возникало необходимости. Прежде чем перейти к более сложным вещам, я подумал, что не помешает поделиться с вами парочкой CSS3-трюков, облегчающих жизнь, особенно при работе над адаптивными веб-дизайнами. Эти трюки позволяют решить простые задачи, которые раньше вызывали головную боль.

Множественные колонки CSS3 для адаптивных веб-дизайнов

Вам когда-нибудь требовалось сделать так, чтобы одна часть текста отображалась в нескольких колонках? До появления CSS3 приходилось распределять содержимое по разным элементам разметки, а затем стилизовать соответствующим образом. Изменение разметки в целях стилизации никогда не было хорошей практикой. CSS3 позволяет сделать так, чтобы какая-то часть содержимого разбивалась на несколько колонок. Взгляните на приведенную далее разметку:

```
<div id="main" role="main">
  <p>llloremipsimLorem ipsum dolor sit amet. consectetur
  // БОЛЬШОЕ КОЛИЧЕСТВО ДОПОЛНИТЕЛЬНОГО ТЕКСТА //
</p>
  <p>llloremipsimLorem ipsum dolor sit amet. consectetur
  // БОЛЬШОЕ КОЛИЧЕСТВО ДОПОЛНИТЕЛЬНОГО ТЕКСТА //
</p>
</div>
```

Вы можете сделать так, чтобы все содержимое «растекалось» по нескольким колонкам, которые либо имеют определенную ширину (например, 12em), либо представлены в определенном количестве (например, 3). Далее показано, как это делается.

Если речь идет о колонках определенной ширины, то используйте следующий синтаксис (префиксы поставщиков не указаны для краткости):

```
#main {
  column-width: 12em;
}
```

Этот код означает, что, независимо от размера области просмотра, содержимое будет простирается на колонки шириной 12em. Изменение размеров области просмотра будет приводить к динамическому изменению количества отображаемых колонок.

Например, вот как все будет выглядеть в Safari при ширине области просмотра 1024 пиксела (рис. 5.2).

На рис. 5.3 показано, как та же страница будет выглядеть на экране iPad в области просмотра шириной 768 пикселей.

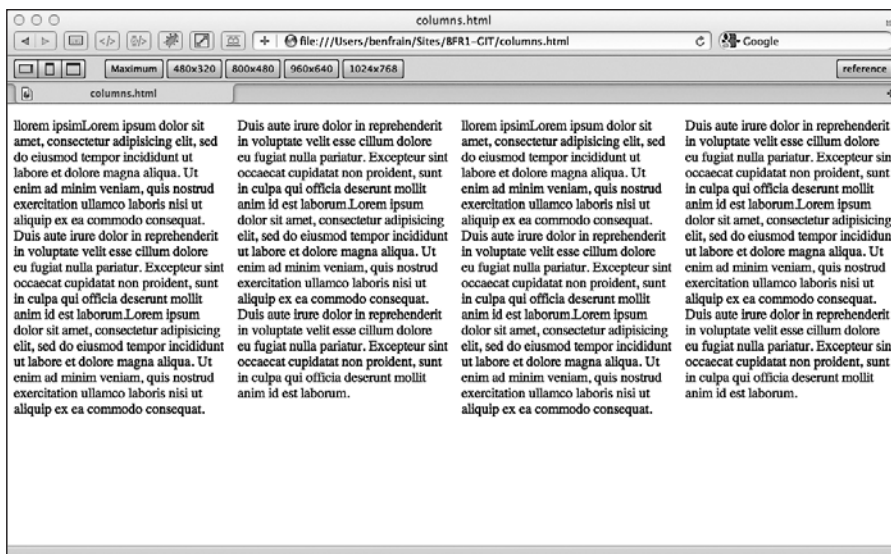


Рис. 5.2. Текст разбит на четыре колонки

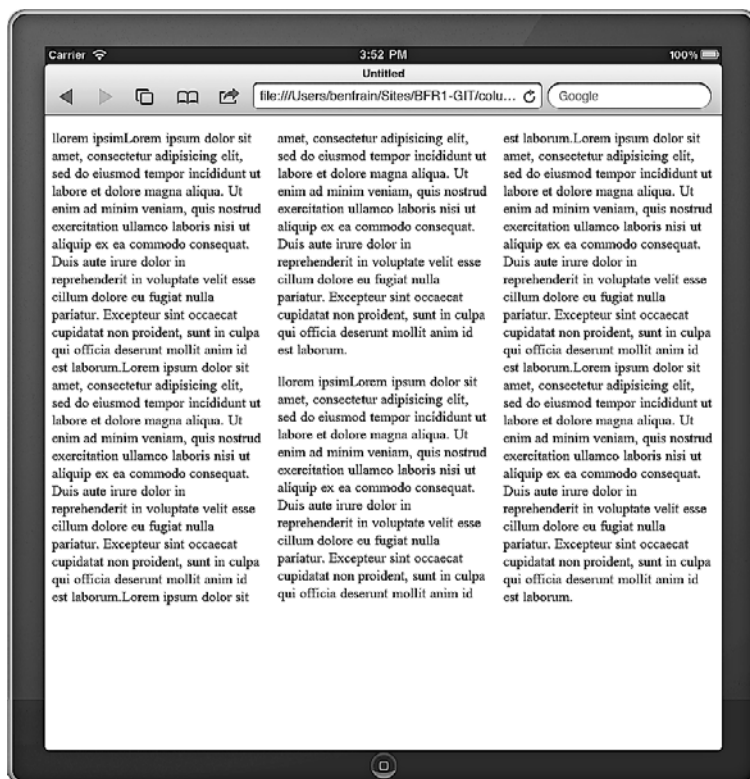


Рис. 5.3. При меньшей области просмотра количество колонок уменьшается

Превосходно приспособляющийся макет, требующий минимума усилий, — это мне нравится! Если вы предпочтете сохранить фиксированное количество колонок и варьировать ширину, то вам потребуется правило вроде следующего:

```
#main {
  column-count: 4;
}
```

Добавление промежутка и разделителя колонок. Мы можем пойти еще дальше и добавить определенный промежуток между столбцами, а также разделитель:

```
#main {
  column-gap: 2em;
  column-rule: thin dotted #999;
  column-width: 12em;
}
```

В результате у нас получится следующее (рис. 5.4).

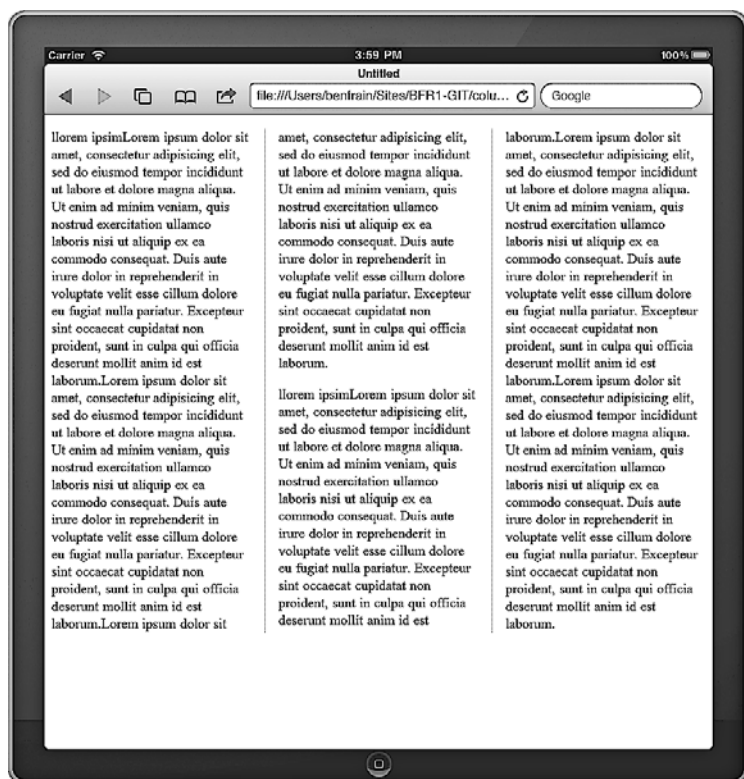


Рис. 5.4. Между колонками появился разделитель

Ознакомиться со спецификацией CSS3 Multi-column Layout Module вы сможете по адресу <http://www.w3.org/TR/css3-multicol/>.

Теперь запомните, что для максимальной совместимости вам придется использовать префиксы поставщиков в объявлениях, касающихся колонок.

Перенос слов

Сколько раз у вас возникала необходимость втиснуть длинный URL-адрес в очень маленькое пространство, и при этом вас постигало разочарование? Взгляните на проблему на рис. 5.5. Обратите внимание на адрес внизу справа, выступающий за пределы отведенного для него пространства.

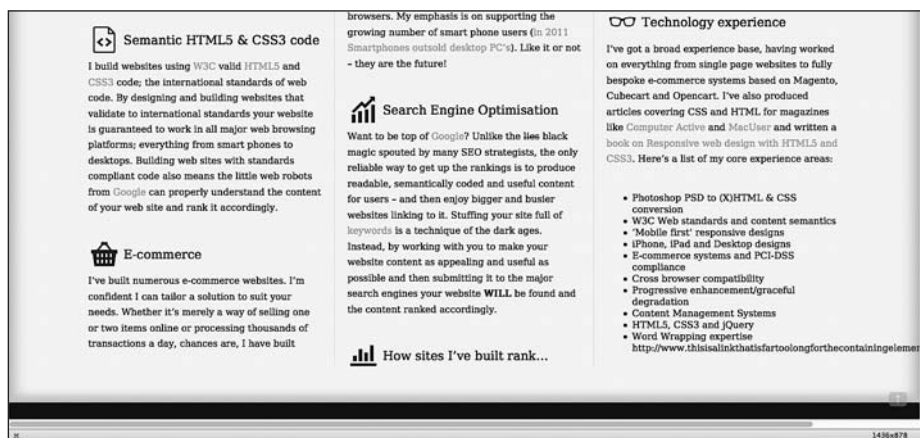


Рис. 5.5. Адрес сайта не уместается в ширину колонки

CSS3 позволяет решить эту проблему с помощью простого объявления, которое по стечению обстоятельств работает и в устаревших версиях Internet Explorer, причем даже в версии 5.5!

word-wrap: break-word;

Добавив это объявление в код элемента-контейнера, мы получим результат, показанный на рис. 5.6. Алле-гоп! Теперь длинный URL-адрес идеально вписывается в отведенное пространство благодаря переносу его части на новую строку!

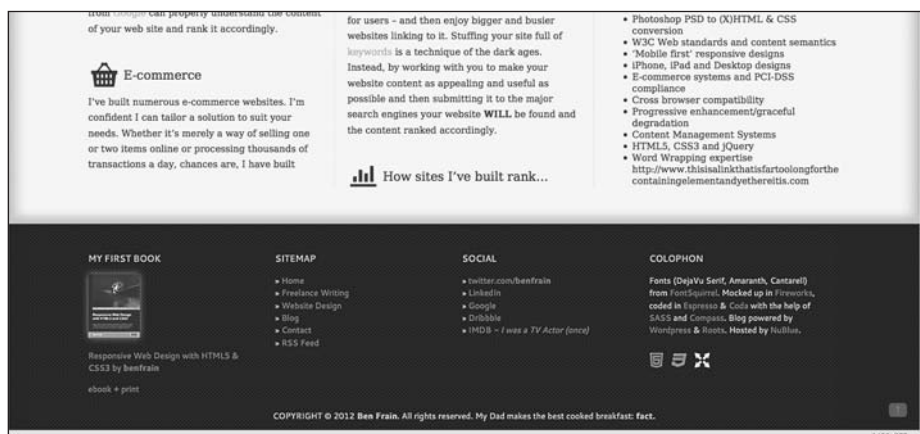


Рис. 5.6. Веб-адрес переносится на другую строку

5.5. Новые CSS3-селекторы и их использование

CSS3 обеспечивает невероятные возможности по выбору элементов в рамках страницы. Вам может показаться, что это звучит не очень впечатляюще, но, поверьте мне, эти возможности облегчат вам жизнь и вы полюбите CSS3! Я приведу доводы, подкрепляющие это смелое заявление...

Селекторы атрибутов CSS3

Вам, возможно, доводилось использовать атрибуты селекторов CSS для нацеливания правил. Например, взгляните на следующее правило:

```
img[alt] {  
  border: 3px dashed #e15f5f;  
}
```

Оно будет применяться к любым тегам `` в разметке с атрибутом `alt`:

```

```

Кроме того, можно сделать правило еще конкретнее, указав значение атрибута. Например, взгляните на приведенное далее правило:

```
img[alt="atwi_oscar"] {  
  border: 3px dashed #e15f5f;  
}
```

В результате оно будет направлено только на теги `` с атрибутом `alt`, имеющим значение `atwi_oscar`. Однако все это возможно и с использованием CSS2. Так что же нам дает здесь CSS3? Главным образом — три новых селектора атрибутов с совпадениями по подстроке...

Селекторы атрибутов CSS3 с совпадениями по подстроке

CSS3 позволяет выбирать элементы на основе подстроки их селектора атрибутов. Звучит сложно, но на самом деле все просто! Мы можем выбрать элемент, исходя из содержимого соответствующего атрибута. Есть три варианта:

- начинается ли атрибут с префикса;
- содержит ли он экземпляр;
- заканчивается ли атрибут суффиксом.

Посмотрим, как выглядят эти селекторы.

Селектор атрибутов с совпадениями по подстроке «начинается с...». Синтаксис такого селектора атрибутов выглядит следующим образом:

```
элемент[атрибут^="значение"]
```

На практике, если вам потребуется выбрать все теги `` в разметке сайта с атрибутом `alt`, значение которого *начинается с film*, для этого понадобится следующее правило:

```
img[alt^="film"] {  
    border: 3px dashed #e15f5f;  
}
```

Ключевым символом здесь является `^`, который означает «начинается с...».

Селектор атрибутов с совпадениями по подстроке «содержит экземпляр...». Синтаксис этого селектора атрибутов выглядит следующим образом:

`элемент[атрибут*="значение"]`

На практике, если вам потребуется выбрать все теги `` в разметке сайта с атрибутом `alt`, который *содержит film*, для этого будет нужно следующее правило:

```
img[alt*="film"] {  
    border: 3px dashed #e15f5f;  
}
```

Ключевым символом здесь является `*`, который означает «содержит...».

Селектор атрибутов с совпадениями по подстроке «заканчивается на...». Синтаксис такого селектора атрибутов выглядит следующим образом:

`элемент[атрибут$="значение"]`

На практике, если вам потребуется выбрать все теги `` в разметке сайта с атрибутом `alt`, который *заканчивается на film*, для этого понадобится следующее правило:

```
img[alt$="film"] {  
    border: 3px dashed #e15f5f;  
}
```

Ключевым символом здесь является `$`, который означает «заканчивается на...».

Практический пример из реальной жизни

Как эти селекторы атрибутов с совпадениями по подстроке в действительности могут нам помочь? Позвольте я приведу пример, в котором часто использую селекторы атрибутов CSS3. Я нередко создаю сайты с системой управления содержимым (например, Wordpress, Concrete или Magento). Такая система позволяет заказчику добавлять новые страницы. Скажем, он сможет добавить новость о своей компании или о выходе обновленной версии того или иного продукта. Каждый раз, когда он будет добавлять страницу с использованием системы управления содержимым, генерируемый HTML-код будет включать значение идентификатора `<body>` или другого соответствующего тега, что поможет отличить эту страницу по разметке от других. Например, один из моих клиентов был увлечен автоспортом, и на его

сайте имелся раздел с годовыми отчетами. Каждый тег `<body>` располагал идентификатором в виде соответствующего года:

```
<body id="2003">
```



В HTML5 ИДЕНТИФИКАТОРЫ МОГУТ НАЧИНАТЬСЯ С ЦИФР

Если вам пока непривычно писать код на HTML5, то вы, возможно, полагаете, что недопустимо применять идентификаторы, начинающиеся с цифр, как это было в HTML 4.01. Однако HTML5 снимает это ограничение. Главное, что вам нужно запомнить, — в идентификационном имени не должно быть пробелов и оно должно быть уникальным в рамках страницы. Дополнительную информацию вы сможете найти по адресу <http://dev.w3.org/html5/spec/Overview.html#the-id-attribute>.

Мне требовалось сделать так, чтобы ссылка **Racing History** в навигационной панели выделялась цветом при просмотре любой из страниц с годовыми отчетами, поскольку они были связаны с разделом **Racing History**. Однако вместо того, чтобы создавать правило стиля, охватывающее все будущие годы, я мог написать защитное правило CSS3 (их иногда называют защитными, поскольку они ограждают от будущих событий):

```
body[id^="2"] .navHistory { color: #00b4ff; }
```

Эта строка означает, что любой элемент с классом `.navHistory`, который является потомком `body` с идентификатором, начинающимся с 2 (например, 2002, 2003, 2004 и т. д.), будет окрашен в цвет, указанный шестнадцатеричным значением `#00b4ff`. Одно простое правило охватывает все возможные случаи. Разумеется, если только этот сайт не просуществует в своей нынешней форме до 3000 года — в такой ситуации, скорее всего, даже если я буду хорошо питаться и делать физические упражнения, то все равно не смогу продолжить заниматься его сопровождением...

Структурные псевдоклассы CSS3

Чем дольше вы будете заниматься созданием сайтов, тем чаще вам, скорее всего, будет необходимо снова и снова решать одну и ту же задачу. Рассмотрим типичный пример. Горизонтальные навигационные панели зачастую состоят из равномерно расположенных ссылок. Допустим, нам необходимо добавить поле справа и слева от каждого элемента списка за исключением первого и последнего элементов. Исторически сложилось так, что решить эту задачу можно добавлением семантически значимого имени класса в первый и последний элементы списка, как показано в выделенных строках в приведенном далее фрагменте кода:

```
<ul>
  <li class="first"><a href="#">Why?</a></li>
  <li><a href="#">Synopsis</a></li>
  <li><a href="#">Stills/Photos</a></li>
  <li><a href="#">Videos/clips</a></li>
  <li><a href="#">Quotes</a></li>
  <li class="last"><a href="#">Quiz</a></li>
</ul>
```

Затем, добавив пару правил в CSS, мы сможем изменить размеры полей для этих двух элементов списка:

```
li {
  margin-left: 5%;
  margin-right: 5%;
}
.first {
  margin-left: 0px;
}
.last {
  margin-right: 0px;
}
```

Это решение сработает, однако его нельзя назвать гибким. Например, при создании сайта на основе системы управления содержимым элементы списка для ссылки на новое содержимое могут добавляться автоматически, поэтому добавление или удаление класса `last` или `first` для соответствующего элемента списка в разметке может оказаться непростой задачей.

Селектор `:last-child`

Еще в версии CSS 2.1 был селектор, соответствующий первому элементу списка:

```
li:first-child
```

Однако в CSS3 появился селектор, который соответствует последнему элементу списка:

```
li:last-child
```

Если использовать эти селекторы сообща, то у нас не будет необходимости в дополнительных классах в нашей разметке.

Откорректируем навигацию сайта *And the winner isn't...*, задействовав эти селекторы в сочетании со свойством `display: table`. На рис. 5.7 показано, как все выглядит на текущий момент.

Теперь взглянем на графический макет (рис. 5.8).

Ссылки на навигационной панели растянуты на всю ширину дизайнера, и нам необходимо это воспроизвести. Наша разметка для навигационной области выглядит так:

```
<nav role="navigation">
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</nav>
```




Рис. 5.7. Страница сайта And the winner isn't... с обычной навигацией

Сначала мы сделаем так, чтобы элемент `nav` отображался как таблица:

```
nav {
  display: table;
  /* еще код... */
}
```

Затем мы обеспечим отображение `` в виде строки таблицы:

```
nav ul {
  display: table-row;
  /* еще код... */
}
```

И наконец, сделаем так, чтобы элементы списка отображались как ячейки таблицы:

```
nav ul li {
  display: table-cell;
  /* еще код... */
}
```




Рис. 5.8. Макет страницы, к которому мы стремимся

Это означает, что при добавлении дополнительных элементов списка они будут автоматически расставляться с соответствующими промежутками. И наконец, используем наши CSS-селекторы для выравнивания первого и последнего элементов списка по правому и левому краям:

```
nav ul li:last-child {
    text-align: right;
}
nav ul li:first-child {
    text-align: left;
}
```

Затем, открыв страницу в браузере, мы увидим, что навигационная область стала больше походить на ту, что приводилась в оригинальной композиции (рис. 5.9).



НЕ БЕСПОКОЙТЕСЬ, ВСЕ ЭТИ ТАБЛИЦЫ ИСПОЛЬЗУЮТСЯ ТОЛЬКО ДЛЯ ВИЗУАЛЬНОГО ПРЕДСТАВЛЕНИЯ!

Вы, возможно, задаетесь вопросом: «О чем же я, в конце концов, думаю?», решив, что мы используем таблицы для навигационного макета. Однако не стоит забывать, что все эти

таблицы предназначены только для визуального представления. Это означает, что они существуют лишь в CSS и не имеют отношения к разметке. Мы просто сообщаем браузеру, что желаем, чтобы соответствующие элементы отображались и вели себя так, как если бы они были таблицами, а не чтобы они на самом деле были ими. Отображение элементов разметки в такой манере также не мешает нам задействовать иной тип макета для областей просмотра с другой шириной, например `display: inline-block` для областей просмотра шириной менее 768 пикселей.

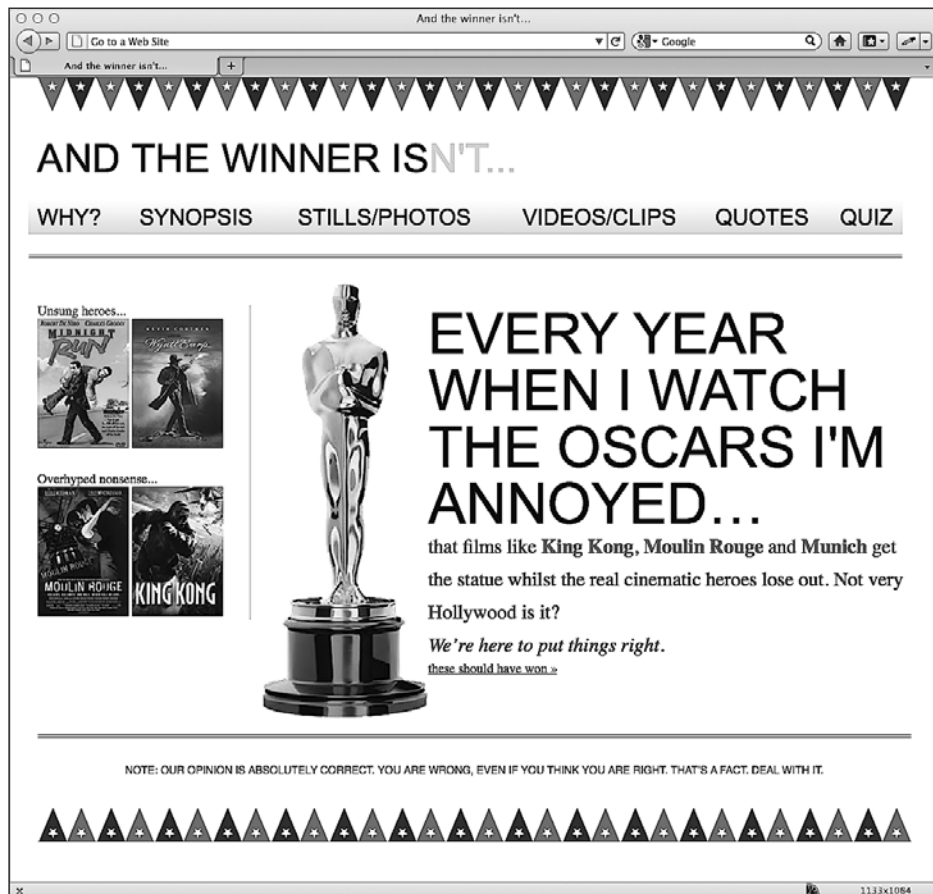


Рис. 5.9. Теперь строка навигации растянута на ширину страницы

Селектор `:nth-child`

Как насчет обеспечения чередования цветов, в которые были окрашены ссылки на навигационной панели из оригинальной композиции? Опять-таки в CSS3 есть селектор, позволяющий нам выполнить это без необходимости дополнительной разметки:

```
:nth-child(even)
```

Используем его для решения стоящей перед нами задачи, а затем рассмотрим некоторые из множества способов, которыми `:nth-child` позволяет решать проблемы, ранее требовавшие для этого дополнительной разметки. Обеспечим чередующуюся окраску ссылок на навигационной панели, чтобы одни из них были черными, а другие — красными, добавив следующее правило стиля:

```
nav ul li:nth-child(even) a {
  color: #fe0208;
}
```

Теперь наши навигационные ссылки окрашены в чередующиеся цвета (рис. 5.10).



Рис. 5.10. Цвета ссылок чередуются

Ну как? Ни единой jQuery-строки в коде сайта и никакой дополнительной разметки! Что я вам говорил? CSS3-селекторы — это отличная штука!

Принцип работы :nth-правил

Ничто так не заставляет трепетать слабых в математике веб-разработчиков и дизайнеров клиентских приложений, как правила на основе `:nth` (разве что за исключением тех, кто может выкрутиться, попросив вас написать небольшой объем кода

на РНР или помочь им с регулярными выражениями). Посмотрим, сможем ли мы в них разобраться и заслужить немного уважения со стороны специалистов по созданию серверных приложений.

Если вести речь о выборке элементов в древовидной структуре объектной модели документа DOM (Document Object Model) (или, проще говоря, элементов в разметке страницы), то CSS3 обеспечивает невероятную гибкость в этом плане благодаря правилам на основе `:nth` — `:nth-child(n)`, `:nth-last-child(n)`, `:nth-of-type(n)` и `:nth-last-of-type(n)`. Вы уже знаете, что можно применять значения (odd) или (even) (поскольку ранее нам пришлось корректировать нашу навигационную область), однако параметр (n) допускается использовать двумя другими способами:

- подставить целое число, например `:nth-child(2)`, то есть будет выбран второй элемент;
- подставить числовое выражение, например `:nth-child(3n+1)`, то есть выборка начнется с элемента номер 1 и будет выбран каждый третий элемент.

Вариант с целым числом достаточно прост для понимания: нужно лишь ввести номер элемента, который вы желаете выбрать. Вариант селектора с числовым выражением может оказаться немного сложным. Проанализируем его. Начнем справа — с того, что заключено в скобки. Таким образом, например, если вы хотите разобраться в том, к выбору чего именно приведет $(2n+3)$, начните справа (то есть с третьего элемента) и знайте, что выборке подвергнется каждый второй элемент, начиная с элемента номер 3. Я изменил наше правило, касающееся навигационной области, чтобы проиллюстрировать это:

```
nav ul li:nth-child(2n+3) a {  
    color: #fe0208;  
}
```

Как вы можете видеть на рис. 5.11, третий элемент списка окрасился в красный цвет, равно как и каждый второй последующий элемент за ним (если бы элементов списка было 100, то дальнейшей выборке подвергся бы каждый второй из них).



Рис. 5.11. Элементы списка окрашиваются через один, начиная с третьего

Как насчет того, чтобы выбрать все элементы, начиная со второго и далее? Что ж, несмотря на то что можно было бы написать `:nth-child(1n+2)`, в действительности не требуется указывать номер 1, поскольку, если не оговорено иное, n равняется 1. Следовательно, мы можем просто написать `:nth-child(n+2)`. Аналогичным

образом, если нам потребуется выбрать каждый третий элемент, мы, вместо того чтобы писать `:nth-child(3n+3)`, можем указать `:nth-child(3n)`, поскольку каждый третий элемент в любом случае будет начинаться с третьего элемента, то есть нам не нужно явным образом указывать это.

В выражении также можно использовать отрицательные числа, например, `:nth-child(3n-2)` будет означать, что выборка начнется с элемента -2 , а затем будет выбран каждый третий элемент. Следующее правило вносит изменения в нашу навигационную область:

```
nav ul li:nth-child(3n-2) a {  
    color: #fe0208;  
}
```

На рис. 5.12 показано, как будет выглядеть результат применения этого кода в браузере.



Рис. 5.12. В красный цвет окрашивается каждая третья ссылка

Надеюсь, теперь вы полностью разобрались во всем этом.

Правила `child` и `last-child` различаются в том, что вариант `last-child` работает с противоположного конца дерева документа. Например, `:nth-last-child(-n+3)` начинает выборку с третьего элемента с конца, а затем выбирает все элементы, следующие за ним. Вот к какому результату приведет использование этого правила в браузере (рис. 5.13).



Рис. 5.13. Красными стали три элемента с конца

И наконец, взглянем на `:nth-last-of-type`. В приводившихся ранее примерах дочерние элементы выбирались независимо от их типа, однако `:nth-last-of-type`

позволяет конкретно указать, элементы какого типа требуется выбрать. Взгляните на следующую разметку:

```
<ul>
  <li class="internal"><a href="#">Why?</a></li>
  <li><a href="#">Synopsis</a></li>
  <li class="internal"><a href="#">Stills/Photos</a></li>
  <li class="internal"><a href="#">Videos/clips</a></li>
  <li class="internal"><a href="#">Quotes</a></li>
  <li class="internal"><a href="#">Quiz</a></li>
</ul>
```

Обратите внимание, что для второго элемента списка не задан класс `internal`.

Теперь взгляните на такое правило:

```
nav ul li.internal:nth-of-type(n+2) a {
  color: #fe0208;
}
```

Как вы можете видеть, мы «говорим» CSS следующее: «Начиная со второго соответствующего элемента, нацелиться на каждый элемент `` с классом `internal`». Вот как результат будет выглядеть в браузере (рис. 5.14).



Рис. 5.14. В красный цвет окрашены все ссылки, для которых задан класс `internal`



В CSS3 ОТСЧЕТ ВЕДЕТСЯ НЕ ТАК, КАК В JQUERY!

Если вам доводилось использовать jQuery, то вы знаете, что там отсчет ведется от 0. Например, если при выборке элемента с применением jQuery указать целочисленное значение 1, то на самом деле это приведет к выборке второго элемента. Однако в CSS3 отсчет начинается с 1, то есть указание значения 1 приведет к выборке первого элемента, которому оно соответствует.

Селектор псевдокласса отрицания (`:not`)

Еще один удобный селектор — селектор псевдокласса отрицания. Он используется для выборки всего, что не является чем-то другим. Например, сохранив ту же разметку, что использовалась в приводившемся ранее примере, изменим наше правило следующим образом:

```
nav ul li:not(.internal) a {
  color: #fe0208;
}
```

В данном случае мы предпочли выбрать все элементы списка, для которых не определен класс `internal`. Результат отображения страницы в браузере показан на рис. 5.15.



Рис. 5.15. Теперь окрашена лишь та ссылка, для которой не установлен класс `internal`

До сих пор мы главным образом рассматривали то, что называется **структурными псевдоклассами** (полную информацию о них можно получить по адресу <http://www.w3.org/TR/selectors/#structural-pseudos>). Однако в CSS3 есть много других селекторов. Если вы работаете над веб-приложением, то вам стоит взглянуть на полный перечень псевдоклассов состояний элементов интерфейса пользователя (<http://www.w3.org/TR/selectors/#UIstates>).

Изменения в псевдоэлементах

Псевдоэлементы существуют еще с версии CSS2, а в спецификации CSS3 синтаксис их применения был очень незначительно пересмотрен. Чтобы освежить вашу память, напомним, что до сих пор `p:first-line` позволял нацеливаться на первую строку в теге `<p>`, а `p:first-letter` — на первую букву. CSS3 требует разделять эти псевдоэлементы двойным двоеточием, чтобы их можно было отличить от псевдоклассов. Следовательно, взамен мы должны писать `p::firstletter`. Надо отметить, что браузер Internet Explorer версии 8 и ниже не понимает синтаксис с двойными двоеточиями, а способен разобрать только с одинарными.

Практичен ли `:first-line` для адаптивных веб-дизайнов? Весьма удобная особенность псевдоэлемента `:first-line` состоит в том, что он по-разному действует в зависимости от ширины области просмотра. Например, мы можем написать и применить следующее правило:

```
p::first-line {
  color: #ff0cff;
}
```

Как вы могли ожидать, первая строка окрашивается в ужасный розовый цвет (в тот момент я думал о фильме «Мулен Руж» (Moulin Rouge)) (рис. 5.16).

Однако в более узкой области просмотра розовой становится уже более короткая первая строка текста (рис. 5.17).



Рис. 5.16. Первая строка окрасилась в розовый цвет



Рис. 5.17. При уменьшении ширины области просмотра окрашенная строка также становится меньше

Таким образом, существует удобный способ сделать так, чтобы в адаптивном веб-дизайне первая отображаемая (в том виде, в каком ее визуализирует браузер, а не в том, в каком она представлена в разметке) строка текста была оформлена по-другому, нежели остальные. И при этом не надо прибегать к изменению разметки.

Надеюсь, этот краткий экскурс в область CSS3-селекторов продемонстрировал вам, как они помогают избежать использования дополнительной разметки в адаптивных веб-дизайнах, которая «утяжеляет» их код. В прошлом нам приходилось применять JavaScript-библиотеки вроде jQuery, чтобы осуществлять сложные выборки, однако CSS3 зачастую сводит эту необходимость на нет. Кроме того, отрадно знать, что модуль CSS3 Selectors уже имеет статус W3C Recommendation (REC). Таким образом, это весьма «зрелый» модуль, который вряд ли значительно изменится в дальнейшем.

5.6. Пользовательская веб-типографика

В течение многих лет мы обходились скучным набором безопасных веб-шрифтов. Когда для того или иного дизайна требовалось специфическое шрифтовое оформление, мы обычно замещали шрифт графическим элементом, а также использовали правило `text-indent` для задания отступа требуемого текста в окне просмотра.

Существует и несколько других способов добавления специфического оформления в страницы. sIFR (<http://www.mikeindustries.com/blog/sifr/>) и Cufyn (<http://cufon.shoqolate.com/generate/>) задействуют соответственно Flash и JavaScript для переделки текстовых элементов таким образом, чтобы они отображались с использованием требуемых шрифтов. Однако для адаптивного веб-дизайна нам необходим компактный механизм обеспечения содержимого, а использования изображений и лишнего кода по возможности следует избегать. К счастью, в CSS предусмотрено средство для добавления пользовательской типографики, о котором мы сейчас и поговорим.

CSS-правило @font-face

CSS-правило `@font-face` появилось еще в CSS2 (однако в вышедшей позднее версии CSS 2.1 оно отсутствовало). Оно даже частично поддерживалось браузером Internet Explorer 4 (нет, это не шутка)! Так что же оно делает здесь, когда мы вроде как должны вести речь о CSS3?

Оказывается, правило `@font-face` снова было представлено с выходом модуля CSS3 Fonts (<http://www.w3.org/TR/css3-fonts>). Из-за исторически сложившихся юридических неурядиц по поводу использования шрифтов в веб-страницах это правило лишь недавно начало набирать силу. Кроме того, существует такая проблема, как рассогласованность форматов и реализаций шрифтов от разных поставщиков. Например, формат шрифтов **Embedded OpenType (EOT)** предпочитался браузером Internet Explorer (и никаким больше). Другие браузеры отдают предпочтение более распространенному формату **TrueType (TTF)**, хотя также существуют **Scalable Vector Graphics (SVG)** и **Web Open Font Format (WOFF)**.

Если вести речь об использовании @font-face для добавления требуемой вам веб-типографики, то здесь есть как хорошие, так и плохие новости. Сначала о плохих...

До тех пор пока не победит какой-то один универсальный формат, нам придется предусматривать наличие различных версий одного и того же формата, чтобы охватить разные браузерные реализации. Как и в случае с конкурирующими видеоформатами, нам нужно дождаться, пока какой-то один формат не одержит победу над остальными, прежде чем мы откажемся от поддержки других форматов.

Тем не менее хорошие новости заключаются в том, что добавление пользовательских шрифтов для любого браузера теперь не составляет труда. Сделаем это!

Реализация веб-шрифтов с помощью @font-face

Облагодородим типографику сайта And the winner isn't... с помощью CSS-правила @font-face.

Сначала нам потребуются шрифты. В Интернете существует множество замечательных источников веб-шрифтов, как бесплатных, так и платных. Я предпочитаю Font Squirrel (www.fontsquirrel.com) (рис. 5.18), однако Google тоже предлагает бесплатные веб-шрифты, обеспечиваемые правилом @font-face (www.google.com/webfonts). Существуют также платные службы от Typekit (www.typekit.com) и Font Deck (www.fontdeck.com).

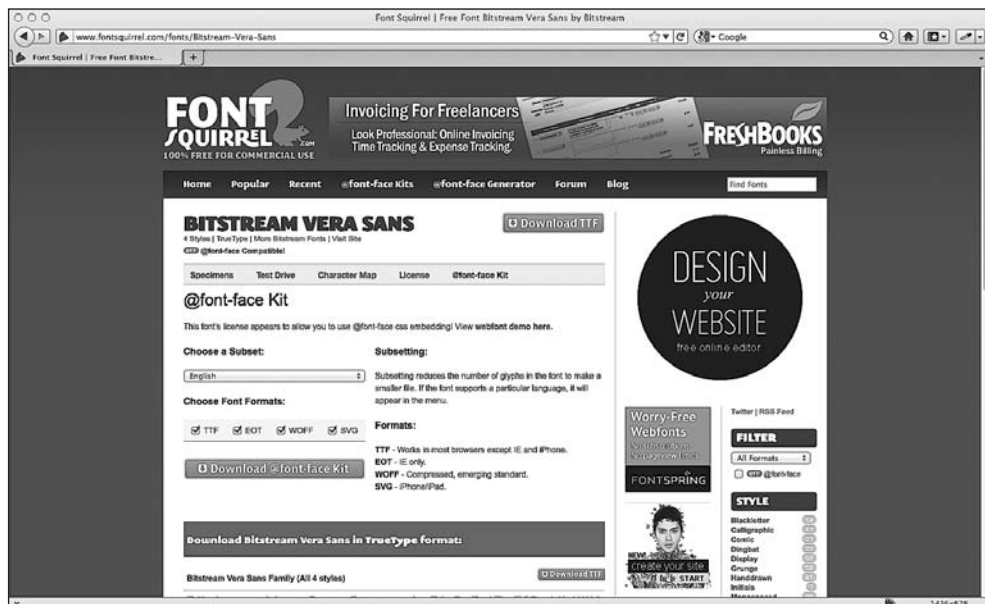


Рис. 5.18. Сайт Font Squirrel

Случилось так, что все шрифты, использованные в моей композиции, бесплатно доступны на сайте Font Squirrel (знаю, что я скряга!). Это шрифты Bebas Neue, Bitstream Vera Sans и Collaborate Thin. Скачав соответствующий набор @font-face для каждого шрифта с сайта Font Squirrel, загляните в ZIP-файл каждого из них, где определенный шрифт представлен в разных форматах (WOFF, TTF, EOT и SVG). Кроме того, там будет файл `stylesheet.css`, содержащий правила для требуемого шрифта. Например, правило для Bebas Neue выглядит следующим образом:

```
@font-face {
  font-family: 'BebasNeueRegular';
  src: url('BebasNeue-webfont.eot');
  src: url('BebasNeue-webfont.eot?#iefix') format('embedded-opentype'),
        url('BebasNeue-webfont.woff') format('woff'),
        url('BebasNeue-webfont.ttf') format('truetype'),
        url('BebasNeue-webfont.svg#BebasNeueRegular') format('svg');
  font-weight: normal;
  font-style: normal;
}
```

Во многом подобно тому, как это происходит с префиксами поставщиков, браузер будет применять стили из этого списка свойств (при этом более низко расположенные свойства, если они окажутся применимы, будут приоритетнее предшествующих) и игнорировать те, которые будут ему не понятны. Таким образом, независимо от того, каким именно будет браузер, должен быть шрифт, который он сможет использовать.

А теперь, хоть следующий блок кода и порадует любителей копирования и вставки, важно обратить внимание на пути, ведущие к сохраненным шрифтам. Например, я имею склонность копировать шрифты из ZIP-файла и сохранять их в папке, творчески названной **fonts**, находящейся на том же уровне, где располагается моя папка **css**. Следовательно, поскольку обычно я копирую соответствующее правило в свою основную таблицу стилей, мне необходимо изменить пути. В результате мое правило станет выглядеть таким образом:

```
@font-face {
  font-family: 'BebasNeueRegular';
  src: url('../fonts/BebasNeue-webfont.eot');
  src: url('../fonts/BebasNeue-webfont.eot?#iefix')
format('embedded-opentype'),
        url('../fonts/BebasNeue-webfont.woff') format('woff'),
        url('../fonts/BebasNeue-webfont.ttf') format('truetype'),
        url('../fonts/BebasNeue-webfont.svg#BebasNeueRegular')
format('svg');
  font-weight: normal;
  font-style: normal;
}
```

Затем останется лишь задать нужный шрифт и его начертание (если требуется) в соответствующем правиле стиля. В данном случае я хочу внести изменение,

в результате которого навигационные ссылки будут отображаться с использованием шрифта Bebas Neue:

```
nav ul li a {
  height: 42px;
  line-height: 42px;
  text-decoration: none;
  text-transform: uppercase;
  font-family: 'BebasNeueRegular';
  font-size: 1.875em; /*30 × 16 */
  color: black;
}
```

Посмотрите на рис. 5.19. Вот как теперь будет выглядеть наша навигационная панель в браузере.



Рис. 5.19. Для ссылок установлен шрифт Bebas Neue

При замене шрифтов вам придется изменять задаваемые для них размеры. Однако, если вычисление размера имеющегося шрифта расшифровывается в комментарии сбоку, то вы сможете легко внести соответствующие исправления. Дополнительное преимущество заключается в том, что, если в композиции задействуются те же шрифты, что и в коде, то вы сможете взять их размеры прямо из файла композиции. Например, размер шрифта текста **EVERY YEAR** в моей композиции составляет 102 пиксела, поэтому, используя проверенную и надежную формулу *ширина цели × ширина контекста = результат*, я могу преобразовать это значение в показатель, выраженный в единицах em:

```
#content h1 {
  font-family: Arial, Helvetica, Verdana, sans-serif;
  text-transform: uppercase;
  font-family: 'BebasNeueRegular';
  font-size: 6.375em; /* 102 × 16 */
}
```

После внесения изменений в объявления `font-family` и `font-size` для всех соответствующих правил наша первая страница сайта будет выглядеть в браузере Google Chrome так, как показано на рис. 5.20 (с использованием формата шрифтов WOFF).

Дизайн по-прежнему небезупречен, однако теперь типографика идеально соответствует той, что представлена в оригинальной композиции. Для сравнения на рис. 5.21 показано, как наша страница будет выглядеть на экране iPad 2, операционная система которого поддерживает шрифты TTF начиная с версии iOS 4.2 и выше.



Рис. 5.20. Страница сайта с новым шрифтом в Google Chrome



Рис. 5.21. Страница сайта с новым шрифтом на экране iPad 2

5.7. Помогите — мои заголовки с применением CSS3-правила @font-face выглядят неаккуратно

Эта проблема довела меня до отчаяния, когда я впервые начал использовать шрифты, устанавливаемые правилом @font-face. С ней можно столкнуться не только в адаптивных веб-дизайнах, но и когда речь идет о любом заголовке, для которого используется шрифт, определяемый в @font-face. Вот фрагмент композиции дизайна, над которой мне довелось работать (рис. 5.22).

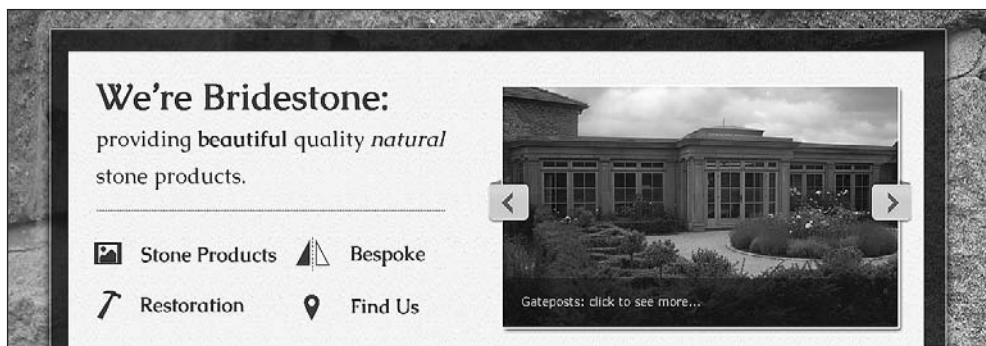


Рис. 5.22. Пример дизайна с пользовательскими шрифтами

Когда я создал сайт, соответствующая разметка выглядела следующим образом:

```
<div class="intro">
  <h1>We're Bridestone: <span>providing <b>beautiful</b> quality <i>natural</i>
  stone products.</span></h1>
  ...еще код...
</div> <!-- intro:КОНЕЦ -->
```

А вот как выглядел соответствующий CSS-код:

```
.intro h1 {
  font-family: CaudexBold, "Times New Roman", Times, serif;
  font-size: 2.63636364em;
  line-height: 1em;
}
.intro h1 span {
  font-size: 0.545454545em;
  font-family: CaudexRegular, "Times New Roman", Times, serif;
  font-weight: normal;
}
```

Несмотря на то что я использовал @font-face, чтобы применить точной такой же шрифт, как в композиции, заголовок выглядел в браузере немного неаккуратно (рис. 5.23).

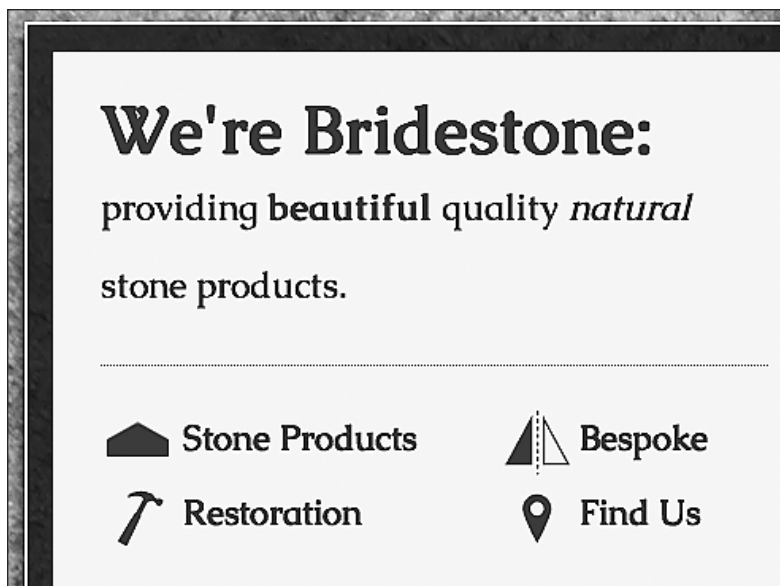


Рис. 5.23. Заголовок не вписывается в композицию

Надеюсь, вы понимаете, что текст **We're Bridestone** не соответствует композиции. Его шрифт толще, чем у остального текста, а это ухудшает восприятие!

Получается, что проблема связана с толщиной шрифта. Если явным образом не определить свойство `font-weight`, то многие браузеры будут применять шрифт стандартной толщины (обычно равной 700) для всех заголовочных элементов. Следовательно, решение данной проблемы — всегда определять свойство `font-weight` для всех шрифтов, устанавливаемых правилом `@font-face` и используемых для заголовков. Например, в данном случае я так изменил CSS-код:

```
.productIntro h1 {  
  font-family: CaudexBold, "Times New Roman", Times, serif;  
  font-weight: 400;  
  font-size: 2.63636364em;  
  line-height: 1em;  
}
```

Выделенная строка переопределяет значение `font-weight`, обычно используемое браузерами, и, как показано на рис. 5.24, дизайн в конечном итоге будет соответствовать композиции.

Примечание насчет адаптивных веб-дизайнов и пользовательской веб-типografии, добавляемой посредством @font-face. Применять правило `@font-face` для пользовательского оформления страницы — отличный подход. Единственное предостережение, о котором необходимо помнить при использовании этой методики в адаптивных веб-дизайнах, касается размера файла шрифтов. Например, для оформления текста на сайте *And the winner isn't...* применяются три пользовательских шрифта — *Bebas Neue*, *Bitstream Vera Sans* и *Collaborate Thin*. В самом худшем случае, если устройство, на котором будут открываться страницы сайта, требует

формата шрифтов SVG, это повлечет необходимость в дополнительных 70 Кбайт данных по сравнению с тем, когда используются безопасные веб-шрифты вроде Arial. Такие шрифты в отличие от других довольно легковесны! Обязательно проверьте размер файла пользовательских шрифтов, если хотите обеспечить наиболее быструю загрузку страниц.

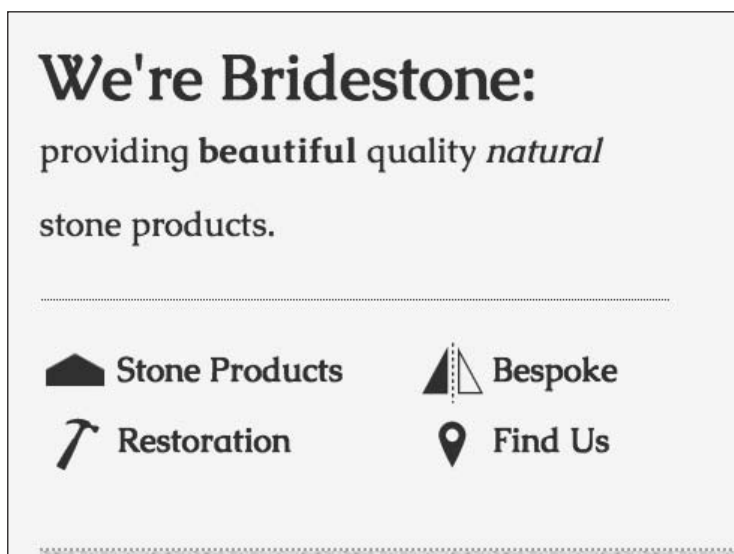


Рис. 5.24. Толщина шрифта для заголовка стала меньше, и он теперь лучше смотрится на странице



ДЕЙСТВИТЕЛЬНО ЛИ АДАПТИВНЫЕ ЕДИНИЦЫ ИЗМЕРЕНИЯ УЖЕ НА ПОДХОДЕ?

В текущих рабочих проектах модуля CSS3 Fonts есть ссылка на шрифты, связанные с областями просмотра (<http://www.w3.org/TR/css3-values/#viewport-relative-lengths>). Единицы измерения vw (viewport width — ширина области просмотра), vh (viewport height — высота области просмотра) и vm (viewport minimum — минимальная ширина или высота области просмотра; значение в vm рассчитывается относительно ширины или высоты области просмотра в зависимости от того, что из них меньше) в будущем могут стать инструментами, позволяющими сэкономить много времени. К сожалению, сейчас они не поддерживаются браузерами (за исключением Internet Explorer 9).

5.8. Новые цветовые форматы CSS3 и альфа-прозрачность

Ранее мы рассмотрели новые CSS3-возможности выборки, а также способы добавления в дизайны пользовательской типографики. Теперь взглянем на методы, благодаря которым CSS3 позволяет нам работать с цветом и которые ранее были просто невозможны.

Прежде всего CSS3 дает нам возможность использовать новые цветовые модели, такие как **RGB** и **HSL**. Кроме того, CSS3 позволяет применять их вместе с альфа-каналами (соответственно **RGBA** и **HSLA**).

RGB-цвет

RGB (Red, Green, Blue — **красный, зеленый, синий**) — это цветовая система, существующая уже в течение десятилетий. Она работает путем определения разных значений для красного, зеленого и синего компонентов цвета. Например, красный цвет, используемый для нечетных навигационных ссылок на сайте And the winner isn't..., на данный момент определен в CSS как шестнадцатеричное значение #fe0208:

```
nav ul li:nth-child(odd) a {  
    color: #fe0208;  
}
```

Однако при использовании CSS3 он может быть равным образом описан как RGB-значение:

```
nav ul li:nth-child(odd) a {  
    color: rgb(254, 2, 8);  
}
```

В цветовых палитрах большинства программ для редактирования изображений цвета демонстрируются как в виде шестнадцатеричных, так и в виде RGB-значений. На рис. 5.25 показана палитра цветов в Photoshop, где в полях R, G и B отображаются значения для каждого из каналов.

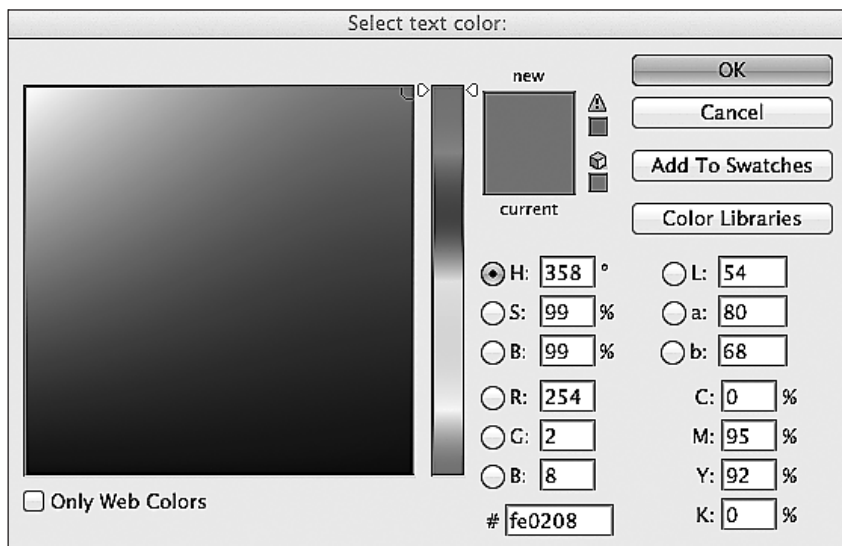


Рис. 5.25. Цветовая палитра в программе Photoshop

Как вы можете видеть, значение R равно 254, значение G — 2, а значение B — 8. Их легко перевести в значения CSS-свойства `color`. В CSS после определения цветового режима (например, RGB) значения красного, зеленого и синего цветов отделяются запятыми в скобках в упомянутом чуть ранее порядке.

HSL-цвет

CSS3 также позволяет объявлять значения цветов в системе HSL (Hue, Saturation, Lightness — оттенок, насыщенность, осветление).



HSL — ЭТО НЕ ТО ЖЕ САМОЕ, ЧТО HSB!

Не думайте ошибочно, что значение HSB (Hue, Saturation, Brightness — цветовой тон, насыщенность, яркость), которое демонстрируется в палитрах цветов программ для редактирования изображений вроде Photoshop, то же самое, что и значение HSL. На самом деле это далеко не так!

Использовать HSL удобно потому, что очень легко понять, как соответствующий цвет будет представлен на основе заданных значений. Например, если только вы не специалист в области выбора цветов, держу пари, что вы не смогли бы мгновенно ответить на вопрос: «Каким является цвет `rgb(255, 51, 204)`?» Есть желающие сразу дать ответ? Нет, я тоже к ним не отношусь. Однако стоит мне взглянуть на HSL-значение `hsl(315, 100%, 60%)`, я могу приблизительно сказать, что это цвет где-то между пурпурным и красным (вообще-то это веселый розовый цвет — возможно, мне все же начинает нравиться фильм «Мулен Руж»). Откуда я это знаю? Все просто...

HSL работает на основе 360-градусной шкалы для выбора цветов. Первое значение HSL-цвета, выраженное в градусах, представляет оттенок, который будет желтого цвета при 60°, зеленого — при 120°, голубого — при 240°, пурпурного — при 300° и, наконец, красного — при 360°. Таким образом, поскольку вышеупомянутый HSL-цвет имел оттенок со значением 315, легко понять, что он будет между пурпурным (при 300°) и красным (при 360°). Следующие два значения относятся соответственно к насыщенности и осветлению и выражаются в процентах, всего лишь изменяя основной оттенок. Для более насыщенного или красочного оформления используйте более высокое второе значение в процентах. Последнее значение, отвечающее за осветление, может варьироваться в промежутке от 0 %, что означает черный цвет, до 100 %, что означает белый цвет.

Таким образом, определив цвет как HSL-значение, можно легко создавать его вариации, изменяя процентные значения насыщенности и осветления. Например, красный цвет наших навигационных ссылок можно определить с использованием HSL-значений, как показано далее:

```
nav ul li:nth-child(odd) a {  
    color: hsl(359, 99%, 50%);  
}
```

Если бы мы решили сделать так, чтобы цвет навигационных ссылок становился немного темнее при наведении на них указателя мыши, то смогли бы использовать то же **HSL-значение** и **изменить лишь значение в процентах** (идущее последним), которое отвечает за осветление, как показано в следующем фрагменте кода:

```
nav ul li:nth-child(odd) a:hover {  
    color: hsl(359, 99%, 40%);  
}
```

В заключение необходимо отметить, что, если вы сможете запомнить мнемосхему **«Young Guys Can Be Messy Rascals»** (то есть **Yellow** (Желтый), **Green** (Зеленый), **Cyan** (Голубой), **Blue** (Синий), **Magenta** (Пурпурный), **Red** (Красный)) (или любую другую мнемосхему, которую, по вашему мнению, следует запомнить) для шкалы выбора цветов HSL, то сумеете на глаз указывать значения цветов HSL, **не прибегая к палитре цветов, а также создавать их вариации**. Пр продемонстрируйте этот фокус крутым специалистам по созданию серверных приложений на PHP и .NET во время вечеринки в офисе, и вы быстро заслужите их похвалу!

Значения резервных цветов для Internet Explorer версий 6, 7 и 8

Как вы, возможно, догадываетесь, значения **RGB** и **HSL** не поддерживаются в версиях **Internet Explorer** ниже 9. Следовательно, если потребуется объявление резервного цвета для таких версий **Internet Explorer**, указывайте его первым до значения **RGB** или **HSL**.

Например, правило, касающееся навигационных ссылок, которое приводилось чуть ранее, могло бы включать шестнадцатеричное значение резервного цвета, определенное следующим образом:

```
nav ul li:nth-child(odd) a {  
    color: #fe0208;  
    color: hsl(359, 99%, 50%);  
}
```

Альфа-каналы

До сих пор вам было простительно задаваться вопросом о том, с какой стати следует применять **HSL** или **RGB** вместо проверенных временем шестнадцатеричных значений, которые мы используем уже многие годы. **HSL** и **RGB** отличаются от шестнадцатеричных значений тем, что позволяют использовать альфа-канал прозрачности. Это означает, что сквозь элемент с альфа-прозрачностью будет видно то, что располагается под ним.

Внесем некоторые изменения в домашнюю страницу сайта And the winner isn't... ради наглядного примера. Сначала мы зададим состаренное фоновое изображение в элементе `body`:

```
body {  
  background: url(../img/grunge.jpg) repeat;  
}
```

Теперь добавим белый фон в элемент `<div>` с идентификатором `#wrapper` (куда входят все остальные элементы). Однако вместо того, чтобы задавать сплошной белый цвет с помощью шестнадцатеричного значения, используем HSLA-значение, как показано в выделенной строке в следующем фрагменте кода:

```
#wrapper {  
  margin-right: auto;  
  margin-left: auto;  
  width: 96%; /* Удерживание самого дальнего элемента <div> */  
  max-width: 1414px;  
  background-color: hsla(0, 0%, 100%, 0.8);  
}
```

Объявление цвета HSLA схоже по синтаксису со стандартными правилами HSL. Однако, кроме всего прочего, вам потребуется объявить значение как HSLA (вместо просто HSL) и задать дополнительное десятичное значение непрозрачности в промежутке от 0 (полная прозрачность) до 1 (полная непрозрачность). В данном случае мы указали, что наш белый элемент `<div>` с идентификатором `#wrapper` будет частично прозрачным. На рис. 5.26 показано, как результат будет выглядеть в браузере.

Синтаксис RGBA подчиняется тем же правилам, что и его HSLA-эквивалент. Необходимо использовать дополнительное значение непрозрачности после значения цвета:

```
background-color: rgba(255, 255, 255, 0.8);
```

Надеюсь, вы понимаете, что добавление альфа-канала к обоим цветовым режимам RGB и HSL позволяет проявлять большую гибкость при расположении элементов слоями. Это означает, что нам больше не нужно полагаться на прозрачность изображений (например, в формате PNG и GIF) для обеспечения визуального эффекта такого рода. И это отличная новость для тех, кто создает адаптивные веб-дизайны.



ПОЧЕМУ БЫ ПРОСТО НЕ ИСПОЛЬЗОВАТЬ НЕПРОЗРАЧНОСТЬ?

CSS3 также позволяет задавать для элементов непрозрачность. Значения непрозрачности определяются в промежутке между 0 и 1 в десятичных числах (например, непрозрачность, заданная как 0.1, равна 10 %). Однако все это отличается от RGBA и HSLA тем, что определение значения непрозрачности для того или иного элемента приводит к тому, что она применяется ко всему этому элементу целиком. А установка значения RGBA или HSLA между тем позволяет отдельным частям элемента иметь альфа-слой. Например, для фона элемента может быть задано значение HSLA, а для текста внутри него — сплошной цвет.



Рис. 5.26. Под частично прозрачным белым цветом видно фоновое изображение

Модуль CSS3 Color стал первым из модулей CSS3, достигшим продвинутого статуса W3C Recommendation (REC). Следовательно, как и CSS3 Selectors, этот модуль абсолютно готов к использованию уже сейчас, и можно с полной уверенностью сказать, что метод его реализации вряд ли изменится в будущем.

5.9. Резюме

В этой главе мы выяснили, насколько легко можно выбрать почти все, что нам необходимо в конкретной странице, используя новые CSS3-селекторы. Мы также взглянули на то, как можно создать адаптивные колонки за очень короткое время и решить распространенные рутинные задачи вроде переноса части длинного URL-адреса на новую строку. Теперь мы также имеем представление о новом модуле CSS3 Colors и о том, как можно применять цвета с использованием значений RGB и HSL наряду с прозрачными альфа-слоями для обеспечения прекрасного эстетического эффекта. Кроме того, в этой главе мы научились добавлять

пользовательские шрифты в дизайны с помощью правила `@font-face`, что раз и навсегда освобождает нас от необходимости применения скучного набора безопасных веб-шрифтов, к использованию которых мы раньше прибегали при работе над дизайнами.

Несмотря на все эти прекрасные новые функции и методики, мы лишь поверхностно затронули возможности CSS3. Теперь двинемся дальше и взглянем на другие методы, благодаря которым CSS3 может сделать адаптивные веб-дизайны настолько быстро загружающимися, эффективными и удобными в сопровождении, насколько это представляется возможным. Рассмотрим, как использовать тени, отбрасываемые текстом и блочными элементами, градиенты и множественные фоновые изображения.

6 Великолепная эстетика с использованием CSS3

В предыдущей главе мы говорили о легко реализуемых и полезных **CSS3-методиках**, помогающих создавать адаптивные веб-дизайны. Мы также внесли значительные изменения в визуальные элементы, задействовав **CSS3-правило @font-face** для добавления пользовательской типографики, и поработали с **CSS3-инструментами** для выборки элементов объектной модели документа **DOM**. Таким образом, теперь, когда мы рассмотрели основы **CSS3**, **взглянем на более продвинутые функции CSS3** и узнаем, как можно улучшить эстетику адаптивного веб-дизайна, прибегнув к **CSS3-методикам**, подавляющее большинство которых не требует ни единого графического изображения, настолько снижая требования нашего адаптивного веб-дизайна к скорости канала подключения пользователя, насколько это представляется возможным.

В этой главе мы поговорим о **CSS3**, в частности о том, как:

- создавать отбрасываемые текстом тени;
- создавать отбрасываемые блочными элементами тени;
- создавать градиентные фоны;
- использовать множественные фоновые изображения;
- применять фоновые градиенты **CSS3** для создания узоров;
- использовать **CSS3-правило @font-face** для создания значков, объем которых не предъявляет высоких требований к скорости канала подключения пользователя.

На данном этапе я повторяюсь, почему считаю **CSS3** таким полезным инструментом, когда речь идет об адаптивных веб-дизайнах: использование **CSS3** вместо изображений снижает количество запросов **HTTP** (и, следовательно, способствует более быстрой загрузке страниц), а также делает дизайны более гибкими и удобными в сопровождении. Эти преимущества будут кстати даже для типичного «настоящего» веб-дизайна с фиксированной шириной. Однако еще более важно то, что новые **CSS3-функции** прекрасно сочетаются с адаптивным веб-дизайном, поскольку позволяют с легкостью обеспечивать разные по размеру тени, отбрасываемые блочными элементами и текстом в различных по величине областях просмотра, без необходимости создавать и экспортировать хоть одно изображение. Предполагаю, что вы тоже все это понимаете, поэтому приступим.



ПРЕФИКСЫ ПОСТАВЩИКОВ

При использовании CSS3 не забывайте добавлять соответствующие префиксы поставщиков для обеспечения наиболее широкой кросс-браузерной совместимости. С другой стороны, если вы любите добавлять в свой код JavaScript, то примите во внимание упоминавшийся ранее сценарий `-prefix-free`. Он автоматически добавляет соответствующие префиксы в любые CSS-правила, которые в них нуждаются, позволяя вам вносить в таблицу стилей только W3C-версию. Этот сценарий доступен по адресу <http://leaverou.github.com/prefixfree/>.

6.1. Создание теней, отбрасываемых текстом, с помощью CSS3

Одно из наиболее широко реализованных CSS3-свойств называется `text-shadow`. Как и `@font-face`, оно поддерживалось в версии CSS2, но затем было исключено из версии CSS 2.1. К счастью, оно снова вернулось и широко поддерживается (всеми современными браузерами и Internet Explorer версии 9 и выше).

Взглянем на базовый синтаксис:

```
.элемент {  
    text-shadow: 1px 1px 1px #cccccc;  
}
```

Помните, что значения в сокращенных правилах всегда обеспечивают тень, которая будет располагаться правее и ниже текста. Следовательно, первое значение — это величина смещения тени вправо от текста, второе — количество пикселей, на которое тень будет смещена вниз относительно текста, третье — радиус размытия (расстояние, на которое будет простираться тень, прежде чем совсем исчезнуть), а последнее значение — цвет тени.

Допустимые шестнадцатеричные, HSL- и RGB-значения цветов

Значение цвета не обязательно должно быть определено как шестнадцатеричное. Это также может быть значение HSL(A) или RGB(A):

```
text-shadow: 4px 4px 0px hsla(140, 3%, 26%, 0.4);
```

Однако имейте в виду, что в таком случае браузер должен также поддерживать цветовые режимы HSL/RGB наряду со свойством `text-shadow`, чтобы обеспечить визуализацию соответствующего эффекта. Если бы мне потребовалось использовать HSLA или RGBA (благодаря совместимости с непрозрачностью), то я бы написал следующее:

```
text-shadow: 4px 4px 0px #404442;  
text-shadow: 4px 4px 0px hsla(140, 3%, 26%, 0.4);
```


Сначала определите тень в виде шестнадцатеричного значения (в качестве резервного варианта для устаревших браузеров), а затем продублируйте правило, но уже с использованием значения HSLA или RGBA.

Пикселы, единицы em или rem

Вы также можете задавать значения для text-shadow в единицах em или rem. Вот, например, композиция сайта And the winner isn't... (рис. 6.1).



Рис. 6.1. Страница сайта And the winner isn't...

В Photoshop текст EVERY YEAR занимает в ширину 102 пиксела, а величина его тени равна 4 пикселам. Следовательно, используя нашу проверенную формулу *ширина цели × ширина контекста = результат* ($4 \times 102 = 0,039215686$), мы получим следующее:

```
text-shadow: .039215686em .039215686em 0em #dad7d7; /* 4 × 102 */
```

На рис. 6.2 можно увидеть, как результат будет выглядеть в браузере.

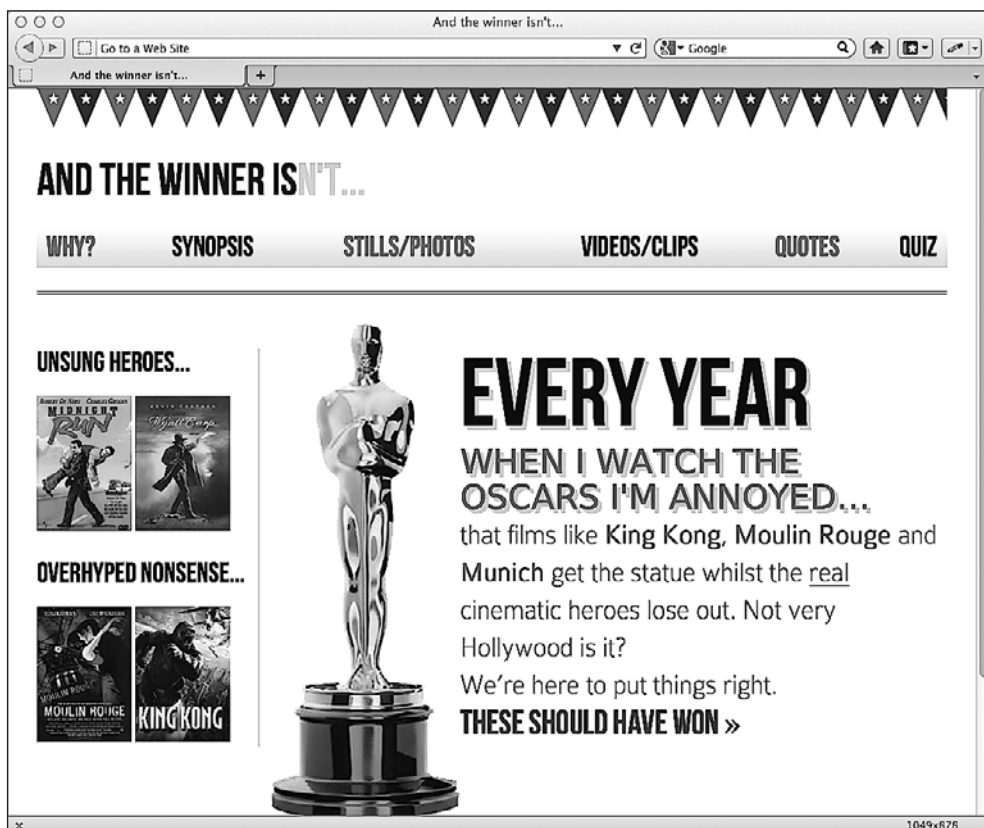


Рис. 6.2. У заголовка появилась тень

Я часто устанавливаю значения для `text-shadow` в единицах `em` или `rem`. Поскольку значения всегда очень маленькие, присвоение `1px` или `2px`, как правило, приводит к тому, что итоговый результат хорошо смотрится во всех областях просмотра.

Предотвращение отбрасывания текстом тени

В зависимости от того, какое у вас зрение, вы, *возможно*, заметили, что теперь у нас также появилась тень у текста во втором предложении `WHEN I WATCH THE OSCARS I'M ANNOYED` («Когда я смотрю церемонию вручения премий “Оскар”, меня возмущает...»). И вот причина этого:

```
<h1>Every year <em>when I watch the Oscars I'm annoyed...</em></h1>
```

На текущий момент `text-shadow` применяется ко всему тегу `<h1>` (который включает в себя тег ``), поэтому нам необходимо удалить `text-shadow` из тега ``:

```
#content h1 em {
    font-family: 'BitstreamVeraSansRoman';
    display: block;
    line-height: 1.052631579em; /* 40 × 38 */
    color: #757474;
    font-size: .352941176em; /* 36 × 102 */
    text-shadow: none;
}
```

Теперь соответствующий текст выглядит как надо (рис. 6.3).



Рис. 6.3. Тень осталась только для текста EVERY YEAR

Тени слева и сверху текста. Тени слева или сверху текста можно установить, задав отрицательные значения. Например:

```
text-shadow: -4px -4px 0px #dad7d7;
```

Этот код приведет к следующему результату (рис. 6.4).



Рис. 6.4. Тень теперь расположена левее и выше заголовка

Если для отбрасываемой текстом тени не будет задаваться размытие, то соответствующее значение можно не включать в объявление, например:

```
text-shadow: -4px -4px #dad7d7;
```

Согласно спецификации первые два значения являются величинами смещения, если не объявлено третье значение.

Создание эффекта рельефного текста

Я всегда считал, что свойство `text-shadow` наилучшим образом подходит для создания эффекта рельефного текста. Этот эффект обычно лучше всего работает при использовании яркого цвета (например, белого или близкого к нему), примененного к темному тексту на цветном фоне. Добавим эффект рельефного текста для навигационных ссылок:

```
nav ul li a {  
    height: 42px;  
    line-height: 42px;  
    text-decoration: none;  
    text-transform: uppercase;  
    font-family: 'BebasNeueRegular';  
    font-size: 1.875em; /*30 × 16 */  
    color: #000000;  
    text-shadow: 0 1px 0 hsla(0, 0%, 100%, 0.75);  
}
```

На рис. 6.5 показан результат. Едва уловимый, но эффектный — всего лишь добавилось немного глубины.



Рис. 6.5. Эффект рельефного текста для ссылок



СОВЕТ

Стремясь обеспечить наилучший эффект рельефного текста, я пришел к выводу, что значение 1px или 2px в качестве величины вертикального смещения и отсутствие размытия и горизонтального смещения наилучшим образом подходят для достижения поставленной цели.

Множественные тени, отбрасываемые текстом

Текст можно снабдить множественными тенями, разделив запятыми два значения. Например:

```
text-shadow: 0px 1px #ffffff, 4px 4px 0px #dad7d7;
```

Как и всегда, действовать нужно деликатно, иначе шрифт может стать неудобочитаемым. Я собираюсь использовать это объявление для комбинирования приводившегося ранее эффекта рельефного текста с уже имеющимся эффектом отбрасываемой текстом тени. На рис. 6.6 показано, как результат будет выглядеть в браузере.



Рис. 6.6. Для заголовка установлена множественная тень



ПРИМЕЧАНИЕ

Прочсть спецификацию W3C свойства `text-shadow` можно по следующему адресу: <http://www.w3.org/TR/css3-text/#text-shadow>.

6.2. Создание теней, отбрасываемых блочными элементами

Если вы разобрались, как создавать тени, отбрасываемые текстом, то легко поймете, как обеспечить отбрасывание теней блочными элементами. В принципе, для их создания потребуется придерживаться точно такого же синтаксиса: горизонтальное смещение, вертикальное смещение, размытие и цвет:

```
box-shadow: 0px 3px 5px #444444;
```

Однако они не очень хорошо поддерживаются браузерами, поэтому будет разумным использовать префиксы поставщиков, чтобы максимизировать совместимость. Например:

```
-ms-box-shadow: 0px 3px 5px #444444;  
-moz-box-shadow: 0px 3px 5px #444444;  
-webkit-box-shadow: 0px 3px 5px #444444;  
box-shadow: 0px 3px 5px #444444;
```

Используем свойство `box-shadow` для добавления теней к постерам фильмов во врезке на сайте *And the winner isn't...*:

```
.sideBlock img {  
    max-width: 45%;  
    box-shadow: 0px 3px 5px #444444;  
}
```

На рис. 6.7 можно увидеть, как результат будет выглядеть в браузере.



Рис. 6.7. Постеры теперь отбрасывают тень

Внутренняя тень

Свойство `box-shadow` также может использоваться для создания внутренней тени, которая применяется внутри целевого элемента, а не снаружи, как обычная тень, отбрасываемая блочным элементом. Такой подход позволяет создавать, например, эффект виньетки. Вот синтаксис:

```
box-shadow:inset 0 0 40px #000000;
```

Все работает как и раньше, однако слово `inset` дает браузеру команду применять эффект **внутри**. Сейчас я собираюсь использовать это правило в теге `<body>` для создания эффекта виньетки для всей страницы. Идея заключается в том, чтобы тень отображалась по всем краям страницы:

```
body {
  -moz-box-shadow:inset 0 0 30px #000000;
```

```

-webkit-box-shadow:inset 0 0 30px #000000;
box-shadow:inset 0 0 30px #000000;
}

```

На рис. 6.8 показано, как результат будет выглядеть в браузере.



Рис. 6.8. По периметру страницы отображается внутренняя тень

Множественные тени

Как и текст, блочные элементы могут иметь множественные тени. Опять-таки вам потребуется лишь разделить запятыми значения, и они будут применяться сверху вниз в том порядке, в каком указаны. Объявление, самое близкое к верхушке

в правиле (в коде), будет самым близким к верхушке порядка при визуализации в браузере.

```
box-shadow: inset 0 0 30px hsl(0, 0%, 0%),
            inset 0 0 70px hsla(0, 97%, 53%, 1);
```

Я добавил эти строки в свое правило `body` и получил в итоге ужасный эффект красного будуара (рис. 6.9). Должно быть, это побочный результат того, что на странице имеется постер фильма «Мулен Руж»!



Рис. 6.9. Тень слишком бросается в глаза

Достаточно сказать, что я решил выбросить из кода это объявление! Однако оно показывает, как благодаря CSS3 можно поэкспериментировать с реализацией раз-

ных идей в дизайне. На добавление и удаление визуальных «украшательств» вы потратите пару секунд, и не придется прибегать к графическим редакторам.



ПРИМЕЧАНИЕ

Прочитать спецификацию W3C свойства `box-shadow` можно по следующему адресу: <http://www.w3.org/TR/css3-background/#the-box-shadow>.

6.3. Фоновые градиенты

Если не применять CSS3, то, когда нам потребуется добавить для какого-либо элемента фоновый градиент, мы будем использовать небольшой графический фрагмент, который продублируем по горизонтали/вертикали. Поскольку серьезные графические ресурсы не задействуются, это довольно экономичный подход. Изображение шириной всего один или два пиксела почти не повысит требования к скорости канала подключения пользователя и может задействоваться для множественных элементов в рамках одного сайта.

Линейные фоновые градиенты

Применим этот метод на практике и создадим линейный фоновый градиент для врезки на сайте And the winner isn't...:

```
aside {
    border-right-color: #e8e8e8;
    border-right-style: solid;
    border-right-width: 2px;
    margin-top: 58px;
    padding-left: 1.5%;
    padding-right: 1.0416667%;
    margin-left: 1.0416667%;
    float: left;
    width: 20.7083333%;
    background: url(../img/sidebarBg2.png) 50% repeat-x;
}
```

На рис. 6.10 показано, как результат будет выглядеть в браузере.

Однако нам все же придется прибегнуть к графическому редактору, если мы захотим изменить эффект. Кроме того, содержимое иногда может выступать за пределы градиентного фона, не вписываясь в ограничения в виде фиксированных размеров. Эта проблема усугубляется при работе с адаптивным веб-дизайном, поскольку мы хотим, чтобы структура страницы могла значительно изменять свою форму (например, становиться длиннее или шире), не нарушая при этом дизайн.

Допустим, я решил добавить постеры двух других фильмов в каждый раздел. Вот что получится в итоге (рис. 6.11).



Рис. 6.10. У врезки появился градиентный фон



Рис. 6.11. Размеры врезки увеличились

Ничего ужасного, однако серый градиент не простирается на всю врезку, как мне того хотелось бы. Обычно для решения подобных проблем я прибегаю к графическому редактору и переделываю изображение. Однако в случае с CSS3-градиентом

есть более гибкое решение. Вот синтаксис для обеспечения аналогичного градиента исключительно с использованием CSS3 вместо изображения:

```
background: linear-gradient(90deg, #ffffff 0%, #e4e4e4 50%,  
                                #ffffff 100%);
```

На рис. 6.12 показано, как результат будет выглядеть в совместимом браузере.



Рис. 6.12. Фоновый градиент растягивается на всю длину врезки

Независимо от того, насколько длинным окажется раздел (ведь существует множество фильмов, которые в равной мере могут вызывать у меня восторг или неприязнь), CSS3-градиент всегда будет простирается на всю соответствующую область.

Единственная большая ложка дегтя в бочке меда фоновых градиентов состоит в том, что они не так хорошо поддерживаются, как некоторые другие параметры CSS3. Например, Internet Explorer 9 не воспринимает их (однако это обещают устранить в Internet Explorer 10). Тем не менее фоновые градиенты поддерживаются в большинстве прочих браузеров, хотя и с префиксами поставщиков. Это не должно служить поводом для вашего отказа от их использования. В качестве резервного варианта для устаревших браузеров иногда желательно определять сначала сплошной цвет фона, чтобы они смогли по крайней мере визуализировать однотонный фон, если градиентные правила окажутся им непонятны.



РАНЬШЕ СИНТАКСИСЫ ФОНОВОГО ГРАДИЕНТА БЫЛИ РАЗНЫМИ

Исторически сложилось так, что разные поставщики браузеров задействовали множество отличающихся синтаксисов для визуализации одного и того же эффекта фонового градиента. Главным «правонарушителем» был WebKit, однако, к счастью, начиная с версии Safari 5.1, они стали руководствоваться теми же соглашениями, что и Mozilla, — соглашениями, которые также использует W3C.

Анализ синтаксиса линейного градиента. Синтаксис линейного фонового градиента может привести вас в замешательство, поэтому проанализируем его.

- Первое (опциональное) значение (в данном случае это 90deg) в скобках определяет направление, в котором будет начинаться градиент. Если не указать это значение, то по умолчанию будет задан градиент, начинающийся сверху и далее вниз. Вы также можете использовать другое значение, например to top right, которое позволит обеспечить диагональный градиент, заканчивающийся сверху справа.
- Следующее значение (#ffffff 0% в данном примере) представляет собой «отправную точку» — значение цвета, заданное как цвет плюс позиция. Вы также могли бы указать что-то вроде blue 20%, в результате чего цвет постепенно переходил бы от синего к следующему цвету на уровне 20 % вдоль воображаемой линии от начала до конца линейного градиента. Равным образом можно было бы задать отрицательное значение позиции, чтобы градиент начинался до того, как он будет фактически видим. Например:

```
background: linear-gradient(90deg, #ffffff -50%, #e4e4e4 50%,  
                             #ffffff 100%);
```

Эта строка означает, что градиент будет начинаться за 50 % до начала видимой области, вдоль которой проходит воображаемая линия.

- Следующее значение — это «остановка цвета». Кратко взглянем, где мы находимся: в нашем примере мы движемся по направлению вверх под углом 90°

(90deg), начиная с белого цвета (#ffffff 0%) и далее к значению цвета #e4e4e4 (светло-серый) на уровне 50 % вдоль линии. Это наша первая «остановка цвета» в градиенте. Если потребуется, мы можем задействовать множественные «остановки цветов» (разделенные запятыми), прежде чем определим нашу «конечную точку».

- Последнее значение в скобках (#ffffff 100% в нашем примере) всегда является «конечной точкой» градиента. Независимо от того, сколько «остановок цветов» будет после «отправной точки», последнее значение всегда будет представлять собой «конечную точку».



ПРИМЕЧАНИЕ

Прочсть спецификацию W3C линейных фоновых градиентов можно по следующему адресу: <http://dev.w3.org/csswg/css3-images/#linear-gradients>.

Радиальные фоновые градиенты

Фоновые градиенты CSS3 не ограничиваются линейными. С той же легкостью можно создавать и радиальные градиенты. Они начинаются с «центральной точки» и плавно расходятся в стороны эллиптически или кругообразно.

Вот синтаксис радиального фонового градиента:

```
background: radial-gradient(center, ellipse cover, #ffffff 72%,  
                           #dddddd 100%);
```

Если добавить это объявление в наше правило #content, то результат будет следующим (рис. 6.13).



Рис. 6.13. Пример радиального градиента на странице

Видите небольшое потемнение в углах? Это наш радиальный градиент. Проанализируем его синтаксис и разберемся, что к чему.

Анализ синтаксиса радиального градиента. Задав соответствующее свойство (`background:`), мы указываем, что нам требуется радиальный градиент (а не линейный). Затем мы указываем в скобках «начальную точку». В предыдущем примере мы использовали `center`, однако точно так же могли задействовать что-то вроде `25px 25px`, чтобы радиальный градиент начинал расходиться в 25 пикселях от верха элемента с его левой стороны. Например:

```
background: radial-gradient(25px 25px, ellipse cover, #ffffff 72%,  
                           #dddddd 100%);
```

Эта строка кода дает следующий результат (рис. 6.14).



Рис. 6.14. Центр находится в 25 пикселях от верха элемента с его левой стороны, и от центра плавно расходится радиальный градиент

Следующее значение в нашем объявлении легче понять — это форма и размеры, которые должны быть у радиального градиента:

```
background: radial-gradient(center, ellipse cover, #ffffff 72%,  
                           #dddddd 100%);
```

Форму радиального градиента можно задать, выбрав один из двух вариантов: либо `circle` (радиальный градиент будет равномерно расходиться от центра во всех направлениях), либо `ellipse` (радиальный градиент будет неравномерно расходиться от центра в разных направлениях). При определении размеров радиального градиента можно проявлять немалую гибкость. Вы можете задать его размеры с помощью одного из следующих свойств:

- `closest-side` — градиент будет расходиться от центра до ближайшей стороны блочного элемента (в случае с `circle`) либо до обеих сторон — вертикальной и горизонтальной, — которые являются ближайшими к центру (в случае с `ellipse`);

- `closest-corner` — радиальный градиент будет расходиться от центра вплоть до ближайшего угла блочного элемента;
- `farthest-side` — противоположность `closest-side` в том, что вместо того, чтобы расходиться до ближайшей стороны блочного элемента, радиальный градиент будет расходиться от центра до его наиболее удаленной стороны (либо до обеих наиболее удаленных от центра сторон — вертикальной и горизонтальной — в случае с `ellipse`);
- `farthest-corner` — радиальный градиент будет расходиться от центра блочного элемента до его самого дальнего угла;
- `cover` — значение идентично `farthest-corner`;
- `contain` — идентично `closest-side`.

Затем остается лишь определить «отправную точку», «остановки цветов» и «конечную точку» (таким же образом, как и для линейных градиентов).

Например, мы можем изменить наше правило следующим образом:

```
background: radial-gradient(20px 20px, circle cover,  
                           hsla(9.69%,85%,0.5) 0%,  
                           hsla(9.76%,63%,1) 50%,  
                           hsla(10.98%,46%,1) 51%,  
                           hsla(24,100%,50%,1) 75%,  
                           hsla(10,100%,39%,1) 100%);
```

Мы начинаем на расстоянии 20 пикселей от верха элемента с его левой стороны, используя значение `circle` в качестве формы радиального градиента и множественные «остановки цветов» HSL(A). Результат показан на рис. 6.15.



Рис. 6.15. Пример необычного радиального градиента

Хоть это был и не самый лучший урок по эстетике, я все же надеюсь, что он продемонстрировал вам мощь использования CSS3 для обеспечения требуемых визуальных эффектов.

**ПРИМЕЧАНИЕ**

Прочсть спецификацию W3C радиальных фоновых градиентов можно по следующему адресу: <http://dev.w3.org/csswg/css3-images/#radial-gradients>.

**КОРОТКИЙ ПУТЬ К ИДЕАЛЬНЫМ ЛИНЕЙНЫМ И РАДИАЛЬНЫМ ГРАДИЕНТАМ CSS3**

Если создание CSS3-градиентов покажется вам тяжелой работой, то знайте, что существует несколько отличных онлайн-инструментов для генерирования градиентов. Я предпочитаю тот, что доступен по адресу <http://www.colorzilla.com/gradient-editor/>. Его интерфейс такой же, как у графических редакторов. Вы сможете выбирать требуемые цвета, «остановки», стиль градиента (поддерживаются линейные и радиальные градиенты) и даже цветовое пространство (HEX, RGB(A), HSL(A)) для вашего итогового градиента. Есть также множество заранее подготовленных градиентов, которые можно использовать в качестве отправной точки. Если этого вам покажется мало, то инструмент даже предоставит вам опциональный код, позволяющий сделать так, чтобы Internet Explorer 9 смог отобразить соответствующий градиент, а также поможет обеспечить сплошной цвет в качестве резервного варианта для устаревших браузеров. Мне все еще не удалось вас убедить? Как насчет возможности генерировать градиенты на основе имеющихся изображений? Думаю, она поспособствует тому, что вы решите воспользоваться этим инструментом.

Повторяющиеся градиенты

CSS3 также позволяет создавать повторяющиеся фоновые градиенты. Посмотрим, как это делается:

```
background: repeating-linear-gradient(90deg, #ffffff 0px,  
    hsla(0, 1%, 50%, 0.1) 5px);
```

Результат применения этого градиента к нашей врезке показан на рис. 6.16.

Прежде всего необходимо добавить префикс `repeating` к свойству `linear-gradient` или `radial-gradient`, а затем будет следовать тот же синтаксис, что и для обычного градиента. Здесь я использовал интервалы в пикселах между белым и серым цветами (соответственно 0px и 5px), однако их также можно было бы указать в процентах. Для получения наилучших результатов рекомендуется придерживаться одних и тех же единиц измерения (например, пикселей или процентов) в рамках синтаксиса того или иного градиента.

Опробуем повторяющийся радиальный градиент:

```
background: repeating-radial-gradient(2px 2px, ellipse,  
    hsla(0,0%,100%,1) 2px, hsla(0,0%,95%,1) 10px,  
    hsla(0,0%,93%,1) 15px, hsla(0,0%,100%,1) 20px);
```

Он очень схож со стандартным радиальным градиентом, который мы использовали ранее. Я лишь изменил «отправную точку», удалил значение `cover`, поскольку оно больше не требуется, а затем задал интервалы для каждой «остановки цвета» в пикселах. Моя «конечная точка» — 20px, поэтому узор повторяется каждые 20 пикселей. Чуть ниже приведен результат применения этого правила к разделу `body`. Обращаю ваше внимание — он некрасиво смотрится (рис. 6.17)!



Рис. 6.16. У врезки теперь есть фоновый повторяющийся линейный градиент



ПРИМЕЧАНИЕ

Прочитать спецификацию W3C повторяющихся градиентов можно по следующему адресу: <http://dev.w3.org/csswg/css3-images/#repeating-gradients>.

Есть еще один способ использования фоновых градиентов, которым я хочу поделиться с вами.



Рис. 6.17. Пример повторяющегося радиального градиента, примененного ко всей странице

6.4. Фоновые градиентные узоры

Несмотря на то что мне часто доводилось использовать едва уловимые линейные градиенты в дизайнах, с применением радиальных и повторяющихся градиентов в своей практике я сталкивался реже. Тем не менее на основе всех этих методик обеспечения фона умные люди создали фоновые градиентные узоры. Рассмотрим пример. Вместо повторяющегося радиального градиента, который я недавно добавлял в раздел `body`, я использую следующее:

```
body {
  background-color:white;
  background-image:
    radial-gradient(hsla(0, 0%, 87%, 0.31) 9px, transparent 10px),
    repeating-radial-gradient(hsla(0, 0%, 87%, 0.31) 0,
```

```

        hsla(0, 0%, 87%, 0.31) 4px, transparent 5px,
        transparent 20px, hsla(0, 0%, 87%, 0.31) 21px,
        hsla(0, 0%, 87%, 0.31) 25px, transparent 26px,
        transparent 50px);
background-size: 30px 30px, 90px 90px;
background-position: 0 0;
}

```

На рис. 6.18 показано, как результат будет выглядеть в браузере.



Рис. 6.18. Для страницы задан фон в виде узора

Ну как? Всего лишь несколько строк CSS3-кода, и у нас имеется легко редактируемый, масштабируемый фоновый узор, созданный с использованием методик обеспечения градиентного фона, которые мы уже рассматривали ранее.

Специалист по CSS Лиа Веру (Lea Verou) создала постоянно пополняющийся ресурс фоновых узоров CSS, доступный по адресу <http://lea.verou.me/css3patterns/> (рис. 6.19).

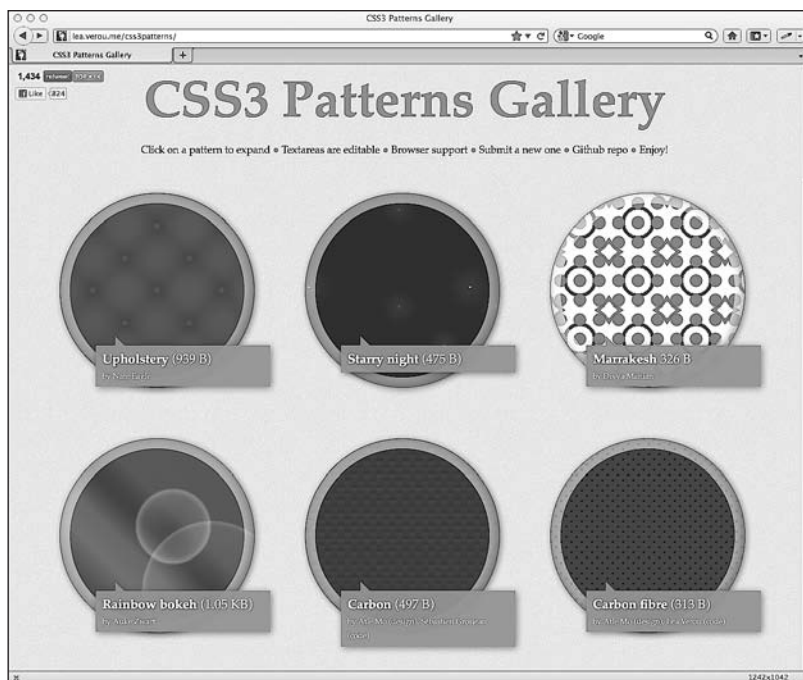


Рис. 6.19. Сайт CSS Patterns Gallery

6.5. Кое-какие соображения насчет CSS3

Следует помнить, что разные объявления можно использовать для различных по размеру областей просмотра. Например, я ничего не имею против того, как описанный выше градиентный узор выглядит в меньших областях просмотра (рис. 6.20).

Однако я могу предпочесть не использовать его для более крупных областей просмотра (например, шириной 768 пикселей и более). Следовательно, можно просто создать специфическое правило для обеспечения фонового градиента с использованием медиазапросов:

```
@media screen and (max-width: 768px) {
  body {
    background-color:white;
    background-image:
      radial-gradient(hsla(0, 0%, 87%, 0.31) 9px, transparent 10px),
      repeating-radial-gradient(hsla(0, 0%, 87%, 0.31) 0,
        hsla(0, 0%, 87%, 0.31) 4px, transparent 5px, transparent 20px,
        hsla(0, 0%, 87%, 0.31) 21px, hsla(0, 0%, 87%, 0.31) 25px,
        transparent 26px, transparent 50px);
    background-size: 30px 30px, 90px 90px;
    background-position: 0 0;
  }
}
```



Рис. 6.20. Фон в виде узора на странице, открытой в небольшой области просмотра

Помните, что медиазапросы при необходимости позволят вам по-разному определять каждый элемент для разных по размеру областей просмотра. Все это направлено на обеспечение наиболее оптимального взаимодействия для пользователей.



ЛЕГКОЕ СОЗДАНИЕ CSS3-КОДА С ПОМОЩЬЮ ПРЕПРОЦЕССОРОВ CSS

В настоящее время CSS3-правилам требуются свойства с разными префиксами поставщиков. Вместо того чтобы сохранять вырезанные сегменты кода с этими префиксами для каждого объявления или использовать JavaScript-файл для добавления префиксов в браузер, можно задействовать препроцессоры CSS вроде SASS и LESS. Например, используя SASS с плагином Compass, вы сможете написать простое правило `box-shadow` вроде следующего: элемент `{ @include box-shadow; }`. Сгенерированный CSS-код будет включать полный набор определяемых поставщиками правил наряду с соответствующими исправлениями для Internet Explorer (при наличии таковых).

Если этого недостаточно для того, чтобы вы обратили на них внимание, то замечу, что препроцессоры также позволяют использовать переменные и относящиеся к программированию соглашения вроде операторов `if/while`. Подробнее о SASS вы сможете узнать на сайте <http://sass-lang.com>, а о LESS — по адресу <http://lesscss.org>.

6.6. Сводим воедино CSS3-свойства

До сих пор мы в основном рассматривали абстрактные реализации разнообразных CSS3-свойств. Теперь используем их для создания ссылки **THESE SHOULD HAVE WON** («Вот они должны были стать победителями»). В Photoshop-файле оригинальной композиции сайта And the winner isn't... для оформления текста применяется пользовательская типографика, которую мы уже обсуждали в главе 5. При этом у данной ссылки в оригинальной композиции также имеется красный градиентный фон, скругленные углы и тень. Вот что определено в нашей таблице стилей на текущий момент:

```
#content a {  
    text-decoration: none;  
    font: 2.25em /* 36px × 16 */ 'BebasNeueRegular';  
}
```

Сначала добавим сплошной цвет фона в качестве резервного варианта для устаревших браузеров. Таким образом, если они не сумеют визуализировать градиент, то по крайней мере смогут отобразить однотонный фон красного цвета. Я намеренно использовал здесь шестнадцатеричное значение, ведь если конкретному устаревшему браузеру непонятны градиенты, то вряд ли он поддерживает цветовые режимы RGB и HSL:

```
#content a {  
    text-decoration: none;  
    font: 2.25em /* 36px × 16 */ 'BebasNeueRegular';  
    background-color: #b01c20;  
}
```

Далее обеспечим скругление углов. Заметьте, что, как и в остальной части текущей главы, для всех CSS3-свойств, которые я буду добавлять, может потребоваться определить префиксы поставщиков. Я не стал включать их в приведенный здесь пример ради краткости:

```
#content a {  
    text-decoration: none;  
    font: 2.25em /* 36px × 16 */ 'BebasNeueRegular';  
    background-color: #b01c20;  
    border-radius: 8px;  
}
```

Вот что у нас имеется на данном этапе (рис. 6.21).



Рис. 6.21. Для ссылки заданы красный фон и скругленные углы

Теперь сделаем текст белым (опять-таки, поскольку мне нужно, чтобы он был видим и в устаревших браузерах, я придерживаюсь простого определения цвета) и добавим отступ (его также можно было бы указать в процентах), чтобы вокруг текста всегда было небольшое пространство:

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px × 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  padding: 30px;
}
```

Вот что у нас получилось в результате (рис. 6.22).

На текущем этапе отступ «посягает» на территорию текста сверху, поэтому наряду с градиентом добавим объявление `float: left`:

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px × 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  padding: 30px;
  float: left;
  background: linear-gradient(90deg, #b01c20 0%, #f15c60 100%);
}
```

Ссылка все больше приобретает требуемый вид (рис. 6.23).



Рис. 6.22. Для текста ссылки задан белый цвет

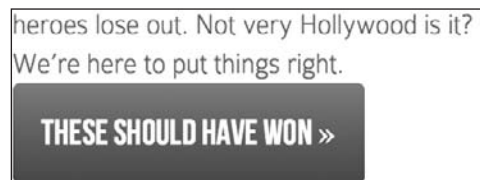


Рис. 6.23. Для фона ссылки установлен линейный градиент

Помимо добавления небольшого отступа сверху, я задействую свойство `box-shadow`:

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px × 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  padding: 30px;
  float: left;
  background: -moz-linear-gradient(90deg, #b01c20 0%,
                                     #f15c60 100%);
  margin-top: 30px;
  box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
}
```

Быстрая проверка в браузере показывает, что мы почти достигли цели (рис. 6.24).

Теперь, несмотря на то что этого нет в нашем Photoshop-файле, я собираюсь добавить небольшую тень к тексту и тонкую белую границу, чтобы ссылка-кнопка казалась слегка выпуклой. В этом и состоит вся прелесть использования CSS вместо файлов изображений — можно легко оценивать изменения на лету!

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px×16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  padding: 30px;
  float: left;
  background: -moz-linear-gradient(90deg, #b01c20 0%,
                                     #f15c60 100%);

  margin-top: 30px;
  box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
  text-shadow: 0px 1px black;
  border: 1px solid #bfbfbf;
}
```

Вот как теперь наша ссылка будет выглядеть в Firefox 8 (рис. 6.25).



Рис. 6.24. Ссылка практически приобрела требуемый вид



Рис. 6.25. Ссылка-кнопка стала выпуклой

Единственная оставшаяся проблема заключается в том, что шрифт символа двойной угловой кавычки (» в HTML) в нашем Photoshop-файле отличается от шрифта основного текста. Не думаю, что есть смысл загружать дополнительный шрифт ради одного символа, поэтому я собираюсь заключить этот символ в строчный тег, чтобы можно было увеличить его размер. Вот как будет выглядеть измененная разметка:

```
<a href="#">these should have won <span>&raquo;</span></a>
```

Кроме того, добавим CSS-правило для корректировки размера шрифта:

```
#content a span {
  font-size: 1.3em;
}
```

Это финальный штрих, позволяющий добиться безупречного внешнего вида (рис. 6.26).



Рис. 6.26. Ссылка приобрела надлежащий вид

Замечательной особенностью этой конструкции, созданной с использованием CSS3, а не изображения, является то, что она может включать любое содержимое, которое потребуется, и никогда при этом не «развалится» (рис. 6.27).



Рис. 6.27. При изменении текста ссылка не теряет своего оформления

6.7. Множественные фоновые изображения

Нередко заказчики просят создать веб-страницу, вверху которой будет фоновое изображение, отличающееся от того, что расположено внизу. Или, возможно, разные изображения для верха и низа области содержимого в рамках страницы. Это требование кажется таким простым, что по вполне понятным причинам можно предположить, что его удастся выполнить с помощью CSS. Однако при использовании версии CSS 2.1 решение подобной задачи обычно требовало дополнительной разметки. Например, вот как я всегда выполнял такое требование до появления CSS3:

```
<body class="headerBackgroundHere">
<div class="footerBackground">
  <div id="container">
    <header>
      // Здесь будет содержимое верхнего колонтитула
    </header>
    <div id="main" role="main">
      // Здесь будет основное содержимое
    </div>
    <footer>
      // Здесь будет содержимое нижнего колонтитула
    </footer>
  </div>
</div> <!--! конец .footerBackground -->
</body>
```

Заметьте, что контейнер всего содержимого (здесь это `<div>` с идентификатором `container`), в свою очередь, заключен в элемент `<div>` с классом `footerBackground`. В данной ситуации мы можем нацелить CSS-правило для задания фонового изображения для верха страницы в теге `<body>`:

```
body {
  background-image: url("../img/topSlice.png");
  background-repeat: repeat-x;
}
```

А затем еще одно правило, но уже для `footerBackground`. Здесь мы разместим нужное изображение для низа страницы.

```
.footerBackground {
  background-image: url("../img/bottomSlice.png");
  background-repeat: repeat-x;
  background-position: bottom;
}
```

Подобная методика хорошо и последовательно работает в большинстве браузеров. Однако я никогда не был сторонником использования дополнительной разметки всего лишь для решения проблем, связанных с внешним видом.

К счастью, эту задачу легко решить с помощью CSS3, поскольку в новой версии можно задавать множественные фоновые изображения для того или иного элемента (часть **CSS Backgrounds and Borders Module Level 3**). Они хорошо поддерживаются браузерами, примечательным исключением среди которых является Internet Explorer версии 8 и ниже. Вот синтаксис:

```
background:
  url('../img/1.png'),
  url('../img/2.png'),
  url('../img/3.png');
```

Как и в случае с порядком расположения множественных теней, изображение, указанное первым, будет ближайшим к «верхушке» в браузере. Если пожелаете, можете добавить значение общего цвета фона в то же объявление, как показано далее:

```
background:
  url('../img/1.png'),
  url('../img/2.png'),
  url('../img/3.png') left bottom, black;
```

Значение цвета указывайте последним, чтобы оно шло после всех изображений, перечисленных выше.

Браузеры, которые не смогут понять правило для задания множественных фоновых изображений (например, Internet Explorer версии 8 и ниже), будут полностью игнорировать его. Возможно, вы захотите объявить «обычное» свойство background непосредственно перед таким CSS3-правилом в качестве резервного варианта для устаревших браузеров.

Если вы задействуете PNG-файлы с прозрачностью, то через все частично прозрачные фоновые изображения, размещенные одно над другим, будут просматриваться рисунки, расположенные ниже. Однако не обязательно, чтобы фоновые изображения размещались одно над другим и все были одинакового размера.

Размеры фоновых изображений

Для задания размеров для каждого изображения применяйте свойство background-size. При использовании множественных фоновых изображений соответствующий синтаксис будет выглядеть так:

```
background-size: 100% 50%, 300px 400px, auto;
```

Значения размеров (сначала ширины, а затем высоты) для каждого изображения объявляются, разделяются запятыми в том порядке, в каком они указаны в свойстве background. Как и в приведенном выше примере, вы можете использовать значения в процентах или пикселах для каждого изображения наряду со следующим:

- auto — задает для элемента его исходные размеры;
- cover — растягивает изображение с сохранением соотношения его сторон, чтобы покрыть область соответствующего элемента;
- contain — растягивает изображение, подгоняя его самую длинную сторону под размеры элемента, в котором оно находится, сохраняя соотношение сторон рисунка.

Позиционирование фоновых изображений

CSS3 позволяет также указать разные позиции для различных изображений. Для этого мы могли бы изменить наше правило следующим образом:

```
background:
  url('../img/1.png') center,
  url('../img/2.png'),
  url('../img/3.png') left bottom, black;
```

Там, где позиция не объявляется, например, для второго изображения, по умолчанию будет устанавливаться месторасположение вверху слева.

Сокращенный способ определения фона

Есть сокращенный способ комбинирования свойств, связанных с фоновыми изображениями. Однако по своему опыту могу сказать, что он дает изменчивые результаты. По этой причине я склонен использовать обычный способ, объявляя сначала множественные изображения, затем размеры, а после этого позиции.



ПРИМЕЧАНИЕ

Прочсть документацию W3C, касающуюся множественных фоновых элементов, можно по адресу <http://www.w3.org/TR/css3-background/#layering>.

О задании размеров для фоновых изображений можно прочсть здесь: <http://www.w3.org/TR/css3-background/#the-background-size>.

О позиционировании фоновых изображений вы прочтете по адресу <http://www.w3.org/TR/css3-background/#the-background-position>.

6.8. Прочие CSS3-свойства

Мы рассмотрели далеко не все замечательные свойства, которые может предложить CSS3. Однако те из них, что мы успели охватить, скорее всего, будут наиболее часто применяться вами. Как мне кажется, эти инструменты лучше всего подойдут, если вам потребуется снабдить адаптивные веб-дизайны экономичными и гибкими визуальными эффектами. Однако, как и всегда, не спускайте глаз с различных модулей CSS3, поскольку в них непременно найдется то, что разожжет у вас интерес к вещам, которые мы здесь не рассматривали.

6.9. Масштабируемые значки, идеально подходящие для адаптивных веб-дизайнов

Умные люди уже занимаются расширением списка возможностей, которые предоставляет CSS3. Мне довелось увидеть реализацию одной интересной методики, и теперь я сам регулярно применяю ее. Она заключается в использовании значков, устанавливаемых в веб-дизайнах правилом `@font-face`.

«А что они собой представляют?» — слышу, как вы громко спрашиваете. Что ж, я все вам расскажу. Помните, как в предыдущей главе мы использовали CSS3-правило `@font-face` для добавления пользовательской типографики в наш веб-дизайн? Значки, устанавливаемые правилом `@font-face`, — это всего лишь шрифты, специально предназначенные для создания широко используемых значков. Вместо того чтобы применять множество отдельных графических файлов для каждого значка или даже объединять их в одно более крупное по размеру спрайтовое изображение, значки из правила `@font-face` позволят вам применять отдельный шрифт для каждого включенного значка (а это подразумевает лишь один запрос HTTP — здорово!). Более того, поскольку это шрифт, он превосходно масштабируется, что идеально подходит для адаптивных веб-дизайнов. Отличным примером является сайт Fico, доступный по адресу <http://fico.lensco.be/>.



Рис. 6.28. Главная страница сайта Fico

6.10. Резюме

В этой главе мы использовали более широкий набор новых CSS3-свойств. Фоновые градиенты CSS3 дали нам возможность создавать интересные эффекты, применяя исключительно CSS3. Мы даже задействовали их для создания фоновых

узов. Мы также научились применять свойство `text-shadow` для создания эффекта рельефного текста и `box-shadow` для добавления эффекта отбрасываемой тени снаружи и внутри элементов.

При использовании методик адаптивного веб-дизайна создание эстетических эффектов с применением исключительно **CSS3 будет нести в себе огромное преимущество**. Однако бывают ситуации, когда приходится использовать изображения. Хотя CSS3 и здесь позволяет нам проявлять большую гибкость. Например, в этой главе мы рассмотрели CSS3-методику, позволяющую задавать множественные фоновые изображения, чтобы добавить разнообразные фоны и обеспечить их независимое позиционирование на странице. Такая методика сводит на нет необходимость в дополнительной разметке, что ранее было неизбежным. По большей части мы используем все эти эффекты для добавления в адаптивные веб-дизайны визуальных «украшательств», относящихся к тому виду деталей, которыми смогут насладиться лишь пользователи современных браузеров независимо от размера области просмотра. Хотя наскучившие устаревшие браузеры вроде **Internet Explorer** не способны визуализировать такие визуальные эффекты, они никак не влияют на внешний вид адаптивных веб-страниц.

Однако до сих пор элементы во всех наших примерах **CSS3-кода были статичными**; они располагались на одном месте и оставались неизменными на страницах в том или ином состоянии. Но CSS3 позволяет сделать намного больше. В следующей главе мы рассмотрим способы обеспечения перехода из одного состояния в другое, а также направим наш CSS-код туда, где он никогда не использовался прежде, — в область анимаций.

7 CSS3-переходы, трансформации и анимации

В двух предыдущих главах мы рассмотрели некоторые новые свойства и возможности CSS3. Однако все, что мы до сих пор рассматривали, было статичным, а версия CSS3 способна на большее.

В настоящее время, если вам потребуется анимировать элементы на веб-странице, то вы, скорее всего, либо напишете собственный JavaScript-код, обеспечивающий выполнение требуемого действия, либо обратитесь к популярной JavaScript-библиотеке вроде jQuery, которой поручите всю тяжелую работу. Однако разработчики, увлекающиеся CSS3, явно не согласны с засильем JavaScript в этой области и надеются, что ситуация вскоре изменится. Несмотря на то что версия CSS3 вряд ли узурпирует «власть» jQuery или чего-то в этом роде в ближайшее время, она идеально подходит для обеспечения плавных переходов (например, при наведении указателя мыши) и перемещения элементов по экрану. Это отличная новость, поскольку она означает, что при увеличении количества замечательных современных браузеров, устанавливаемых на устройствах (например, на новейших смартфонах), мы можем использовать CSS для обеспечения анимаций вместо того, чтобы задействовать для этого JavaScript. Вывод: вы, надо полагать, можете вычеркнуть пункт «научиться анимировать элементы с помощью jQuery» из своего списка текущих дел, поскольку уже знаете, что это можно сделать, используя только CSS. Как и всегда, CSS3-свойства не приведут к каким-либо отрицательным последствиям в устаревших браузерах. Такие браузеры будут просто пропускать непонятные им правила, как будто тех и нет вовсе.

В этой главе мы поговорим о следующем:

- что такое CSS3-переходы и как их можно использовать;
- как писать CSS3-переходы и их сокращенный синтаксис;
- как применять временные функции для контроля над плавностью выполнения CSS3-переходов (ease, cubic-bezier и др.);
- как устанавливать занятые переходы для адаптивных сайтов;
- что такое CSS3-трансформации и как мы можем их использовать;
- как добавлять разные 2D-трансформации (scale, rotate, skew, translate и др.);
- что такое 3D-трансформации и как их обеспечить;
- как добавить анимацию с помощью CSS3 (с использованием @keyframes).

7.1. Что такое CSS3-переходы и как мы можем их использовать

Обычно при стилизации гиперссылок с использованием CSS обеспечивают для них выделенное состояние, в котором они будут пребывать при наведении на них указателя мыши. Это позволит ясно дать понять пользователям, что элемент, на который они наводят указатель, представляет собой ссылку. Такие состояния менее актуальны для растущего количества пользователей устройств с сенсорными экранами, однако для всех остальных они являются примером отличного и простого взаимодействия между сайтом и пользователем.

По традиции, если используется только CSS, то **выделенные состояния работают по принципу «включено/выключено»**. Одно состояние принято по умолчанию и моментально сменяется на другое при наведении указателя мыши. Однако CSS3-переходы, как следует из их названия, позволяют обеспечивать определенный переход из одного состояния в другое.

В предыдущей главе мы создали с помощью CSS3 ссылку-кнопку с красным градиентным фоном. Вот CSS3-код, который мы использовали (ради краткости из него убраны дополнительные префиксы поставщиков):

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px x 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: #ffffff;
  padding: 3%;
  float: left;
  background: linear-gradient(90deg, #b01c20 0%, #f15c60 100%);
  margin-top: 30px;
  box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
  text-shadow: 0px 1px black;
  border: 1px solid #bfbfbf;
}
```

Добавим выделенное состояние:

```
#content a:hover {
  border: 1px solid #000000;
  color: #000000;
  text-shadow: 0px 1px white;
}
```

На рис. 7.1 показано первое состояние, которое определено по умолчанию.

А вот выделенное состояние при наведенном указателе мыши (рис. 7.2).

При наведении указателя мыши просто изменяется текст, отбрасываемая им тень и цвет границы. Таким образом, как вы, возможно, уже догадались, при использовании текущего CSS-кода наведение указателя мыши будет приводить к моментальному переходу из первого состояния (белый текст) во второе (черный

текст). Это работа по принципу «включено/выключено». Добавим немного CSS3-магии в наше первое правило:

```
#content a {  
    /*...имеющиеся стили...*/  
    transition: all 1s ease 0s;  
}
```



Рис. 7.1. Неактивная ссылка



Рис. 7.2. Ссылка при наведенном указателе мыши

Теперь, если навести указатель мыши на ссылку-кнопку, то текст, отбрасываемая им тень и цвет границы будут плавно переходить из одного состояния в другое. Вы заметите, что к исходному элементу применяется переход, а не выделенное состояние. Это установлено для того, чтобы при других состояниях, например при активном, когда пользователь щелкает на ссылке кнопкой мыши, также могли задаваться другие стили и выполняться переход. Поэтому помните, что объявление перехода необходимо добавлять для элемента в том состоянии, из которого будет выполняться переход. Но как фактически работают переходы?

Свойства, используемые для объявления переходов

Переход можно объявить, используя четыре свойства или одно сокращенное объявление, включающее значения всех этих свойств:

- `transition-property` — имя CSS-свойства, к которому будет применен эффект перехода (например, `background-color`, `text-shadow` или `all` для применения эффекта перехода ко всем свойствам);
- `transition-duration` — отрезок времени, в течение которого должен осуществляться переход (указывается в секундах, например `0.3s`, `2s` или `1.5s`);

- `transition-timing-function` — определяет, как будет изменяться скорость перехода в течение заданного отрезка времени (например, можно указать значение `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out` или `cubic-bezier`);
- `transition-delay` — опциональное свойство для определения задержки перед началом выполнения перехода. Как вариант, для него можно задать отрицательное значение, чтобы переход начинался незамедлительно, однако при этом часть процесса перехода окажется пропущенной.

Используя по отдельности разные свойства для объявления переходов, можно создать такой переход:

```
#content a {  
    ... (еще стили) ...  
    transition-property: all;  
    transition-duration: 1s;  
    transition-timing-function: ease;  
    transition-delay: 0s;  
}
```

Собирательное свойство `transition`

Кроме того, отдельные объявления можно объединить в одну сокращенную версию:

```
transition: all 1s ease 0s;
```

При написании сокращенной версии важно помнить, что первым значением времени задается `transition-duration`, а вторым — `transition-delay`.

Как и всегда, важно использовать префиксы поставщиков. Например, набор вариантов с префиксами поставщиков предыдущего сокращенного объявления выглядел бы следующим образом:

```
-o-transition: all 1s ease 0s;  
-ms-transition: all 1s ease 0s;  
-moz-transition: all 1s ease 0s;  
-webkit-transition: all 1s ease 0s;  
transition: all 1s ease 0s;
```

Мы разместили «официальную» версию без префикса, в силу чего она будет иметь приоритет над остальными, когда браузеры полностью реализуют соответствующий стандарт.



ОГРАНИЧЕНИЯ ПЕРЕХОДОВ

При использовании переходов нужно помнить о некоторых ограничениях. К отдельным свойствам, например к свойству `background-gradient`, нельзя применить эффект перехода, несмотря на то что в спецификации (и даже в самом последнем редакторском черновике по адресу <http://dev.w3.org/csswg/css3-transitions/>) говорится, что это допустимо. Однако теоретически вы можете применить эффект перехода ко всем свойствам, перечисленным по адресу <http://www.w3.org/TR/css3-transitions/#properties-from-css->.

Применение разных по длительности эффектов перехода к различным свойствам

Если в правиле объявлено несколько свойств, то вам не обязательно применять ко всем одинаковые по длительности эффекты перехода. Взгляните на следующее правило:

```
#content a {  
    ...(еще стили)...  
    transition-property: border, color, text-shadow;  
    transition-duration: 2s, 3s, 8s;  
}
```

Здесь, в `transition-property`, мы указали, что хотим применить эффект перехода к `border`, `color` и `text-shadow`. Затем, в объявлении `transition-duration`, мы определили, что для `border` эффект перехода должен длиться 2 секунды, для `color` — 3 секунды, а для `text-shadow` — 8 секунд. Порядок разделенных запятыми значений длительности соответствуют порядку свойств, к которым будет применяться эффект перехода.

Понятие временных функций

Большинство свойств, к которым применяется эффект перехода, не требуют пояснения. Мы рассмотрели свойства, к которым можно (или нужно!) применить такой эффект. Длительность и задержки задаются в секундах (например, 2s), поэтому в них довольно легко разобраться, однако временные функции могут вас озадачить. Что именно делают `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out` и `cubic-bezier`? Каждая из этих функций в действительности позволяет «рисовать» кубическую кривую Безье, — по сути, делает то же самое, что и функция замедления. Я понимаю, что все это, возможно, вам тоже не о многом говорит. Поэтому я рекомендую вам заглянуть по адресу <http://cubic-bezier.com/> (рис. 7.3).

На этом сайте вы сможете сравнить временные функции и увидеть, чем каждая из них отличается от других. Однако, даже если вы умеете «с завязанными глазами» создавать свои собственные кубические кривые Безье (и при этом считать в обратном направлении от тысячи по-китайски), существует вероятность, что в большинстве практических ситуаций это не будет иметь особого значения. И вот почему...

Как и любое усовершенствование, эффекты перехода необходимо применять искусно. При практической реализации переходы, длящиеся слишком долго, имеют тенденцию создавать у пользователей впечатление, что сайт работает медленно. Например, навигационные ссылки, для которых переход длится 5 секунд, скорее вызовут у ваших пользователей раздражение, нежели заставят их воскликнуть: «Ух ты!» Следовательно, если только не будет убедительных причин поступить по-другому, использование перехода по умолчанию (`ease`), длящегося короткий промежуток времени (я предпочитаю такой, что длится максимум 1 секунду), зачастую будет наилучшим решением.

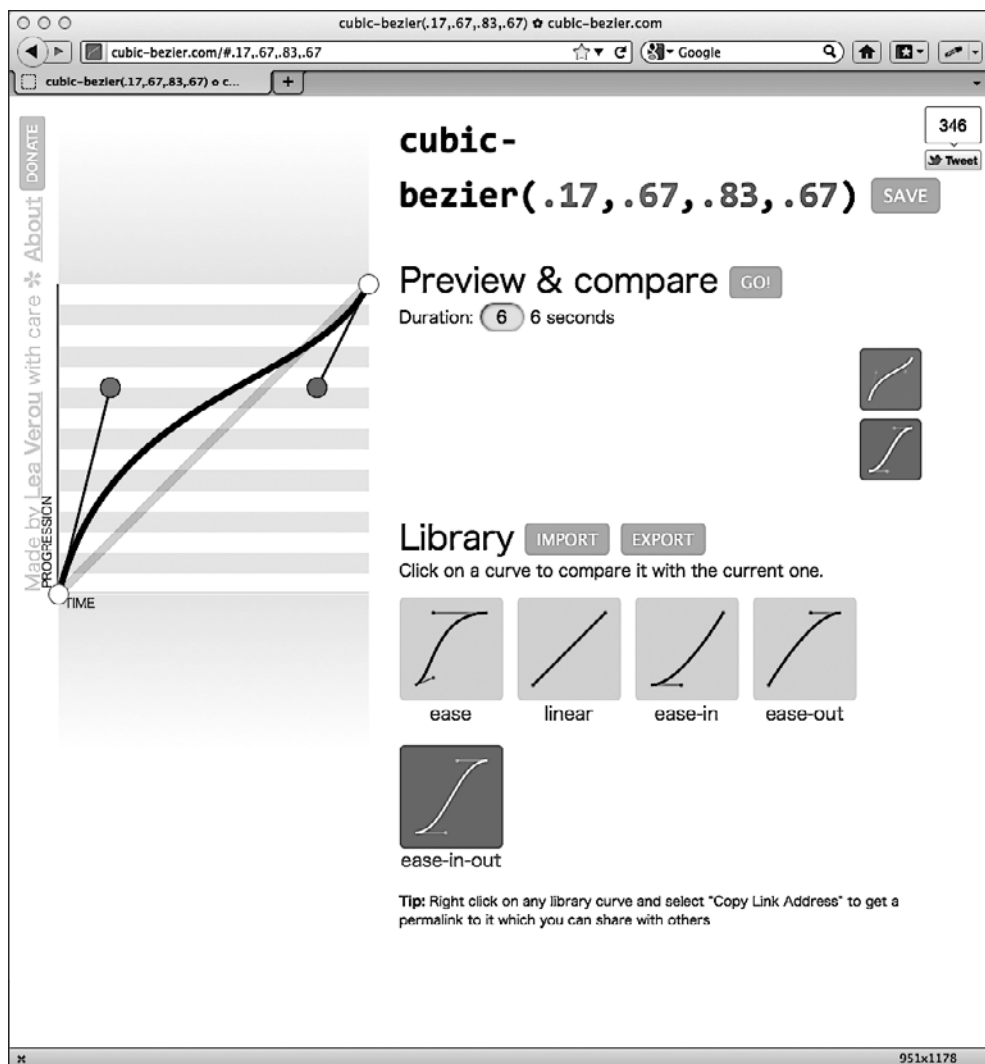


Рис. 7.3. Сайт, демонстрирующий возможности временных функций

Занятные переходы для адаптивных сайтов

Превратившись в одержимого адаптивным веб-дизайном, вы станете замечать, что постоянно изменяете размеры окна браузера при просмотре посещаемых вами сайтов, чтобы узнать, являются ли они адаптивными. Имейте в виду, что такая привычка раздражает многих людей, поэтому это лучше делать только при отсутствии посторонних.

По адресу <http://css-tricks.com> располагается отличный сайт Криса Койера (Chris Coyier), где обсуждаются CSS-методики. Однажды, внося изменения в дизайн, я поменял размеры окна браузера и понимающе улыбнулся, когда увидел, как для разных экранных элементов быстро выполнялся занятный переход. К какой магии прибег Крис, чтобы добиться такого эффекта? К чему-то вроде следующего:

```
* {  
    transition: all 1s;  
}
```

Здесь используется универсальный CSS-селектор `*` для выбора всех элементов, после чего задается переход для этих элементов (`all`) длительностью 1 секунда (`1s`). Поскольку здесь не указана временная функция, `ease` станет использоваться по умолчанию, и не будет никаких задержек, так как предполагается, что они отсутствуют, если специально не указать альтернативное значение. Результат? Что ж, поведение большинства объектов (ссылок, выделенных состояний и т. п.) будет таким, как вы ожидаете. Однако, поскольку переход осуществляется для *всех* элементов, также задействуются любые правила в медиазапросах, чтобы при изменении размеров окна браузера элементы как бы перетекали из одного состояния в другое. Важно ли это? Абсолютно нет! Забавно ли наблюдать такой эффект и «играть» с ним? Безусловно!

7.2. 2D-трансформации CSS3

CSS-трансформации (как 2D-, так и 3D-варианты) совершенно не то же самое, что CSS-переходы, несмотря на то что они могут показаться схожими. Думайте о них так: переходы позволяют сделать плавным процесс смены одного состояния на другое, а трансформации определяют то, каким станет соответствующий элемент. Вот мой собственный (надо признать, ребяческий) способ запомнить все это:

Представьте робота-трансформера, например Оптимуса Прайма. Это робот, который за некоторый промежуток времени (переход) превращается (трансформируется) в кое-что другое — в грузовик.

Если этот способ лишь еще сильнее запутал вас (либо вы не имеете ни малейшего понятия о том, кто такой Оптимус Прайм), копнем глубже. Мы добавим 2D-переход к выделенному состоянию навигационных ссылок на сайте And the winner isn't...:

```
nav ul li a:hover {  
    transform: scale(1.7);  
}
```

Теперь при наведении указателя мыши на ссылку в современном браузере будет наблюдаться следующий эффект (рис. 7.4).



Рис. 7.4. При наведении указателя мыши ссылка увеличивается

Мы сообщили браузеру, что хотим, чтобы при наведении указателя мыши на данный элемент он увеличивался в 1,7 раза по сравнению со своими первоначальными размерами.

Если вы решите добавить такое правило для элемента в Safari, то знайте, что необходимо, чтобы основной элемент отображался как блочный. Например:

```
nav ul li a {  
    height: 42px;  
    text-decoration: none;  
    text-transform: uppercase;  
    color: black;  
    text-shadow: 0 1px 0 hsla(0, 0%, 100%, 1.0);  
    font: 1.875em/42px 'BebasNeueRegular';  
    display: block;  
}
```

В противном случае ничего не произойдет, что, как вы понимаете, будет плохо.

Что можно трансформировать?

Существует две группы доступных CSS3-трансформаций: 2D и 3D. 2D-варианты намного более широко реализованы, поддерживаются браузерами и, несомненно, более просты в написании, поэтому сначала взглянем на них. CSS3 2D Transforms Module позволяет использовать следующие трансформации:

- `scale` — применяется для масштабирования элементов (чтобы делать их больше или меньше);
- `translate` — дает возможность сдвигать элементы на экране (вверх, вниз, влево и вправо);
- `rotate` — позволяет поворачивать элементы на заданную величину (определяемую в градусах);
- `skew` — используется для сдвига элементов по осям координат *X* и *Y*;
- `matrix` — позволяет смещать элементы и задавать форму их трансформаций с точностью до пиксела.

Попробуем каждую из этих трансформаций и посмотрим, каких результатов можно добиться.

scale

Мы уже рассматривали эту трансформацию ранее. Тогда мы задавали положительные значения. Однако вам следует знать, что, применяя значения меньше 1, можно уменьшать элементы (рис. 7.5). Приведенный далее код позволит уменьшить элемент наполовину:

```
transform: scale(0.5);
```

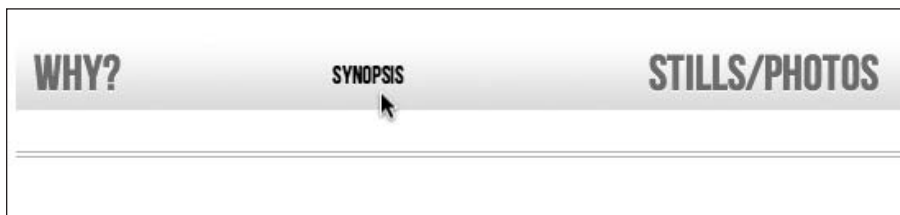


Рис. 7.5. При наведении указателя мыши ссылка уменьшается

translate

```
transform: translate(40px, 0px);
```

Команда `translate` дает браузеру указание сдвинуть соответствующий элемент на величину, определяемую либо в пикселах, либо в процентах. Этот синтаксис будет применяться так: сначала элемент сдвинется слева направо (на 40 пикселей в нашем случае), а затем сверху вниз (на 0 пикселей в нашем случае, благодаря чему он останется на одной линии с прочими навигационными элементами). Положительные значения, указанные в скобках, приводят к сдвигу элемента вправо или вниз, а отрицательные — к сдвигу элемента влево или вверх. Таким образом, в результате применения этого объявления к навигационной ссылке в выделенном состоянии она сдвинется на 40 пикселей вправо, когда пользователь наведет на нее указатель мыши (рис. 7.6).



Рис. 7.6. При наведении указателя мыши ссылка сдвигается вправо

rotate

```
transform: rotate(90deg);
```

Команда `rotate` позволяет поворачивать элементы. В этом примере мы заставили ссылку поворачиваться на 90° при наведении на нее указателя мыши. На рис. 7.7 показано, как все это будет выглядеть в браузере.



Рис. 7.7. При наведении указателя мыши ссылка поворачивается

Значение в скобках всегда следует указывать в градусах (например, `90deg`). Однако это не мешает вам посумасбродничать — вы можете заставить элементы вращаться, указав значение вроде следующего:

```
transform: rotate(3600deg);
```

В результате элемент будет совершать десять полных оборотов. Случаи практического использования этого конкретного значения встречаются редко, однако, если вам когда-нибудь доведется заниматься дизайном сайтов для компаний, деятельность которых связана с ветряными установками, то оно может пригодиться!

skew

Если у вас есть кое-какой опыт работы в Photoshop, то вы хорошо сможете представить, что делает `skew`. Трансформация `skew` позволяет скашивать элементы по обоим осям либо по одной из них.

```
transform: skew(10deg, 2deg);
```

Применение этого кода к ссылке приведет к следующему результату (рис. 7.8).



Рис. 7.8. При наведении указателя мыши ссылка перекашивается

Первое значение — это угол скоса элемента по оси *X* (`10deg` в нашем примере), а второе (`2deg`) — угол скоса элемента по оси *Y*. Если не указать второе значение,

то по оси X (по горизонтали) будет применен скос произвольной величины. Например:

```
transform: skew(10deg);
```

Этот код абсолютно правильный, однако он обеспечит скашивание элемента только по оси X . Значения обязательно нужно указывать в градусах. Положительные значения всегда приводят к вращению элемента по часовой стрелке, а отрицательные — против часовой стрелки.

matrix

Вернемся к вопросу о слишком переоцененных фильмах. Что такое? Вы хотите узнать о CSS3-трансформации `matrix`, а не о фильме с одноименным названием? Ох, ну ладно...

Синтаксис трансформации `matrix` выглядит пугающе:

```
transform: matrix(1.678, -0.256, 1.522, 2.333, -51.533, -1.989);
```

По сути, вы можете объединять ряд отличающихся трансформаций (`scale`, `rotate`, `skew` и т. д.) в одно объявление. Приведенное чуть выше объявление обеспечит в браузере следующий эффект (рис. 7.9).



Рис. 7.9. При наведении указателя мыши к ссылке применяется несколько трансформаций

Я люблю справляться с самыми сложными задачами (если только речь не идет о том, чтобы высидеть до конца фильма «Мулен Руж»), однако уверен, что вы согласитесь с тем, что попытка разобраться в данном синтаксисе может превратиться в настоящее испытание. Все станет еще хуже, когда вы заглянете в спецификацию и осознаете, что для его полного понимания потребуется хорошее знание математики: <http://www.w3.org/TR/css3-2d-transforms/#cssmatrix-interface>.

Матричные трансформации для хитрецов и ленивых. Меня нельзя назвать математиком даже с самой большой натяжкой, поэтому, столкнувшись с необходимостью создать трансформацию на основе матрицы, я решил схитрить. Если ваши познания в математике тоже не блещут, то я рекомендую вам зайти по адресу <http://www.useragentman.com/matrix/> (рис. 7.10).

Сайт **Matrix Construction Set** позволяет перетаскивать элементы именно так, как вы желаете, а также предусматривает возможность выполнить старую добрую операцию копирования и вставки кода (включая префиксы поставщиков) для ваших CSS-файлов.

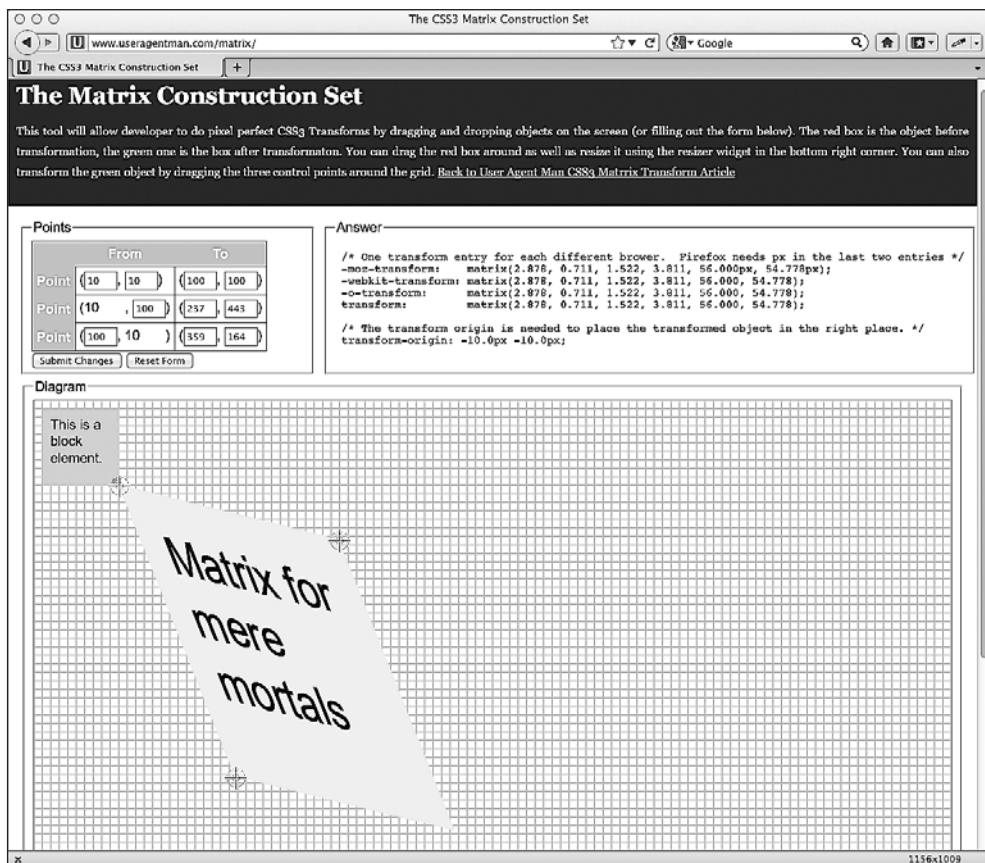


Рис. 7.10. Сайт для задания матричных трансформаций

Свойство transform-origin

Наряду с упоминавшимися ранее трансформациями вы можете использовать свойство `transform-origin` для изменения точки, относительно которой осуществляются трансформации:

```
transform: rotate(45deg);
transform-origin: 20% 20%;
```

Задействовав этот код для навигационных ссылок, мы получим следующий результат (рис. 7.11).

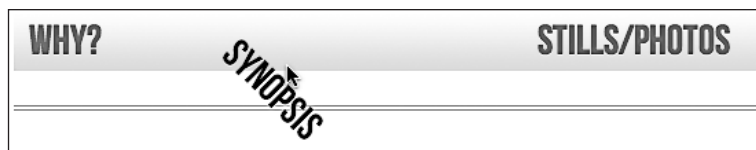


Рис. 7.11. При наведении указателя мыши ссылка поворачивается на 45° и смещается вниз

Свойство `transform-origin` придется кстати, поскольку по умолчанию трансформации выполняются относительно центра элементов. Это удобный инструмент, позволяющий сместить точку, относительно которой осуществляются трансформации, и помогающий добиться интересных результатов.

**ПРИМЕЧАНИЕ**

Полную информацию о свойстве `transform-origin` можно отыскать по адресу <http://www.w3.org/TR/css3-2d-transforms/#transform-origin-property>.

Вот мы и рассмотрели основы 2D-трансформаций. Они намного более широко реализованы в браузерах, чем их 3D-собратья. Если разумно подходить к использованию 2D-трансформаций, они будут легковесными средствами обеспечения визуальных «украшательств», которыми смогут насладиться пользователи современных браузеров.

**ПРИМЕЧАНИЕ**

Полную спецификацию CSS3 2D Transforms Module Level 3 вы можете найти по адресу <http://www.w3.org/TR/css3-2d-transforms/>.

7.3. Вкратце об обеспечении 3D-трансформаций CSS3

Несмотря на то что 3D-трансформации CSS3 уже поддерживаются браузерами на основе WebKit (Safari и Chrome) и Firefox версии 10 и выше, их работа в Internet Explorer будет возможна только в его версии под номером 10. Однако, невзирая на недостаточную поддержку настольными браузерами, 3D-трансформации CSS3 хорошо работают в браузерах, устанавливаемых в операционных системах Android (версии 3 и выше) и iOS (всех версий), благодаря тому, что эти браузеры базируются на WebKit.

В дальнейшем вам лучше всего проверять свои результаты в браузере на основе WebKit, например в Chrome или Safari (если только, конечно, вы не читаете эту книгу в то время, когда ваш излюбленный браузер уже *поддерживает* 3D-трансформации).

В этом разделе мы лишь вкратце поговорим об обеспечении 3D-трансформаций. О 3D-трансформациях можно говорить долго, а предоставляемые ими возможности практически безграничны. Я уже представляю себе, как к тому моменту, когда они получат широкую поддержку, большинство из нас будет прибегать к ним для создания, например, эффектов карусели, вместо того чтобы полагаться на JavaScript-решения вроде jQuery. А пока это время не наступило, просто разберемся, что можно делать с помощью 3D-трансформаций.

Допустим, мы решили организовать на сайте And the winner isn't... незамысловатую викторину. Для обеспечения соответствующей функциональности мы задействуем изображения в виде постеров фильмов, а пользователям нужно будет угадать, хорошим или плохим считается определенный фильм по мнению самого авторитетного мирового кинокритика (да, это я). При наведении указателя мыши

на любое из этих изображений (или прикосновении к нему на сенсорном экране) будет показываться ответ.

Далее приведен соответствующий фрагмент разметки. Обратите внимание, что я не стал дублировать разметку для каждого изображения, поскольку она будет абсолютно одинакового формата для всех постеров:

```
<section class="Qcontainer">
  <div class="film">
    <div class="face front">
      
    </div>
    <div class="face back"><h5>HOT!</h5></div>
  </div>
</section>
```

А вот и CSS-код. Поскольку браузеры на основе WebKit наилучшим образом поддерживают 3D-трансформации, то во всех объявлениях здесь используется соответствующий префикс поставщика. Как и всегда, при реализации на практике префиксы поставщиков будут вашими друзьями.

```
.Qcontainer {
  height: 100%;
  width: 28%;
  position: relative;
  -webkit-perspective: 800;
  float: left;
  margin-right: 2%;
}
.film {
  width: 100%;
  height: 15em;
  -webkit-transform-style: preserve-3d;
  -webkit-transition: 1s;
}
.Qcontainer:hover .film {
  -webkit-transform: rotateY(180deg);
}
.face {
  position: absolute;
  -webkit-backface-visibility: hidden;
}
.back {
  width: 66%;
  height: 127%;
  -webkit-transform: rotateY(180deg);
  background: #3b3b3b;
  background: -webkit-linear-gradient(top,
                                     rgba(0,0,0,0.65) 0%,
                                     rgba(0,0,0,0) 100%);
  padding: 15%;
}
```

Когда этот код займет свое место, при наведении указателя мыши на соответствующий постер фильма изображение будет переворачиваться, после чего пользователь увидит простой ответ — **НОТ! (ХОРОШИЙ!)** или **НОТ! (ПЛОХОЙ!)** (рис. 7.12).

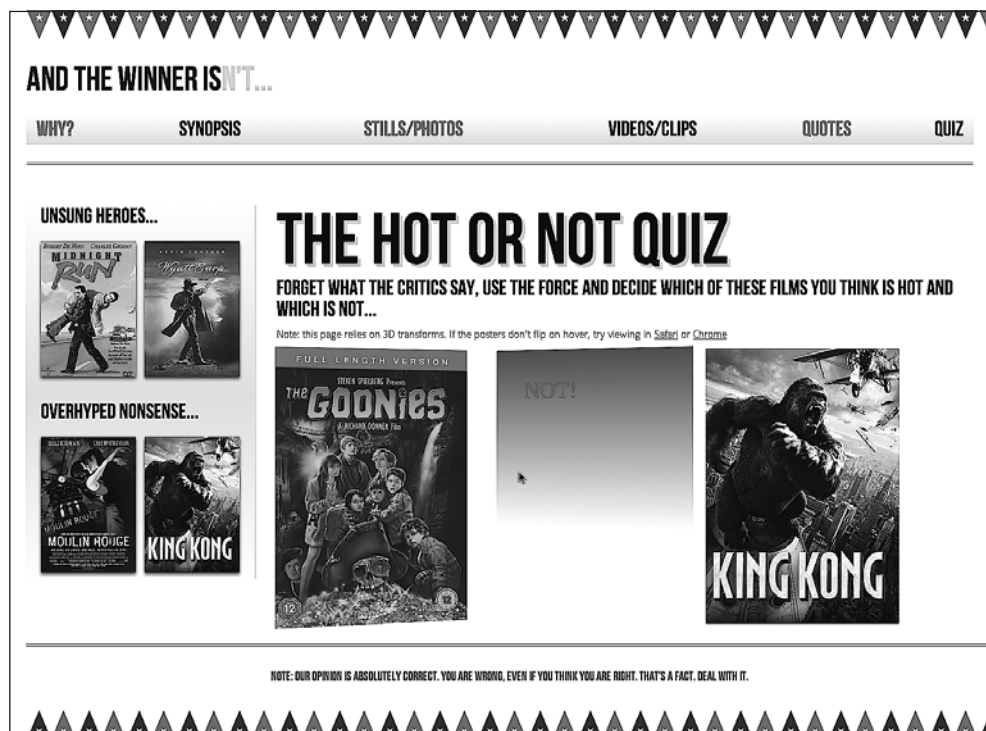


Рис. 7.12. При наведении указателя мыши постер к фильму переворачивается

Анализ 3D-эффекта

Шаг за шагом разберем код, чтобы понять, как был достигнут данный эффект.

Первый важный этап — установка значения для `-webkit-perspective` родительского элемента. Это позволит активизировать 3D-пространство:

```
.Qcontainer {
  height: 100%;
  width: 28%;
  position: relative;
  -webkit-perspective: 200;
  float: left;
  margin-right: 2%;
}
```

Чем выше значение `-webkit-perspective`, тем больше будет виртуальная глубина 3D-пространства с вашей точки зрения. Следовательно, для слабо выраженного 3D-эффекта нужно увеличить данное значение, а для сильно выраженного 3D-эффекта — уменьшить его.

Следующая команда, заслуживающая внимания:

```
.film {
  width: 100%;
  height: 15em;
  -webkit-transform-style: preserve-3d;
  -webkit-transition: 1s;
}
```

Первое объявление перспективы, добавленное в класс `.Qcontainer`, применяется только к первому прямому потомку (элемент `div` с классом `.film` в этом примере). Следовательно, для дальнейшей передачи перспективы родительского элемента мы используем значение `preserve-3d`.

Теперь мы добавим правило для «переворачивания» элемента `div` с классом `.film` при наведении указателя мыши на область `.Qcontainer`:

```
.Qcontainer:hover .film {
  -webkit-transform: rotateY(180deg);
}
```

Следующее правило скрывает из виду противоположную сторону постера, когда он переворачивается:

```
.face {
  position: absolute;
  -webkit-backface-visibility: hidden;
}
```

Абсолютное позиционирование элемента `div` с классом `.face` необходимо для того, чтобы он располагался над элементом `div` с классом `.back`:

```
.back {
  width: 66%;
  height: 127%;
  -webkit-transform: rotateY(180deg);
  background: #3b3b3b;
  background: -webkit-linear-gradient(top,
    rgba(0,0,0,0.65) 0%,
    rgba(0,0,0,0) 100%);
  padding: 15%;
}
```

И наконец, мы также добавляем простой метод `rotateY` для `div` с классом `.back`. Без него этот элемент `div`, по сути, «просвечивался» бы сквозь `div` с классом `.front`.

Вот и все. Теперь при наведении указателя мыши на любой из постеров картинка будет переворачиваться довольно эффектным образом.

Однако в браузерах, не основанных на WebKit, функциональность нашей страницы явно хромает (рис. 7.13).



Рис. 7.13. В браузерах, не поддерживающих 3D-трансформации, анимация не работает

Что ж, мы можем обеспечить приемлемый резервный вариант для браузеров, которые не основаны на WebKit, добавив еще немного CSS-кода, как в былые времена:

```
.front {
  z-index: 5;
}
.Qcontainer:hover .front {
  z-index: 0;
}
```

Сначала мы задаем для z-index значение 5 для элемента div с классом .front, чтобы он по умолчанию располагался над div с классом .back:

```
.front {
  z-index: 5;
}
```

Для состояния наведения указателя мыши на область `.Qcontainer` мы зададим для `z-index` значение 0, чтобы она снова оказалась позади `div` с классом `.back`:

```
.Qcontainer:hover .front {
  z-index: 0;
}
```

Теперь лишенная «украшательств» функциональность «вопрос/ответ» будет доступна в браузерах, не поддерживающих 3D-трансформации, но без занятого 3D-эффекта (рис. 7.14).



Рис. 7.14. Изображения переворачиваются, но без 3D-эффекта

3D-трансформации не готовы к повсеместному внедрению

По своему опыту могу сказать, что в настоящее время многие 3D-трансформации не очень хорошо работают в сочетании с размерами в процентах (например, изменение ширины области просмотра в предыдущем примере приведет к тому, что поведение всех элементов станет крайне ненормальным). Поэтому зачастую приходится тратить немало сил на конфигурирование, чтобы 3D-трансформации хо-

рошо работали в адаптивных макетах. Более того, поскольку на текущий момент поддержка 3D-трансформаций ограничена, они редко оказываются наиболее подходящими решениями при создании кросс-браузерных сайтов. Из-за этого я пока все еще прибегаю к jQuery или схожим инструментам для обеспечения функциональности такого рода.

Однако возможности 3D-трансформаций CSS выглядят весьма многообещающими, и, когда их поддержка со стороны браузеров расширится, они позволят нам перенести многие занятные эффекты в таблицы стилей, а не полагаться на JavaScript, как мы это делаем сейчас.



ПРИМЕЧАНИЕ

Ознакомиться с новейшими разработками W3C в сфере 3D-трансформаций CSS можно по адресу <http://dev.w3.org/csswg/css3-3d-transforms/>.

7.4. Анимация с помощью CSS3

Если вам когда-нибудь доводилось работать с Flash, то имеющиеся у вас знания очень помогут вам в создании CSS3-анимаций. CSS3 работает по тем же соглашениям, касающимся кадрирования изображений для обеспечения анимаций, которые можно встретить во Flash- и прочих приложениях, базирующихся на временной шкале.

Анимации шире реализованы, чем 3D-трансформации. Они поддерживаются в Firefox версии 5 и выше, Chrome, Safari версии 4 и выше, Android (всех версий), iOS (всех версий), кроме того, их поддержка должна быть внедрена в Internet Explorer 10.

CSS3-анимация состоит из двух компонентов: сначала идет объявление @keyframes, которое затем используется в свойстве animation. Взглянем на них.

В предыдущем разделе мы обеспечили простой эффект переворачивания постеров для тех фильмов, которые я считаю хорошими или плохими. Текст выглядит довольно скучно, поэтому добавим красивый пульсирующий эффект к ответам, отображаемым на экране после переворачивания постеров.

Сначала напишем правило @keyframes:

```
@keyframes warning {
  0% {
    text-shadow: 0px 0px 4px #000000;
  }
  50% {
    text-shadow: 0 0 20px #000000;
  }
  100% {
    text-shadow: 0px 0px 4px #000000;
  }
}
```

Здесь я использую версию кода без префиксов, поэтому, если не будет наблюдаться никакого эффекта, то вам, вероятно, потребуется добавить полный набор вариантов с префиксами поставщиков (среди которых, например, будет @-webkit-keyframes).

Проанализируем код:

```
@keyframes warning {
  0% {
    text-shadow: 0px 0px 4px #000000;
  }
  50% {
    text-shadow: 0 0 20px #000000;
  }
  100% {
    text-shadow: 0px 0px 4px #000000;
  }
}
```

Сначала мы указываем объявление @keyframes. Затем мы присваиваем ему имя warning в данном случае. Вы можете присваивать объявлениям @keyframes любые имена по своему усмотрению, однако, поскольку эти объявления могут быть повторно использованы в отношении разных элементов, им следует присваивать соответствующие имена.

Вы можете задать столько процентных пунктов, сколько захотите (например, 10, 20, 30, 40 и т. д.), либо, если вас это больше устроит, определить анимацию с помощью значений from и to. Однако имейте в виду, что браузеры на основе WebKit не всегда хорошо работают с этими значениями (предпочитая 0% и 100%):

```
@keyframes warning {
  from {
    text-shadow: 0px 0px 4px #000000;
  }
  50% {
    text-shadow: 0 0 40px #000000;
  }
  to {
    text-shadow: 0 0 4px #000000;
  }
}
```

В данном примере я изменяю text-shadow, начиная с одного и того же значения радиуса размытия и заканчивая им же. Это значение равно 4px, но превращается в 40px при 50%.

Теперь, объявив @keyframes, мы можем ссылаться на него в свойстве animation:

```
.back h5 {
  font-size: 4em;
  color: #f2050b;
  text-align: center;
  animation: warning 1.5s infinite ease-in;
}
```

Указав свойство `animation`, мы определяем конкретное правило `@keyframes`, которое желаем использовать (warning в данном случае), после чего задаем значение для `animation-iteration-count` (здесь мы устанавливаем значение `infinite`, из-за чего анимация будет выполняться непрерывно) и, наконец, вводим временную функцию (`ease-in`). Ясно, что статичное изображение не позволит наглядно передать весь эффект, но, надеюсь, вы сможете представить себе, как отбрасываемая текстом тень пульсирует. Чтобы увидеть эффект вживую, откройте браузер и зайдите на сайт <http://www.andthewinnerisnt.com> (рис. 7.15).

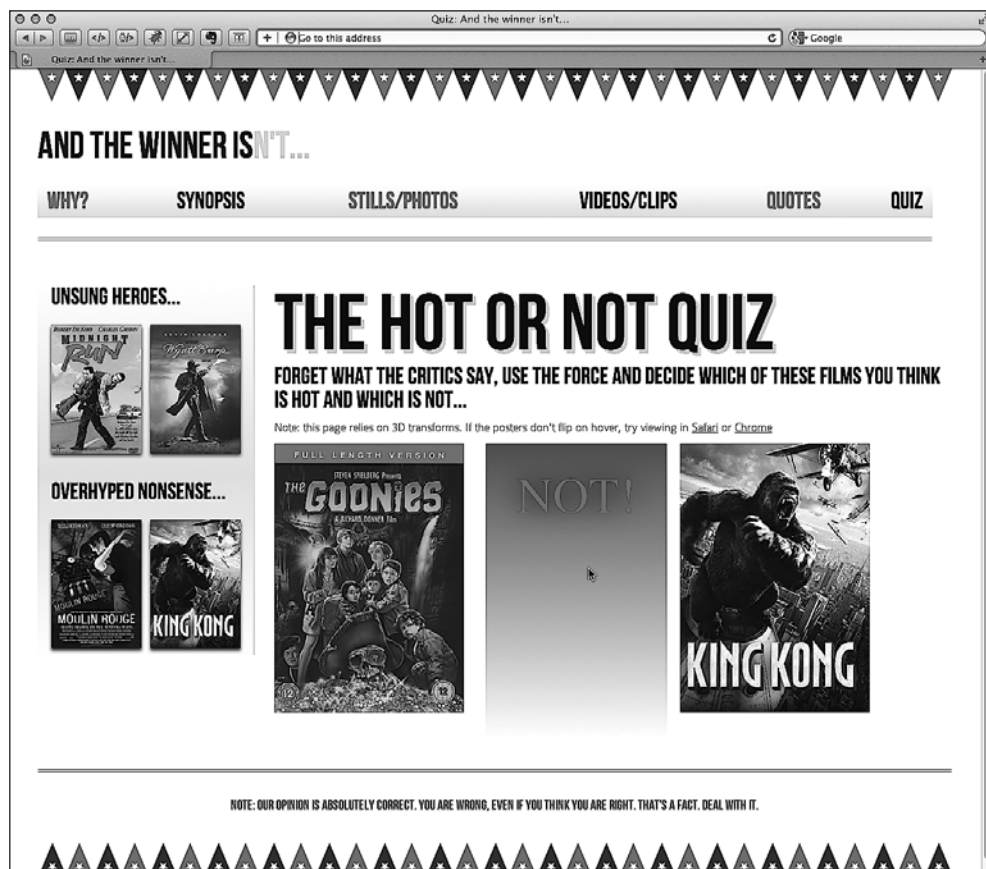


Рис. 7.15. Текст на обратной стороне постера теперь пульсирует красным цветом

Собирательное свойство может включать в себя все семь анимационных свойств. В дополнение к тем, что мы указали в примере выше, можно указать `animation-delay` (например, если вам необходимо, чтобы выполнение анимации начиналось с задержкой), `animation-play-state` (для него можно задать значение `running` либо `paused`, чтобы воспроизводить и ставить анимацию на паузу соответственно) и, наконец, `animation-fill-mode`, с необходимостью применения которого я в своей практике еще не сталкивался (по умолчанию для него устанавливается значение `none`).

Естественно, вы не обязаны использовать собирательное свойство, а можете указать все свойства по отдельности, как показано далее:

```
animation-name: warning;
animation-duration: 1.5s;
animation-timing-function: ease-in-out;
animation-iteration-count: infinite;
animation-play-state: running;
animation-delay: 0s;
animation-fill-mode: none;
```

Как уже отмечалось, анимации можно повторно использовать для других элементов. Например:

```
nav ul li a:hover {
    animation: warning 1.5s infinite ease-in;
}
```

Этот код обеспечит аналогичный пульсирующий эффект для наших навигационных ссылок. Я надеюсь, вы заметили ссылку **STILLS/PHOTOS** на рис. 7.16 среди всех этих анимаций. Зайдите на сайт <http://www.andthewinnerisnt.com> и попробуйте навести на нее указатель мыши.

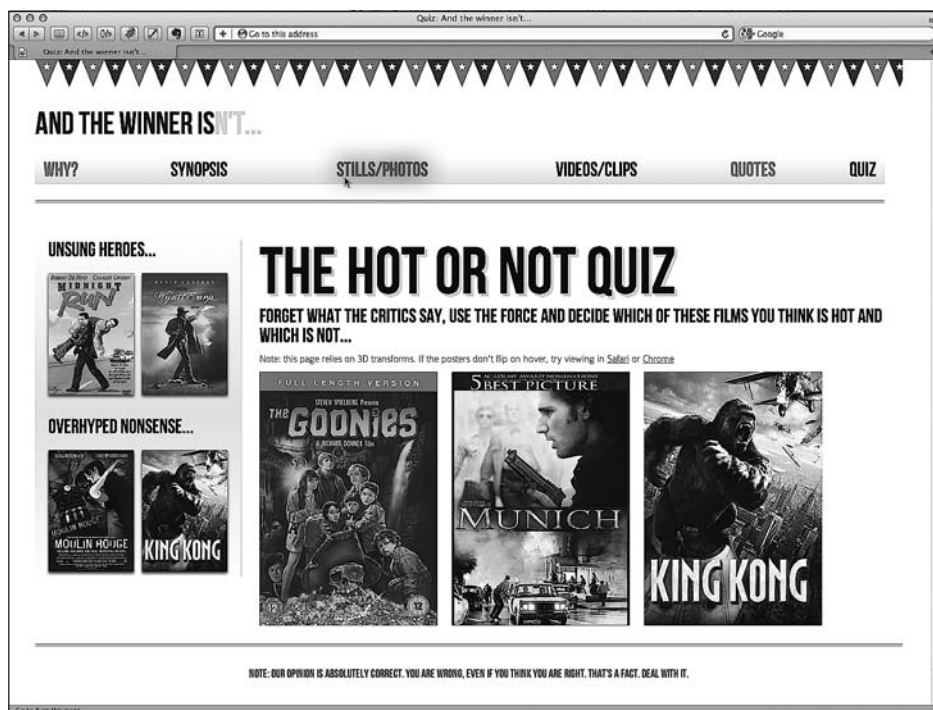


Рис. 7.16. При наведении на ссылку указателя она начинает пульсировать

Это был всего лишь один простой пример использования CSS-анимаций. Поскольку, в сущности, анимировать с помощью ключевых кадров можно все,

что угодно, то открывающиеся возможности поистине безграничны. В Интернете представлено бесчисленное множество примеров использования анимационных методик CSS3. Страницы вроде расположенной по адресу <http://webdesignerwall.com/trends/47-amazing-css3-animation-demos> наверняка вдохновят вас на создание таких анимаций.



ПРИМЕЧАНИЕ

Ознакомиться с новейшими разработками W3C в области CSS3-анимаций можно по адресу <http://dev.w3.org/csswg/css3-animations>.

Сведение воедино CSS3-трансформаций и анимаций. Попробуем сделать еще одну вещь, чтобы поиграть нашими «CSS3-мускулами». Я хотел бы попытаться разместить все располагающиеся во врезке <aside> изображения под разными углами, а затем анимировать их. Цель состоит в том, чтобы они «раскачивались» из стороны в сторону при первом посещении нашей веб-страницы. Вот разметка для врезки:

```
<aside>
  <div role="complementary">
    <div class="sideBlock unsung">
      <h1>Unsung heroes...</h1>
      <a href="#"></a>
      <a href="#"></a>
    </div>
  </div>
  <div role="complementary">
    <div class="sideBlock overHyped">
      <h1>Overhyped nonsense...</h1>
      <a href="#"></a>
      <a href="#"></a>
    </div>
  </div>
</aside>
```

Теперь создадим CSS3-код, чтобы обеспечить нужный эффект. Сначала напишем новое объявление @keyframe с именем swing:

```
@-webkit-keyframes swing {
  from {
    transform: rotate(3deg);
  }
  20% {
    transform: rotate(7deg);
  }
  60% {
    transform: rotate(10deg);
  }
}
```

```

80% {
  transform: rotate(7deg);
}
to {
  transform: rotate(3deg);
}
}

```

В нашей анимации будет использоваться 2D-трансформация `rotate` для поворота элементов на величину от 3 до 10° в разные стороны. А вот как мы добавим свойство `animation`:

```

#quiz .unSung a:nth-child(odd) img {
  transform: rotate(3deg);
  animation: swing 0.1s 5 ease-in;
}
#quiz .unSung a:nth-child(even) img {
  transform: rotate(-3deg);
  animation: swing 0.1s 5 0.3s ease-in;
}
#quiz .overHyped a:nth-child(odd) img {
  transform: rotate(3deg);
  animation: swing 0.1s 5 0.2s ease-in;
}
#quiz .overHyped a:nth-child(even) img {
  transform: rotate(-3deg);
  animation: swing 0.1s 5 0.5s ease-in;
}

```

Проанализируем приведенный выше код. Сначала, исходя из специфики CSS, мы можем нацелить эти правила исключительно на страницу с викториной (в которой имеется тег `<body id="quiz">`).

Перед добавлением свойства `animation` я хочу задать трансформацию `rotate`, которая будет использоваться по умолчанию, чтобы изображения «застыли» в беспорядочном состоянии, когда анимация завершится. Мне не нужно, чтобы все они располагались под одним углом, поэтому используем селектор `nth-child`, о котором мы говорили в главе 5, для выбора четных и нечетных изображений и обеспечим их поворот под разными углами:

```

#quiz .unSung a:nth-child(odd) img {
  transform: rotate(3deg);
  animation: swing 0.1s 5 ease-in;
}
#quiz .unSung a:nth-child(even) img {
  transform: rotate(-3deg);
  animation: swing 0.1s 5 0.3s ease-in;
}
#quiz .overHyped a:nth-child(odd) img {
  transform: rotate(3deg);
  animation: swing 0.1s 5 0.2s ease-in;
}
#quiz .overHyped a:nth-child(even) img {

```

```

transform: rotate(-3deg);
animation: swing 0.1s 5 0.5s ease-in;
}

```

Затем мы добавим свойство `animation` для каждого экземпляра. Вы заметите небольшие различия в каждом из правил. Собирательное свойство также учитывает, что второе указанное значение времени (0.5s) — это величина задержки начала выполнения анимации. Используя его, мы фактически можем по отдельности запускать выполнение анимации для каждого изображения.

```

#quiz .overHyped a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.5s ease-in;
}

```

Опять-таки, когда пишешь об анимациях, трудно передать то, как будет выглядеть соответствующий эффект вживую. Если вы не находитесь поблизости от места, где можно подключиться к Интернету, то лучшее, что я могу сделать для описания получившегося эффекта, — сказать, что постеры фильмов «раскачиваются» из стороны в сторону, а затем «застывают» в беспорядочном состоянии, как показано на рис. 7.17.



Рис. 7.17. На рисунке сложно увидеть анимацию, но она есть!

7.5. Резюме

Рассказу о возможностях CSS-трансформаций, переходов и анимаций можно посвятить множество книг. Однако я надеюсь, что благодаря приведенному в этой главе материалу вы сможете разобраться в основах и использовать их в своей работе. В конечном счете применение новых свойств и методик CSS3 преследует цель сделать адаптивные веб-дизайны еще более легковесными и насыщенными, чем когда-либо прежде, вместо того, чтобы использовать JavaScript для внесения любительских эстетических улучшений. В этой главе мы выяснили, что такое CSS3-переходы и как их создавать, разобрались в таких функциях, как `ease` и `linear`, которые затем использовали для создания простых, но занятных эффектов для нашего адаптивного веб-дизайна. После этого мы поговорили обо всех 2D-трансформациях, например о `scale` и `skew`, а затем о том, как их использовать совместно с переходами. Кроме того, мы кратко взглянули на 3D-трансформации, прежде чем узнать всю мощь и относительную простоту CSS-анимаций. Безусловно, наши «CSS3-мускулы» растут!

Однако если и есть в дизайне сайтов область, которую я всегда по возможности стараюсь обходить стороной (так же упорно, как я стараюсь обходить стороной фильмы «Мюнхен» и «Кинг-Конг», когда идет их показ), то это создание форм. Не знаю почему, но я всегда считал это утомительной и сложной задачей. Представьте мою радость, когда я узнал, что HTML5 и CSS3 делают весь процесс создания, стилизации и даже валидации (да, валидации!) форм проще, чем когда-либо прежде. Я был просто счастлив. А значит, эта новость может столь же обрадовать и вас. В следующей главе я поделюсь с вами соответствующими знаниями.

8 Покорение форм с помощью HTML5 и CSS3

Исторически сложилось так, что для единообразной стилизации форм в разных браузерах приходится прилагать немало усилий. Кроме того, формы требуют применения JavaScript для валидации введенных данных, а также лишены специфических типов полей ввода для указания повседневной информации вроде телефонных номеров, адресов электронной почты и URL-адресов.

Хорошая новость заключается в том, что HTML5 решает большинство из этих распространенных проблем. Познакомимся с новыми функциями, касающимися HTML5-форм, и посмотрим, как они облегчают бремя разработчика.

В этой главе мы научимся использовать HTML5, чтобы:

- легко вставлять подсказку в соответствующие поля форм;
- при необходимости отключать автозаполнение полей форм;
- делать определенные поля обязательными для заполнения перед отправкой форм;
- задавать разные типы полей ввода, например, для указания адресов электронной почты, телефонных номеров и URL-адресов;
- создавать ползунки диапазонов чисел для легкого выбора необходимых значений;
- добавлять календари и палитры цветов;
- применять регулярные выражения для определения значений, разрешенных для ввода в формы;
- добавлять полизаполнения, обеспечивая поддержку для менее совместимых браузеров;
- применять CSS3 для легкой и гибкой стилизации форм.

8.1. HTML5-формы

Вот сценарий: рассматривая адаптивный сайт-образец And the winner isn't..., я решил, что пользователи должны иметь возможность высказать свое разочарование в неудачных фильмах, которые удостоились награды. Мы добавим форму,

позволяющую людям поведать нам о фильмах, которые, по их мнению, не должны были стать победителями, а также о фильмах, которые должны были оказаться на месте победителей.

На рис. 8.1 показано, как будет выглядеть наша основная форма с небольшой базовой стилизацией в браузере Chrome (версии 16).

The screenshot shows a web browser window with the address bar displaying "Oscar Redemption: And the winner isn't...". The page has a dark theme with a large, bold title "OSCAR REDEMPTION" in the center. Below the title is a subtitle: "HERE'S YOUR CHANCE TO SET THE RECORD STRAIGHT: TELL US WHAT YEAR THE WRONG FILM GOT NOMINATED, AND WHICH FILM SHOULD HAVE RECEIVED A NOD...".

On the left side, there are two columns of movie posters. The first column is titled "UNSUNG HEROES..." and shows posters for "MIDNIGHT RUN" and "KING KONG". The second column is titled "OVERHYPERED NONSENSE..." and shows posters for "MOULIN ROUGE" and "KING KONG".

The main form area is divided into three sections:

- About the offending film (part 1 of 3)**: This section contains four input fields: "The film in question?" (with the example "e.g. King Kong"), "Year Of Crime" (with a calendar icon), "Award Won", and "Tell us why that's wrong?" (with the example "I fell asleep within 20 minutes..."). Below these is a rating slider for "How you rate it (1 is woeful, 10 is awesomesauce)".
- What should have won? (part 2 of 3)**: This section contains three input fields: "The film that should have won?" (with the example "e.g. Cable Guy"), "Tell us why it should have won?" (with the example "Hello? CAABLLLE GUUY!!!!"), and "How you rate it (1 is woeful, 10 is awesomesauce)" (with the value "5" entered).
- About you? (part 3 of 3)**: This section contains four input fields: "Your Name" (with the example "Dwight Schultz"), "Telephone (so we can berate you if you're wrong)" (with the example "1-234-546758"), "Your Email address" (with the example "dwight.schultz@gmail.com"), and "Your Web address" (with the example "www.mysite.com").

At the bottom right of the form is a large button labeled "SUBMIT REDEMPTION".

Рис. 8.1. Форма на сайте And the winner isn't...

Помимо стандартных полей ввода и текстовых областей, в нашей форме будут присутствовать числовые поля ввода со значениями, изменяемыми с помощью мыши, ползунков, а также подсказки во многих полях.

Если мы установим фокус на определенном поле (то есть выберем его), то подсказка исчезнет, а если мы уберем фокус с этого поля, ничего не введя в него (еще раз щелкнув кнопкой мыши за пределами соответствующего поля), то подсказка снова отобразится в нем. Более того, открыв эту страницу в браузере Google

Chrome и попытавшись отправить форму, ничего не введя, мы увидим следующее (рис. 8.2).

OSCAR REDEMPTION
HERE'S YOUR CHANCE TO SET THE RECORD STRAIGHT: TELL US WHAT YEAR THE WRONG FILM GOT NOMINATED, AND WHICH FILM SHOULD HAVE RECEIVED A NOD...

UNsung HEROES...
MIDWINTER
KING KONG

OVERHYPED NONSENSE...
MIDWINTER
KING KONG

About the offending film (part 1 of 3)

The film in question?

Year Of Crime

Award Won

Tell us why that's wrong?

How you rate it (1 is woeful, 10 is awesomesauce)

What should have won? (part 2 of 3)

The film that should have won?

Tell us why it should have won?

How you rate it (1 is woeful, 10 is awesomesauce)

About you? (part 3 of 3)

Your Name

Telephone (so we can berate you if you're wrong)

Your Email address

Your Web address

SUBMIT REDEMPTION

Рис. 8.2. Форма просит заполнить поле

Получается, помимо того, что добавляются визуальные украшения (ползунок и числовые поля ввода со значениями, изменяемыми с помощью мыши), для нашей формы также будет выполняться валидация на стороне клиента. Как уже отмечалось, для того чтобы заставить форму работать подобным образом, обычно приходится прибегать к использованию разного рода JavaScript-решений.

Хорошая новость заключается в том, что все эти элементы интерфейса пользователя (включая упоминавшиеся ранее ползунок, подсказку и числовые поля ввода со значениями, изменяемыми с помощью мыши) обрабатываются исключительно посредством HTML5 без привлечения JavaScript. Посмотрим, как новые возможности HTML5 в области форм делают все это реальным.

Понятие составных частей HTML5-форм

В форму, созданную с использованием HTML5, входит много элементов, поэтому проанализируем ее. Для облегчения стилизации форма наделяется идентификатором, а затем — HTML5-тегом `<hgroup>` для заголовка и вступительного текста:

```
<form id="redemption" method="post">
  <hgroup>
    <h1>Oscar Redemption</h1>
    <h2>Here's your chance to set the record straight: tell us what year the wrong
film got nominated, and which film <b>should</b> have received a nod...</h2>
  </hgroup>
```

Три области формы затем заключаются в элемент `<fieldset>`, сопровождаемый тегом `<legend>`:

```
<fieldset>
<legend>About the offending film (part 1 of 3)</legend>
<div>
  <label for="film">The film in question?</label>
  <input id="film" name="film" type="text" placeholder="e.g. King Kong" required
    aria-required="true" >
</div>
```

Как вы можете видеть из этого фрагмента кода, каждый элемент `<input>` формы также заключается в `<div>` с соответствующей меткой. Пока все нормально. В первом теге `<input>` мы как раз сталкиваемся с нашим первым инструментом, касающимся HTML5-форм. После привычных атрибутов `id`, `name` и `type` мы видим `placeholder`.

placeholder

Атрибут `placeholder` в нашем случае выглядит так:

```
placeholder="e.g. King Kong"
```

Наличие подсказки в полях формы стало настолько распространенным требованием, что люди, создающие HTML5-код, решили, что она должна быть встроена в разметку и поддерживаться браузерами. Просто включите атрибут `placeholder` в свой тег `<input>`, и его значение будет по умолчанию отображаться, пока соответствующее поле ввода не получит фокус. Если оно потеряет фокус и при этом окажется, что пользователь не ввел никакого значения, то в данном поле ввода вновь отобразится подсказка.

После атрибута `placeholder` в приведенном чуть ранее фрагменте кода идет следующий параметр, касающийся HTML5-форм, — атрибут `required`.

required

Атрибут `required` в нашем случае выглядит так:

```
required aria-required="true"
```

В совместимых с HTML5 браузерах при добавлении логического атрибута `required` в элемент `<input>` вы даёте понять, что соответствующее поле ввода обяза-

тельно для заполнения. Если пользователь попытается отправить форму, не введя в это поле требуемое значение, то на экране отобразится предупреждающее сообщение. Это сообщение будет специфичным (по содержанию и стилизации) в зависимости от браузера и типа поля ввода. Помимо HTML5-атрибута `required`, в примере мы задействовали его эквивалент WAI-ARIA — `aria-required="true"`. Если только у вас не будет веской причины поступить по-другому, включайте в код эту версию атрибута `required`, чтобы облегчить задачу экранным дикторам (если помните, мы рассматривали WAI-ARIA еще в главе 4).

Мы уже видели, как выглядит предупреждающее сообщение в обязательном для заполнения поле ввода в браузере Chrome. На рис. 8.3 показано, как это же сообщение будет выглядеть в браузере Firefox (версии 9).

The screenshot shows a web browser window with the title "Oscar Redemption: And the winner isn't...". The page content includes a header "OSCAR REDEMPTION" and a sub-header "HERE'S YOUR CHANCE TO SET THE RECORD STRAIGHT: TELL US WHAT YEAR THE WRONG FILM GOT NOMINATED, AND WHICH FILM SHOULD HAVE RECEIVED A NOD...". The form is divided into three sections: "About the offending film (part 1 of 3)", "What should have won? (part 2 of 3)", and "About you? (part 3 of 3)". The first section contains fields for "Year Of Crime", "The film in question?", "Award Won", "Tell us why that's wrong?", and "How you rate it (1 is woeful, 10 is awesomesauce)". The second section contains fields for "The film that should have won?", "Tell us why it should have won?", and "How you rate it (1 is woeful, 10 is awesomesauce)". The third section contains fields for "Your Name", "Telephone (so we can berate you if you're wrong)", "Your Email address", and "Your Web address". A "SUBMIT REDEMPTION" button is at the bottom right. A validation error message "Please fill out this field." is displayed over the "Award Won" field.

OSCAR REDEMPTION

HERE'S YOUR CHANCE TO SET THE RECORD STRAIGHT: TELL US WHAT YEAR THE WRONG FILM GOT NOMINATED, AND WHICH FILM SHOULD HAVE RECEIVED A NOD...

About the offending film (part 1 of 3)

Year Of Crime

The film in question?

Award Won

Tell us why that's wrong?

How you rate it (1 is woeful, 10 is awesomesauce)

What should have won? (part 2 of 3)

The film that should have won?

Tell us why it should have won?

How you rate it (1 is woeful, 10 is awesomesauce)

About you? (part 3 of 3)

Your Name

Telephone (so we can berate you if you're wrong)

Your Email address

Your Web address

SUBMIT REDEMPTION

Рис. 8.3. Требование заполнить поле в браузере Firefox 9

Атрибут `required` можно применять наряду со многими типами полей ввода для гарантии того, что пользователь введет значение. Примечательными исключениями являются типы полей ввода `range`, `color`, `button` и `hidden`, поскольку они почти всегда содержат значение по умолчанию.

Следующим HTML5-атрибутом форм, который можно добавить в элементы `<input>`, является `autofocus`.

autofocus

HTML5-атрибут `autofocus` позволяет сделать так, чтобы в загружаемой форме то или иное поле ввода уже имело фокус (было выбрано), являясь готовым к вводу значения. Приведенный далее фрагмент кода — пример элемента `<input>`, который заключен в `<div>` и имеет атрибут `autofocus`:

```
<div>
  <label for="search">Search the site...</label>
  <input id="search" name="search" type="search" placeholder="Wyatt Earp" autofocus>
</div>
```

Будьте внимательны, используя этот атрибут. Возможно возникновение кросс-браузерной неразберихи, если добавить атрибут `autofocus` для нескольких полей ввода. Например, если сразу несколько элементов `<input>` включают атрибут `autofocus`, то при загрузке страницы в Chrome (версии 16) фокус получит последнее поле ввода, обозначенное этим атрибутом. Однако в Firefox (версии 9) в такой ситуации, наоборот, фокус получит первое поле ввода, располагающее атрибутом `autofocus`.

Следует также учитывать, что некоторые пользователи для прокрутки содержимого веб-страниц нажимают **Пробел**. На веб-странице, содержащей форму с полем ввода, которое обозначено атрибутом `autofocus`, эта функция будет недоступна. Вместо прокрутки добавится пробел в поле с фокусом. Легко догадаться, что это может не понравиться пользователям.

autocomplete

Большинство браузеров по умолчанию помогают пользователям вводить данные, по возможности поддерживая функцию автозаполнения полей соответствующими значениями. В то время как пользователи могут активизировать либо деактивизировать эту функциональность в своих браузерах, теперь мы также можем сообщить браузеру, что не хотим, чтобы форма или поле позволяло задействовать автозаполнение. Это целесообразно не только когда речь идет о конфиденциальных данных (например, о номерах банковских счетов), но и в ситуации, когда вам требуется обязательно обратить внимание пользователя на поле, попросив ввести что-то *вручную*. Например, во многих формах, которые мне доводится заполнять, если поле с телефонным номером обязательно для заполнения, я ввожу в него вымышленный телефонный номер. Я знаю, что не один так поступаю (разве не все так делают?), но смогу гарантировать, что пользователи не введут вымышленный

телефонный номер, если задам значение off для атрибута autocomplete соответствующего поля ввода. Приведенный далее фрагмент кода является примером поля ввода, для атрибута autocomplete которого задано значение off.

```
<div>
  <label for="tel">Telephone (so we can berate you if you're wrong)</label>
  <input id="tel" name="tel" type="tel" placeholder="1-234-546758" autocomplete="off"
    required aria-required="true" >
</div>
```

Мы также можем сделать так, чтобы автозаполнение не применялось во всех полях формы (а не только в тех, что заключены в элемент <fieldset>), задействовав атрибут autocomplete в отношении формы как таковой. Вот пример кода:

```
<form id="redemption" method="post" autocomplete="off">
```

list (и ассоциированный элемент <datalist>)

Атрибут list и ассоциированный элемент <datalist> позволяют предоставить пользователю несколько вариантов на выбор, когда он начинает вводить значение в соответствующем поле. Далее приведен пример кода, в котором атрибут list используется вместе с ассоциированным элементом <datalist>, заключенным в тег <div>:

```
<div>
  <label for="awardWon">Award Won</label>
  <input id="awardWon" name="awardWon" type="text" list="awards">
  <datalist id="awards">
    <select>
      <option value="Best Picture"></option>
      <option value="Best Director"></option>
      <option value="Best Adapted Screenplay"></option>
      <option value="Best Original Screenplay"></option>
    </select>
  </datalist>
</div>
```

Значение, заданное для атрибута list (awards), аналогично идентификатору <datalist>. Это позволяет ассоциировать <datalist> с соответствующим полем ввода. Хотя заключение всех тегов <option> в элемент <selection> не является строго обязательным, оно придется кстати, когда речь пойдет о применении полизаполнений для устаревших браузеров.

Несмотря на то что это поле ввода кажется лишь обычным текстовым полем, если начать печатать в нем, под ним появится окно выбора (в поддерживающих браузерах) с подходящими вариантами из списка <datalist>. На рис. 8.4 можно увидеть список в действии (в браузере Firefox 9). Поскольку буква В присутствует во всех вариантах в <datalist>, то в перечне отображаются все значения, из которых можно выбрать соответствующий вариант.

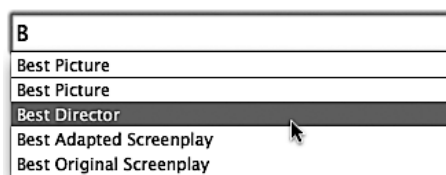


Рис. 8.4. При начале ввода появляется список подходящих значений для выбора

Однако если взамен ввести букву D, то на экране отобразятся только два подходящих варианта (рис. 8.5).

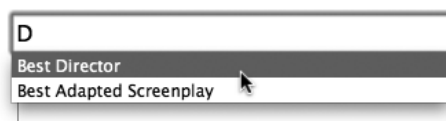


Рис. 8.5. Выводятся варианты значений, содержащих букву D

Это не мешает пользователям ввести в данное поле ввода что-нибудь еще по желанию, а задействованный подход — один из отличных способов добавления общей функциональности и улучшений с применением одной лишь разметки.

Типы полей ввода HTML5

В HTML5 включены несколько дополнительных типов полей ввода, позволяющих ограничивать данные, которые пользователи смогут ввести, без необходимости прибегать к лишнему JavaScript-коду. Самая отрадная черта этих новых типов полей состоит в том, что, если браузеры не поддерживают их, они по умолчанию превращаются в стандартные текстовые поля ввода. Более того, существуют отличные полизаполнения, позволяющие обеспечить поддержку требуемых функций в устаревших браузерах. Вскоре мы о них поговорим. А пока взглянем на новые типы полей ввода HTML5 и разберем их преимущества.

email

Браузеры, поддерживающие поле `type="email"`, будут ожидать, что пользователь введет данные, соответствующие синтаксису адресов электронной почты. В приведенном далее примере кода `type="email"` используется наряду с `required` и `placeholder`:

```
<div>
  <label for="email">Your Email address</label>
  <input id="email" name="email" type="email" placeholder=
    "dwight.schultz@gmail.com" required aria-required="true">
</div>
```

Если `type="email"` применяется в сочетании с атрибутом `required`, то попытка отправить форму с данными, не соответствующими требованиям, приведет к выводу на экран предупреждающего сообщения (рис. 8.6).

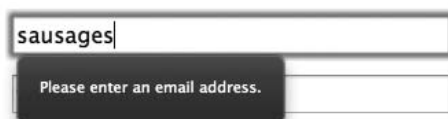


Рис. 8.6. В это поле следует ввести электронный адрес

Кроме того, на многих устройствах с сенсорными экранами (например, Android, iPhone и т. д.) внешний вид встроенной виртуальной клавиатуры изменяется исходя из типа поля ввода. На рис. 8.7 показано, как она будет выглядеть на экране iPad при использовании поля ввода `type="email"`. Обратите внимание на виртуальную клавишу с символом @, предназначенную для облегчения ввода адреса электронной почты.

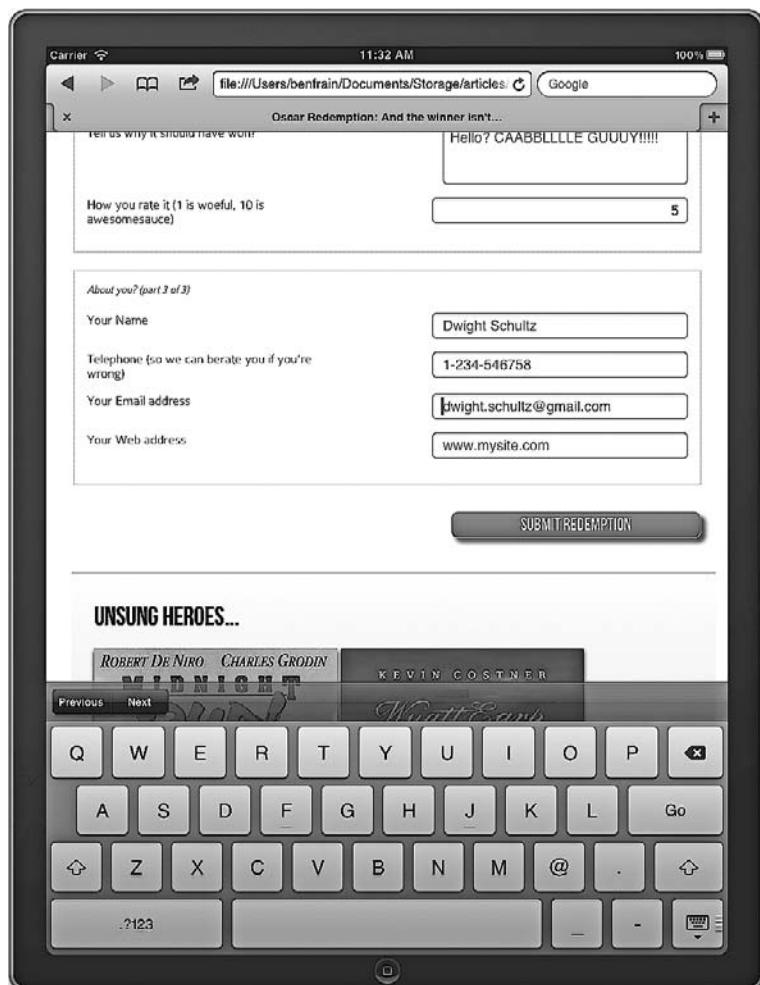


Рис. 8.7. Виртуальная клавиатура на экране iPad при использовании поля ввода `type="email"`

number

Браузеры, поддерживающие поле `type="number"`, ожидают, что пользователь введет в него числовое значение. Они также по умолчанию отображают элементы управления в виде *числовых полей ввода со значениями, изменяемыми с помощью мыши*, давая пользователям возможность менять определенное значение, просто нажимая располагающиеся рядом с ним стрелки «вверх/вниз». Вот пример кода:

```
<div>
  <label for="yearOfCrime">Year Of Crime</label>
  <input id="yearOfCrime" name="yearOfCrime" type="number" min="1929" max="2015"
    required aria-required="true" >
</div>
```

На рис. 8.8 показано, как результат будет выглядеть в поддерживающем это поле браузере (Chrome 16).



Рис. 8.8. Поле со счетчиком

Поведение, которое будет наблюдаться, если вы не введете число, варьируется. Например, Chrome (версии 16) очистит поле, как только оно потеряет фокус, не предусматривая при этом никакой обратной связи, в то время как Firefox (версии 9) позволит ввести все что угодно (поле по умолчанию будет работать как стандартное текстовое поле ввода). В предыдущем примере кода мы задали диапазон допустимых значений, указав минимальное и максимальные значения, как в следующем коде:

```
type="number" min="1929" max="2015"
```

Числа, не попадающие в этот диапазон, подвергаются (должны подвергаться) особой обработке. Реализация различается в зависимости от браузера. Например, Chrome (версии 16) выводит предупреждающее сообщение, а Firefox (версии 9) ничего не делает.

url

Как вы, возможно, и предполагали, поля ввода `type="url"` предназначены для указания URL-адресов. Подобно полям ввода, относящимся к типам `tel` и `email`, поведение рассматриваемого поля почти идентично тому, что наблюдается у стандартного текстового поля. Однако некоторые браузеры добавляют специфическую информацию в предупреждающие сообщения, выводимые на экран при попытке отправить форму с некорректными значениями. Далее приведен пример кода, включающего атрибут `placeholder`:

```
<div>
  <label for="web">Your Web address</label>
  <input id="web" name="web" type="url" placeholder="www.mysite.com">
</div>
```

На рис. 8.9 показано, что произойдет при попытке отправить форму, в соответствующем поле которой неправильно введен URL-адрес, в браузере Chrome (версии 16).



Рис. 8.9. Требуется ввести корректный электронный адрес

Как и в случае с `type="email"`, на устройствах с сенсорными экранами внешний вид встроенной виртуальной клавиатуры зачастую изменяется исходя из типа поля ввода. На рис. 8.10 показано, как она будет выглядеть на экране iPad при использовании поля ввода `type="url"`.



Рис. 8.10. Виртуальная клавиатура на экране iPad при использовании поля ввода `type="url"`

Заметили виртуальные клавиши Go, прямой слеш (/) и .com? Поскольку мы использовали тип поля ввода url, они отображаются на экране для облегчения набора URL-адреса (если только вы не собираетесь указать адрес сайта, который не заканчивается на .com, в случае чего, как вы сами понимаете, благодарить компанию Apple будет не за что).

tel

type="tel" — еще один тип поля ввода, предназначенного для указания контактной информации. Поле типа tel применяется для того, чтобы сообщить браузеру, что в соответствующем поле формы пользователь должен ввести телефонный номер. Вот пример кода:

```
<div>
  <label for="tel">Telephone (so we can berate you if you're wrong)</label>
  <input id="tel" name="tel" type="tel" placeholder="1-234-546758" autocomplete="off"
    required aria-required="true" >
</div>
```

Несмотря на то что предполагается числовой формат вводимых данных, во многих браузерах, даже современных, вроде Chrome 16 и Firefox 9, это поле будет попросту вести себя как текстовое поле ввода. На текущий момент в этих браузерах не предусматривается вывод надлежащих предупреждающих сообщений при попытке отправить форму с некорректными введенными значениями.

Однако, как и в случае с типами полей ввода email и url, устройства с сенсорными экранами зачастую внимательно подходят к типу поля ввода tel, обеспечивая отображение измененной соответствующим образом встроенной виртуальной клавиатуры для облегчения ввода информации. На рис. 8.11 показано поле ввода типа tel при доступе к нему на экране iPad (под управлением операционной системы iOS 5).

Вы заметили, что на встроенной виртуальной клавиатуре отсутствуют знаки алфавита? Благодаря этому пользователи смогут намного быстрее ввести значение в корректном формате.

search

Несмотря на то что поле ввода type="search" работает таким же образом, как и стандартное текстовое поле ввода, некоторые браузеры немного по-разному выполняют визуализацию на основе одного и того же кода. Вот пример кода:

```
<div>
  <label for="search">Search the site...</label>
  <input id="search" name="search" type="search" placeholder="Wyatt Earp">
</div>
```

На рис. 8.12 показано, как будет выглядеть результат применения этого кода в браузере Firefox (версии 9). Обратите внимание, что при стилизации по умолчанию поле ввода является прямоугольным.

Однако в браузере Chrome (версии 16) визуализация на основе того же самого кода получается другой — поле ввода по умолчанию имеет закругленные края и кнопку быстрой очистки справа (рис. 8.13).

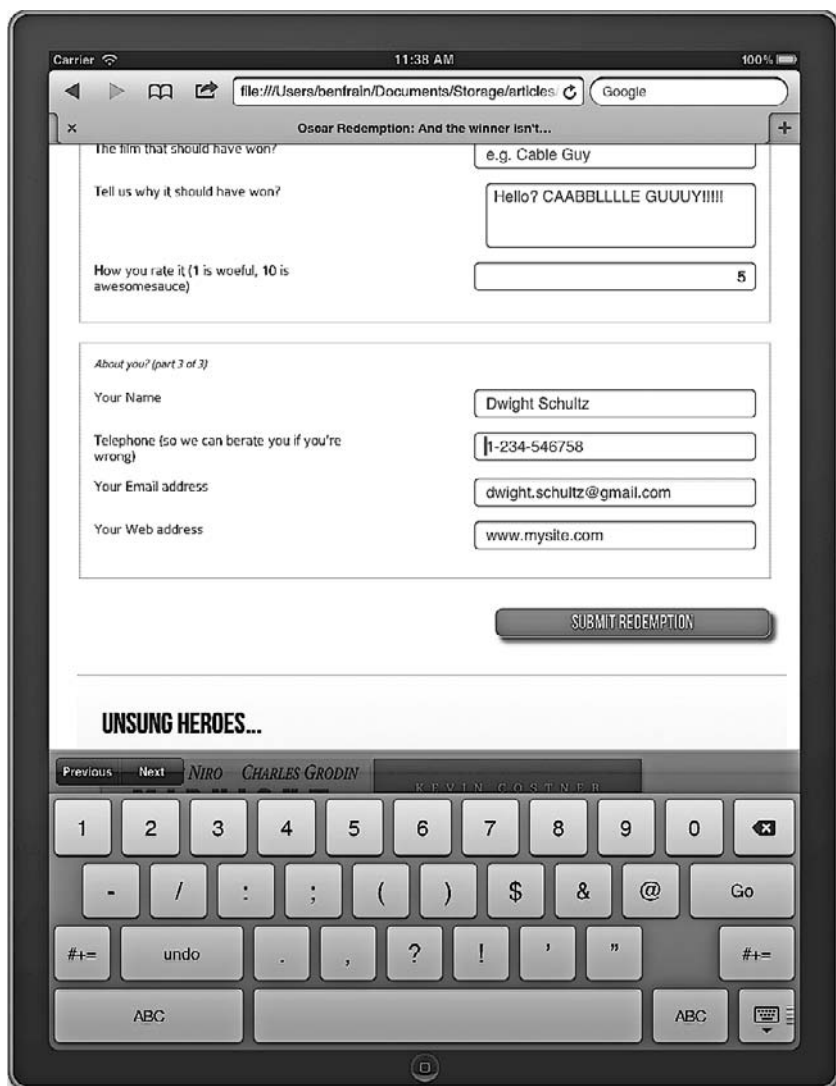


Рис. 8.11. Пример поля ввода tel на экране iPad

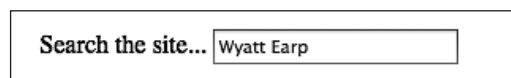


Рис. 8.12. Поле поиска в Firefox

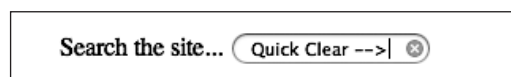


Рис. 8.13. Поле поиска в Chrome

pattern

Говоря об атрибуте `pattern=""`, сразу хочется предупредить: *«Бойтесь, очень бойтесь»* (помните, из какого фильма эта реплика?). На мой взгляд, эта реплика с тем же успехом применима к **регулярным выражениям**. Если вы не знаете, что такое регулярные выражения, то смею заметить, меньше знаешь — крепче спишь. Но если знаете, а еще хуже, понимаете их, то следующий раздел — для вас.



УЗНАЙТЕ О РЕГУЛЯРНЫХ ВЫРАЖЕНИЯХ

Если вы смотрели фильм «Изгоняющий дьявола» (The Exorcist) в одиночку, на кладбище, в полночь, во время Хеллоуина, то, вероятно, готовы узнать о регулярных выражениях. Читайте о них по адресу http://en.wikipedia.org/wiki/Regular_expressions.

Атрибут `pattern` позволяет определять с помощью регулярного выражения синтаксис данных, ввод которых должен быть разрешен в определенном поле. Вот пример кода:

```
<div>
  <label for="name">Your Name (first and last)</label>
  <input id="name" name="name" pattern="([a-zA-Z]{3,30}\s*)+[a-zA-Z]{3,30}"
    placeholder="Dwight Schultz" required ariarequired="true" >
</div>
```

При написании книги я потратил примерно 458 секунд на поиск в Интернете регулярного выражения, которое соответствовало бы синтаксису имени и фамилии. Если указать регулярное выражение в виде значения атрибута `pattern`, то поддерживающие браузеры будут ожидать ввода соответствующего синтаксиса. Кроме того, при использовании этого атрибута в сочетании с атрибутом `required` некорректно введенные значения будут подвергаться показанной далее обработке в поддерживающих их браузерах. В данном случае я попытался отправить форму, не указав фамилию (рис. 8.14).

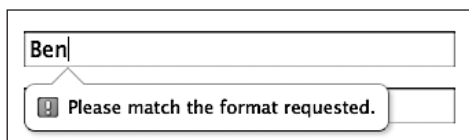


Рис. 8.14. Выводится предупреждение о некорректном вводе данных

color

Поле ввода `type="color"` генерирует палитры цветов в поддерживающих браузерах, давая пользователям возможность выбирать значения цветов в шестнадцатеричном формате. Вот пример кода:

```
<div>
  <label for="color">Your favorite color</label>
  <input id="color" name="color" type="color">
</div>
```

К сожалению, в настоящее время очень немногие браузеры поддерживают тип поля ввода `color`. Похоже, что только Opera (версии 11) обеспечивает палитру

цветов. Если требуемый цвет не отображается на палитре изначально, то, нажав располагающуюся внизу кнопку **Other** (Другие), можно открыть палитру цветов, используемую в операционной системе по умолчанию (рис. 8.15).

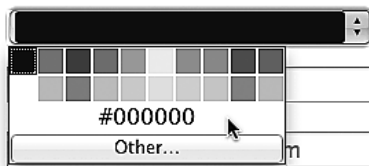


Рис. 8.15. Отображение палитры цветов

Типы полей ввода date и time

Согласно задумке, новые типы полей ввода `date` и `time` призваны обеспечить пользователю удобный выбор значений дат и времени. Если вам когда-либо доводилось приобретать в Интернете билеты, например, на спортивные соревнования, то вы, скорее всего, пользовались при этом элементом управления в виде календаря того или иного рода. Для обеспечения такой функциональности почти всегда применяется JavaScript (обычно это библиотека jQuery), однако есть надежда, что это стало возможным с использованием одной лишь HTML5-разметки.

date

Вот пример кода:

```
<input id="date" type="date" name="date" />
```

Как и в случае с полем ввода типа `color`, в настоящее время немногие браузеры поддерживают поле типа `date`, и в большинстве из них поля такого типа по умолчанию работают как стандартные текстовые поля ввода. Однако в старом добром браузере Opera эта функциональность уже реализована, а на рис. 8.16 показано, как в Opera (версии 11) будет выглядеть результат визуализации на основе приведенного выше примера кода.



Рис. 8.16. Календарь в браузере Opera

Существует несколько разных «модификаций» полей ввода типов `date` и `time`. Далее приведен их краткий обзор.

month

Вот пример кода:

```
<input id="month" type="month" name="month">
```

Интерфейс позволит пользователю выбрать один месяц, после чего обеспечит ввод данных в виде года и месяца, например 2012-06.

На рис. 8.17 показано, как все это будет выглядеть в браузере.



Рис. 8.17. Пользователь может выбрать в календаре целый месяц

week

Вот пример кода:

```
<input id="week" type="week" name="week">
```

При использовании типа поля ввода `week` соответствующий инструмент-указатель позволит пользователю выбрать одну неделю в году, после чего обеспечит ввод данных в формате, например, 2012-W47.

На рис. 8.18 показано, как все это будет выглядеть в браузере.

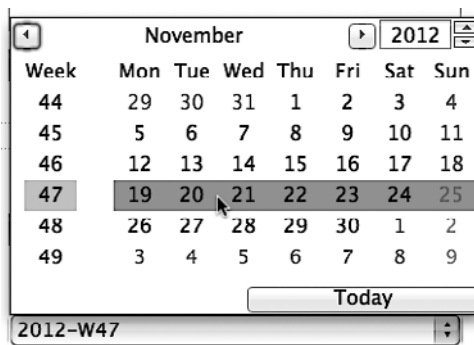


Рис. 8.18. Пользователь выбирает в календаре всю неделю

time

Вот пример кода:

```
<input id="time" type="time" name="time">
```

Поле ввода, относящееся к типу `time`, допускает ввод значений в 24-часовом формате, например 23:50. В поддерживающих браузерах оно отображается как элемент управления в виде числового поля ввода со значением, изменяемым с помощью мыши, и допускает ввод только значений времени (рис. 8.19).



Рис. 8.19. Поле для ввода значения времени

datetime и datetime-local

Вот пример кода:

```
<input id="datetime" type="datetime" name="datetime">
```

На рис. 8.20 показано, как результат выполнения этого кода будет выглядеть в браузере Opera (версии 11).



Рис. 8.20. На календаре выбран текущий день (на момент выполнения кода)

На экране устройств под управлением операционной системы iOS все это будет выглядеть еще лучше, о чем свидетельствует рис. 8.21.

При вводе данных в поле типа `datetime` генерируются значения даты и времени (разделенные буквой T), а затем помечается часовой пояс (буквой Z в случае с UTC (Universal Coordinated Time — универсальное координированное время) либо знаком «+» или «-», если речь идет о величинах смещения). 25 октября 2009 года в системе UTC будет выглядеть так:

```
2009-10-25T05:05:00Z
```

Поскольку UTC является практически эквивалентом GMT (Greenwich Mean Time — среднее время по Гринвичу), разобраться в смещениях не составит труда.

Например, тихоокеанское стандартное время (Лос-Анджелес) на 8 часов отстает от GMT (UTC – 8 часов). Это нашло бы свое отражение во вводимом значении, как показано далее:

2009-10-25T05:05:00-8:00

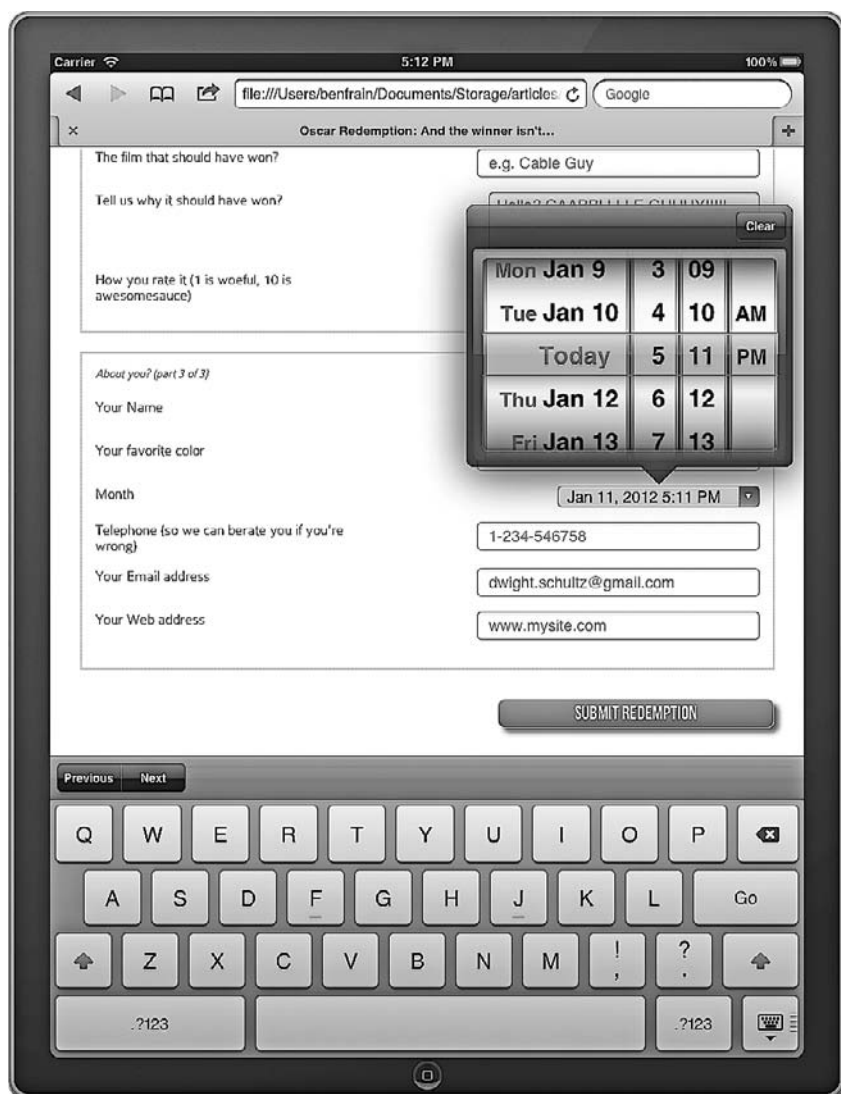


Рис. 8.21. Результат выполнения того же кода на экране iPad

Версия `datetime-local` работает точно так же, как и `datetime`, однако не включает информацию о часовом поясе.



ИЗМЕНЕНИЕ ШАГОВЫХ ПРИРАЩЕНИЙ

При работе с элементами управления в виде числовых полей ввода различных типов со значениями, изменяемыми с помощью мыши, вы можете менять шаговые приращения (величину инкремента), используя для этого атрибут `step`. Например, для шага величиной 4 часа за один раз вам потребуется ввести значение, равное 4 часам и выраженное в виде 14 400 секунд (60 (секунд) умножить на 60 (минут) и умножить на 4 (часа)). Далее приведен пример с `datetime`, измененный для использования 4-часовых шагов во временном селекторе:

```
<input id="datetime" type="datetime" name="datetime" step="14400">
```

range

Поле ввода типа `range` позволит создать такой элемент интерфейса, как ползунок. Вот пример кода:

```
<input id="howYouRateIt" name="howYouRateIt" type="range" min="1" max="10" value="5" >
```

На рис. 8.22 показано, как результат будет выглядеть в браузере Safari (версии 5.1).



Рис. 8.22. Ползунок в браузере Safari

Диапазон по умолчанию — от 0 до 100. Однако, указав значения для `min` и `max` в предыдущем примере, мы ограничили диапазон, и теперь он составляет от 1 до 10.

Серьезная проблема, с которой я столкнулся при работе с полем ввода типа `range`, заключается в том, что для пользователя никогда не отображается текущее значение. Хотя ползунок предназначен только для выбора числовых значений в некоем диапазоне, мне зачастую требовалось, чтобы текущее значение отображалось во время его изменения. На данный момент сделать это с помощью HTML5 невозможно. Однако если вам потребуется обязательно обеспечить отображение текущего значения, выбранного ползунком, то вы сможете легко сделать это, используя незамысловатый JavaScript-код. Измените предыдущий пример кода следующим образом:

```
<input id="howYouRateIt" name="howYouRateIt" type="range" min="1" max="10" value="5"
onchange="showValue(this.value)"><span id="range">5</span>
```

Здесь мы добавили атрибут `onchange` и элемент `` с идентификатором `range`. А теперь добавим следующий небольшой JavaScript-фрагмент:

```
<script>
function showValue(newValue)
{
    document.getElementById("range").innerHTML=newValue;
}
</script>
```

Этот код лишь извлекает текущее значение, выбранное с помощью ползунка, и отображает его в элементе с идентификатором `range` (в нашем теге ``). Используем также немного **CSS-стилизации, чтобы сделать значение крупнее и задать для него красный цвет**. На рис. 8.23 демонстрируется итоговый эффект, при котором значение будет обновляться по мере движения ползунка.



Рис. 8.23. При движении ползунка отображается текущее значение



ПРИМЕЧАНИЕ

В HTML5 имеются и другие новые свойства, касающиеся форм, однако, поскольку они относятся больше к созданию приложений и разработке серверных программ, мы не рассматривали их здесь. Изучить редакторский черновик W3C раздела о HTML5-формах вы сможете по следующему адресу: <http://dev.w3.org/html5/spec-author-view/forms.html#forms>.

8.2. Добавление полизаполнений для браузеров, не поддерживающих требуемые функции

HTML5-формы — это, конечно, замечательно. Однако, похоже, есть кое-что, серьезно ограничивающее наши возможности по их использованию: различия в том, как поддерживающие браузеры реализуют определенные функции. Кроме того, непонятно, как поступать с браузерами, которые эти функции вообще не поддерживают. К счастью, как и всегда, веб-сообщество смогло найти выход из сложившейся ситуации.

Еще в главе 4 я упоминал о Modernizr (<http://www.modernizr.com>) — потрясающей JavaScript-библиотеке, которая позволяет добавлять **полизаполнения** для браузеров, не поддерживающих требуемые функции HTML5/CSS3. Библиотека Webshims Lib, написанная Александром Фаркасом (Alexander Farkas) (<http://afarkas.github.com/webshim/demos/>), базируется на Modernizr и вездесущей jQuery и **дает возможность загружать полизаполнения, которые касаются форм (она также позволяет добавлять полизаполнения, относящиеся к другим HTML5-функциям) и необходимы для того, чтобы браузеры, не поддерживающие требуемые функции, смогли обрабатывать наши HTML5-формы**. Поскольку библиотека Webshims Lib использует загрузочные возможности Modernizr, то соответствующие полизаполнения добавляются только при необходимости, что очень удобно. Она совсем немного «утяжеляет» веб-страницы, если они просматриваются в браузере, поддерживающем определенные HTML5-функции. Несмотря на то что устаревшим браузерам приходится загружать больше кода (поскольку они заведомо являются менее совместимыми), они предусматривают схожее пользовательское взаимодействие,

хотя для обеспечения соответствующей функциональности прибегают к помощи JavaScript.

Однако библиотека Webshims Lib полезна, не только когда речь идет об устаревших браузерах. Как мы уже видели, многие современные браузеры не полностью реализуют функции, касающиеся HTML5-форм. Задействуя Webshims Lib для конкретной веб-страницы, мы устраняем любые пробелы и в их функциональных возможностях. Например, браузер Safari (версии 5.1) не выводит предупреждающих сообщений, если пользователь пытается отправить данные HTML5-формы, не заполнив какие-либо поля. Хотя отправки данных на самом деле не происходит, браузер не сообщает пользователю, в чем заключается проблема — едва ли такой подход можно назвать идеальным. Вот что произойдет при описанном выше сценарии, если внедрить в веб-страницу Webshims Lib (рис. 8.24).



Рис. 8.24. При некорректном вводе данных выводится предупреждение

Таким образом, если браузер Firefox (версии 9) не сможет обеспечить работу числового поля ввода со значением, изменяемым с помощью мыши, при использовании атрибута `type="number"`, то Webshims Lib позаботится о подходящем резервном варианте, «приводимом в действие» библиотекой jQuery. Webshims Lib — отличный инструмент, поэтому установим и «подключим» замечательный небольшой пакет, который его содержит. В результате мы сможем продолжить создавать HTML5-формы, пребывая в полной уверенности, что посетители сайта увидят все необходимое для того, чтобы пользоваться нашими формами (за исключением тех двух человек, которые все еще используют браузер Internet Explorer 6 с отключенным JavaScript).

Сначала скачайте пакет с библиотекой Webshims Lib (<http://github.com/aFarkas/webshim/downloads>) и распакуйте его. Затем скопируйте папку `js-webshim` в соответствующий каталог, относящийся к вашей веб-странице. Для простоты в этом примере я скопировал ее в корневой каталог сайта.

Далее добавьте следующий код в тег `<head>` своей страницы:

```
<script src="js/jquery-1.7.1.js"></script>
<script src="js-webshim/minified/extras/modernizr-custom.js"></script>
<script src="js-webshim/minified/polyfiller.js"></script>
<script>
  // загрузить все указанное
  $.webshims.polyfill();
</script>
```

Сразу разберемся в этом коде. Сначала я указал ссылку на локальную копию библиотеки jQuery (ее последнюю версию можно скачать по адресу www.jquery.com):

```
<script src="js/jquery-1.7.1.js"></script>
```

Затем я добавил соответствующие версии JavaScript-файлов `modernizr-custom.js` и `polyfiller.js`, которые имеются в Webshims Lib:

```
<script src="js-webshim/minified/extras/modernizr-custom.js"></script>
<script src="js-webshim/minified/polyfiller.js"></script>
```

И наконец, я дал указание сценарию загрузить все необходимые полизаполнения:

```
<script>
  // загрузить все указанное
  $.webshims.polyfill();
</script>
```

Вот и все. Теперь отсутствующая функциональность будет автоматически добавляться соответствующим полизаполнением. Отлично!

8.3. Стилизация HTML5-форм с помощью CSS3

Сейчас наша форма стала полностью функциональной во всех браузерах, и, хотя мы снабдили ее небольшой базовой стилизацией, нам с вами известно, что с помощью CSS3 можно гораздо лучше стилизовать ее. Чтобы немного приукрасить нашу форму, применим отдельные методики, которые мы изучили и использовали ранее. Далее приведены все характерные для форм стили, которые имеются у нас на текущий момент:

```
#redemption {
  width: 100%;
  font-family: 'ColaborateThinRegular';
  font-weight: 400;
}
#redemption hgroup {
  margin-bottom: 20px;
}
#redemption div {
  width: 100%;
  margin-bottom: 15px;
  float: left;
}
#redemption span#range {
  float: left;
  font-size: 3em;
  width: 100%;
  color: red;
  clear: both;
  text-align: center;
}
#howYouRateThis, #yearOfCrime {
```

```

    text-align: right;
}
#redemption legend {
    font-style: italic;
    color: #434242;
    font-size: 0.8em;
    margin-bottom: 20px;
    float: left;
    width: 100%;
}
#redemption fieldset {
    border: 1px dotted #cccccc;
    padding: 2%;
    margin-bottom: 20px;
}
#redemption label {
    width: 40%;
    float: left;
}
#redemption input {
    height: 20px;
    font-size: 1em;
    width: 40%;
    float: right;
}
#redemption textarea {
    height: 60px;
    font-size: 1em;
    width: 40%;
    float: right;
}
#redemption input#submit {
    text-decoration: none;
    height: 34px;
    font: 1.25em /* 36px × 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
    border-radius: 8px;
    color: white;
    float: right;
    margin-bottom: 10px;
    background: linear-gradient(top, rgb(241,92,96) 0%, rgb(176,28,32) 100%);
    margin-top: 10px;
    box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
    text-shadow: 0px 1px black;
    border: 1px solid #bfbfbf;
}
.polyfill-important .input-range,.polyfill-important .step-controls {
    float: right;
}

```

```
.polyfill-important .step-controls {
  margin-right: -20px!important;
}
```

Здесь следует отметить, что два последних стиля актуальны, только когда загружаются некоторые из полизаполнений.

Прежде всего я хочу сделать так, чтобы каждый элемент fieldset сильнее выделялся благодаря утонченному градиентному фону. Далее приведен измененный CSS-код для fieldset:

```
#redemption fieldset {
  border: 1px dotted #cccccc;
  padding: 2%;
  margin-bottom: 20px;
  background: #ffffff;
  background: linear-gradient(top, #ffffff 77%, #f2f2f2 100%);
  border-radius: 4px;
  box-shadow: 2px 2px 5px hsla(0, 0%, 16.6667%, 0.3);
}
```

Мы использовали border-radius и background для задания фонового градиента, а также добавили объявление box-shadow для обеспечения небольшой тени, отбрасываемой блочным элементом.

Как и во многих приводившихся ранее примерах, я не стал включать версии CSS3-объявлений с префиксами поставщиков.

На рис. 8.25 показано, как результат будет выглядеть в браузере Chrome.

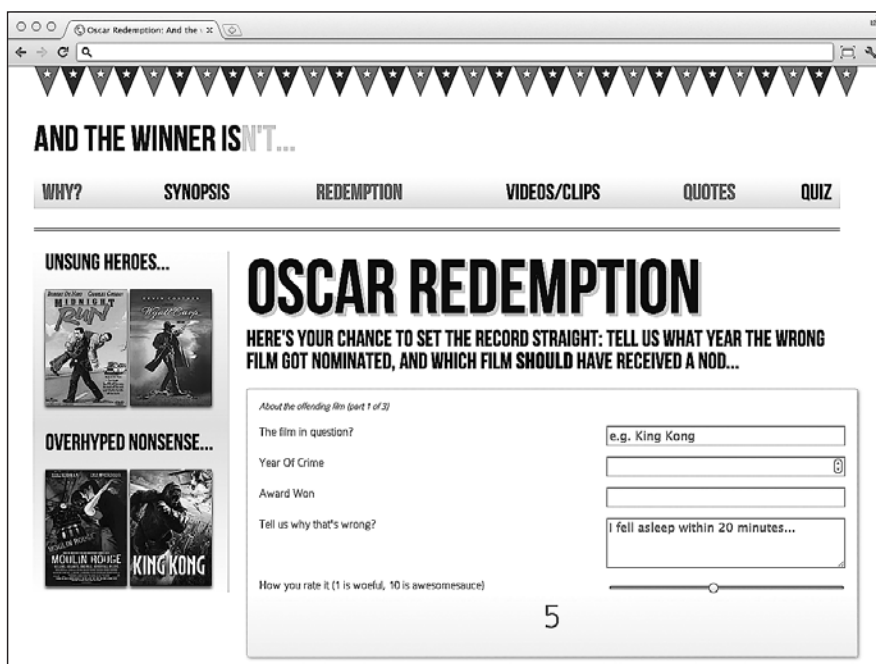


Рис. 8.25. Для формы заданы фоновый градиент и небольшая тень



КОМБИНИРОВАНИЕ РАЗНЫХ ФОРМАТОВ ЗНАЧЕНИЙ ЦВЕТОВ

Как видно из приведенных примеров, я по-разному определял цвета. В некоторых случаях я использовал такие значения, как, например, red, однако применял и шестнадцатеричные, RGB и HSL-значения. В поддерживающих браузерах это не приведет ни к каким отрицательным последствиям. Однако при написании кода для сайтов вы, возможно, предпочтете придерживаться одного или двух форматов.

Пока все хорошо. Но наши текстовые поля ввода все равно выглядят несколько уныло. Добавим для них немного CSS3-кода:

```
input, textarea, select {
  border: 1px solid #bfbfbf;
  padding: 0.2em;
  font-size: 1.1em;
  line-height: 1.2em;
  background: #ffffff;
  background: linear-gradient(top, #ffffff 0%, #ededed 8%, #ffffff 100%);
  border-radius: 4px;
  box-shadow: 2px 2px 5px hsla(0, 0%, 16.6667%, 0.1);
}
```

Опять-таки здесь у нас есть свойство background для задания фонового градиента, border-radius для незначительного скругления углов и box-shadow для обеспечения небольшой тени, отбрасываемой блочным элементом. На рис. 8.26 показано, как результат будет выглядеть в браузере Chrome.

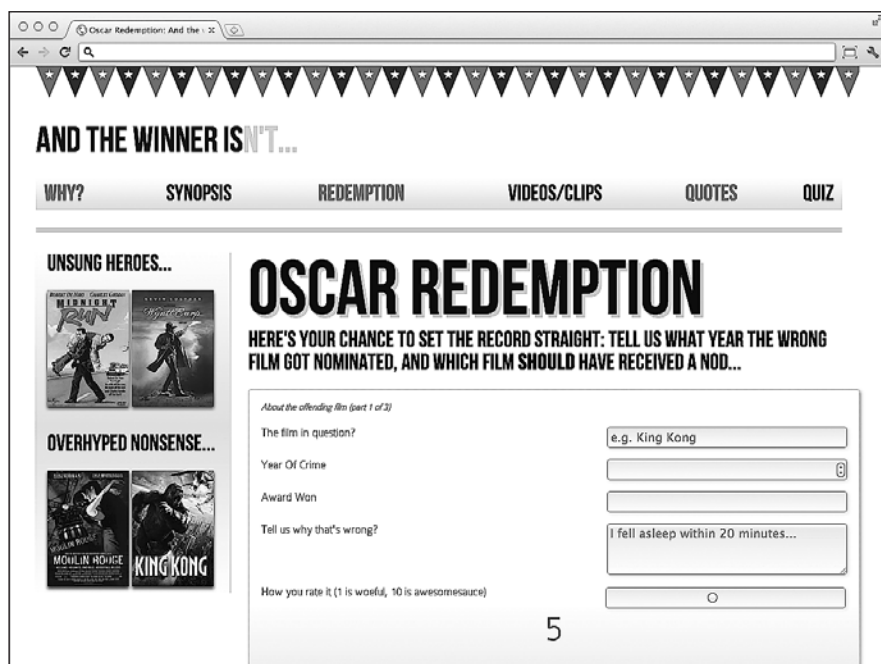


Рис. 8.26. У полей появились градиентный фон, скругленные углы и небольшая тень

Такой результат меня устраивает... О, постойте-ка. Взгляните на ползунок внизу страницы. Это не то, что я хотел. Мне не нужно, чтобы приведенные выше правила влияли на ползунок, поэтому я пересмотрю стили и задействую один из новых CSS3-селекторов для решения возникшей проблемы:

```
input:not([type="range"]), textarea, select{
  /* стили */
}
```

Я использовал псевдоселектор `:not`, благодаря которому указал, что не хочу, чтобы правило применялось к полям ввода с атрибутом `type="range"`. Снова взглянем на нашу страницу в браузере Chrome (рис. 8.27).

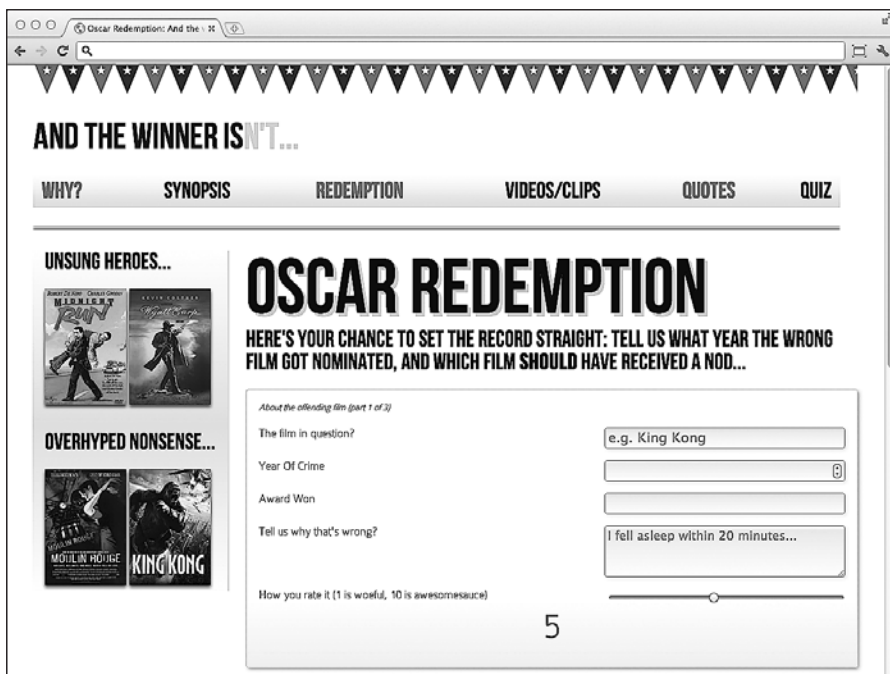


Рис. 8.27. К ползунку теперь не применяются стили, установленные для полей

Отлично! Именно к такому результату я и стремился, а благодаря CSS3 можно с легкостью не только добавлять соответствующие стили, но и предотвращать их применение к элементам, для которых эти стили будут нежелательными.

Псевдоселекторы классов CSS3, специфичные для форм. Наряду со всеми интересными CSS3-инструментами, о которых мы уже говорили, существует несколько специфичных для форм псевдоселекторов:

- `input:required` — предназначен для полей, обязательных для заполнения;
- `input:focus:invalid` — используется для полей с установленным фокусом, в которых введено недопустимое значение;
- `input:focus:valid` — применяется для полей с установленным фокусом, в которых введено допустимое значение.

Итак, используем эти псевдоселекторы при создании трех дополнительных правил стилей, как показано в следующих примерах кода:

```
input:required {  
    border: 1px solid rgba(253, 8, 8, 0.29);  
}  
input:focus:invalid {  
    background: url('../img/cross.png') no-repeat right;  
    padding-right: 3px;  
}  
input:focus:valid {  
    background: url('../img/tick.png') no-repeat right;  
    padding-right: 3px;  
}
```

Первое правило обеспечивает тонкую границу вокруг обязательных для заполнения полей, второе добавляет изображение красного крестика для применения в ситуациях, когда пользователь, начав печатать, введет некорректное значение, а третье правило добавляет изображение зеленого флажка, который будет свидетельствовать о корректности введенного значения.

На рис. 8.28 показано, как все будет выглядеть в браузере (Firefox 9), когда страница загрузится.

Рис. 8.28. Форма, стилизованная с использованием псевдоселекторов

Теперь, если мы установим фокус (щелчком кнопкой мыши) на одном из обязательных для заполнения полей, появится **красный крестик** (поскольку мы еще не ввели допустимое значение) (рис. 8.29).

Рис. 8.29. При установке фокуса в поле ввода появляется красный крестик

Если мы пойдем дальше и введем допустимое значение, то изображение **красного крестика** сменится на изображение **зеленого флажка** (рис. 8.30).

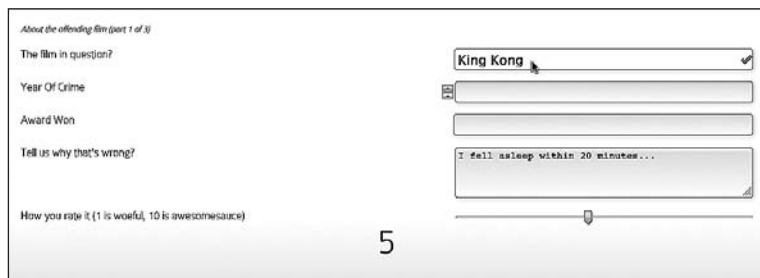


Рис. 8.30. При вводе допустимого значения в поле появляется зеленый флажок

Новые псевдоселекторы классов CSS3 помогают создать тонкий, легкий в реализации набор улучшений, который повышает качество общего пользовательского взаимодействия, обеспечиваемого при заполнении форм.

8.4. Резюме

В этой главе мы научились использовать немало новых HTML5-атрибутов форм. Они дают нам возможность сделать формы более удобными в применении, чем когда-либо прежде, а вводимые в них данные — более соответствующими определенным критериям. Кроме того, мы можем сделать так, что разметка, в которой применяются эти новые HTML5-атрибуты, и со временем не потеряет актуальности. Для этого можно прибегнуть к условной загрузке JavaScript-сценариев, являющихся полизаполнениями, чтобы для всех пользователей был доступен схожий набор элементов форм независимо от функционала их браузеров.

Мы подходим к завершению нашего путешествия в мир адаптивного веб-дизайна с использованием HTML5 и CSS3. Мы охватили множество теоретических аспектов при рассмотрении практического примера в виде сайта And the winner isn't... Однако при реализации адаптивных веб-дизайнов в реальном мире часто возникают дополнительные трудности. Как уместить множество навигационных ссылок на небольшом экране? Как обеспечить загрузку дополнительных файлов только в тех браузерах, которые в них нуждаются? В последней главе мы рассмотрим большинство из таких распространенных проблем (и их решения), возникающих при реализации адаптивных веб-дизайнов, созданных с применением HTML5 и CSS3. Мы также снова поговорим о наиболее оптимальных способах устранения некоторых недостатков распространенных устаревших браузеров.

9 Решение кросс-браузерных проблем с адаптивностью

В последней главе мы поговорим о следующем:

- в чем состоит принципиальная разница между прогрессивным улучшением и плавным сокращением возможностей;
- как обеспечить поддержку адаптивных веб-дизайнов в устаревших версиях браузера Internet Explorer;
- как использовать Modernizr для условной загрузки CSS-файлов и полизаполнений JavaScript;
- как преобразовать длинные списки навигационных ссылок в раскрывающиеся меню в небольших по размеру областях просмотра;
- как обеспечить изображения для устройств с экранами высокого разрешения (Retina).

Прежде чем мы приступим к изучению материала этой главы, обобщим, на каком этапе мы находимся и что уже знаем.

Количество мобильных пользователей растет взрывными темпами. Соответственно, сайты просматриваются в разных (по размеру и ориентации) областях просмотра с применением разных по скорости каналов подключения. В обозримом будущем при дизайне и создании сайтов нам придется начинать с основного содержимого, а затем поступательно «наслаивать» функции и улучшения. Кроме того, принимая во внимание разницу в скорости каналов подключения пользователей, нужно учесть, что кодовая база должна быть как можно более компактной и гибкой.

Продуманно подходя к дизайну нашего сайта, мы максимально использовали методологию адаптивного дизайна, разработанную Итаном Маркоттом. Мы применили **CSS3-медиазапросы (рассмотренные в главе 2)** для создания контрольных точек в дизайне, по достижении которых макет эффектно адаптируется к соответствующей области просмотра. Далее мы задействовали «резиновые»

изображения и мультимедиа наряду с «резиновыми» сетками (в главе 3), которые используются для обеспечения плавных переходов между контрольными точками медиазапросов. В результате у нас получился дизайн, который будет хорошо смотреться не только в распространенных сегодня, но и в будущих областях просмотра.

Чтобы сохранить компактность кодовой базы, в главе 4 мы переключили нашу разметку на использование HTML5. Эта версия обеспечивает эффект экономии, более семантически значимый код и делает возможными такие вещи, как, например, просмотр сайтов в автономном режиме. Двигаясь далее, мы обеспечили большую доступность нашего сайта с помощью WAI-ARIA, позаботившись о наличии дополнительных средств для пользователей экранных дикторов и вспомогательных технологий.

В главах 5 и 6 мы оценили невероятную мощь и гибкость CSS3, изучая новые цветовые режимы RGBA и HSLA, а также выяснили, как можно обеспечить такие «украшательства» в дизайне, как тени, отбрасываемые блочными элементами и текстом, фоновые градиенты и т. д., используя для этого не изображения, а лишь CSS3. Кроме того, мощные CSS-селекторы позволили нам выбирать все необходимое из объектной модели документа (DOM) с эффективностью, ранее возможной только благодаря JavaScript. Однако CSS3 позволяет нам не только адаптировать дизайны и значительно снизить требования к скорости канала подключения пользователя при просмотре наших сайтов. Эта версия также привносит функциональность, которой никогда ранее нельзя было насладиться без привлечения Flash или JavaScript — возможность пользовательского шрифтового оформления (см. главу 5) и создания красивых плавных переходов (см. главу 7) между разными визуальными состояниями. Всматриваясь в будущее, мы также мельком взглянули на такие замысловатые вещи, как 3D-трансформации CSS3.

И наконец, в главе 8 мы поговорили о решении рутинной задачи по созданию форм, поспрашивали возможность легко справиться с тяжелой работой, связанной с валидацией форм, а также обсудили способы создания элементов интерфейса форм с помощью HTML5-разметки. Важно и то, что мы добавили резервный JavaScript-вариант, чтобы обеспечивать качественное взаимодействие в тех ситуациях, когда используются такие устаревшие браузеры, как Internet Explorer версий 6, 7 и 8.

По ходу этой книги, применяя HTML и CSS3, мы создали довольно простой адаптивный сайт под названием And the winner isn't... Вы можете посетить его в Интернете, введя в своем браузере адрес <http://www.andthewinnerisnt.com>.

На рис. 9.1 показано, как на текущий момент выглядит первая страница нашего сайта на экране iPhone.

На рис. 9.2 демонстрируется, как первая страница нашего сайта выглядит на экране iPad.

А на рис. 9.3 показано, как она выглядит в браузере Android (эмуляторе).

На рис. 9.4 можно увидеть, как первая страница нашего сайта выглядит в современном настольном браузере (Google Chrome 16).

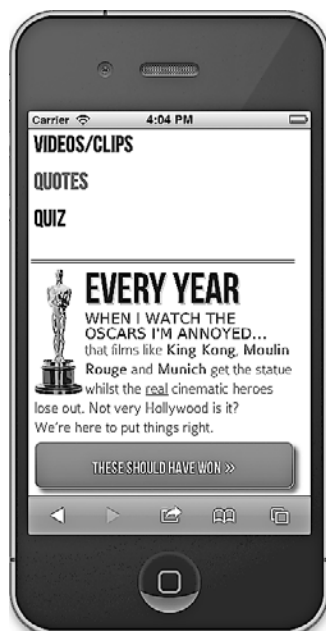


Рис. 9.1. Главная страница сайта And the winner isn't... на экране iPhone

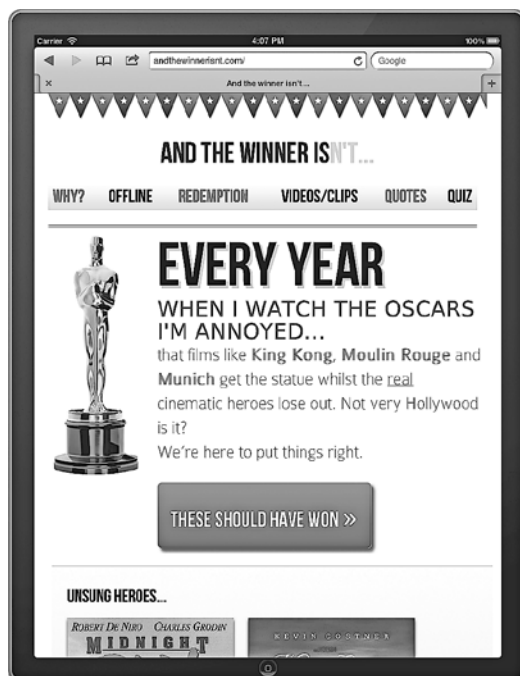


Рис. 9.2. Главная страница сайта And the winner isn't... на экране iPad

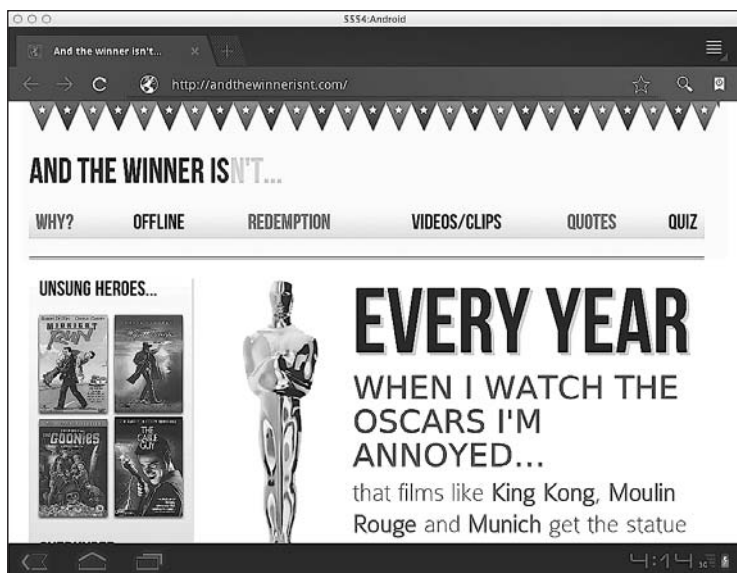


Рис. 9.3. Главная страница сайта And the winner isn't... в браузере Android

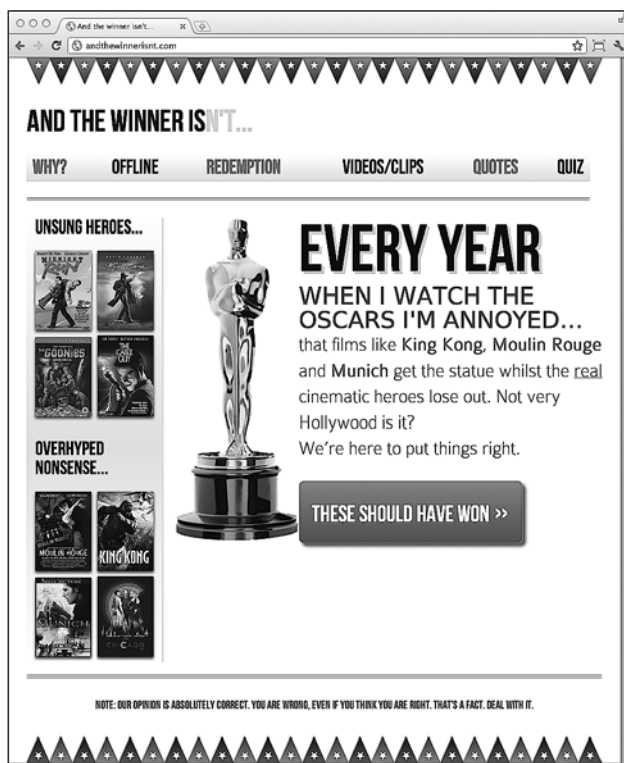


Рис. 9.4. Главная страница сайта And the winner isn't... в браузере Google Chrome 16

И наконец, на рис. 9.5 показано, как она в настоящий момент выглядит в браузере Internet Explorer 8.



Рис. 9.5. Главная страница сайта And the winner isn't... в браузере Internet Explorer 8

О, какой ужас! Дайте мне служебный револьвер...

Вид первой страницы нашего сайта в браузере Internet Explorer 8, которому непонятны такие элементы, как `<aside>`, `<header>`, `<nav>` и `<footer>`, сразу же приводит нас к основной идее этой главы — решению кросс-браузерных проблем с адаптивностью.

9.1. Прогрессивное улучшение против плавного сокращения возможностей

Вам, скорее всего, доводилось слышать словосочетания «прогрессивное улучшение» и «плавное сокращение возможностей». Под ними подразумеваются методологии, к которым прибегают, когда речь идет о сильно различающейся браузерной поддержке. При этом они вызывают ожесточенные дебаты среди членов веб-сообщества. Хотя сначала может показаться, что между ними есть что-то общее, на самом деле эти два понятия в корне противоположны. Вот мое пояснение...

Плавное сокращение возможностей подразумевает создание сайта для современных браузеров и гарантирует, что для пользователей определенных устаревших браузеров будет обеспечено приемлемое взаимодействие. Функции деградируют в устаревших браузерах, при этом обычно есть «точка прерывания», начиная с которой наиболее устаревшие браузеры больше не поддерживаются. Кроме того, в некоторых случаях пользователи просто предупреждаются о том,

что с их браузерами возникла проблема, и для них предлагаются пути ее решения (например, «ваш браузер сильно устарел — обновите его!»).

Прогрессивное улучшение — это противоположность плавного сокращения возможностей. Прогрессивное улучшение подразумевает разметку, соответствующую веб-стандартам, то есть она будет понятна всем браузерам (невзирая на такие технологии, как JavaScript и даже CSS). Качество обеспечиваемого взаимодействия затем будет прогрессивно улучшаться для более совместимых браузеров с помощью CSS-стилизации и в конечном счете JavaScript (если потребуется).

О преимуществах и недостатках каждого из этих подходов написано сотни статей. На вашем месте я для начала заглянул бы на сайт разработчика браузера Opera по адресу <http://dev.opera.com/articles/view/graceful-degradation-progressive-enhancement/>, а также прочел бы отличную статью веб-разработчика Аарона Гюстафсона (Aaron Gustafson), которая доступна по адресу <http://www.alistapart.com/articles/understandingprogressiveenhancement>.

Реальность. В настоящее время многие веб-специалисты считают прогрессивное улучшение наилучшим подходом при разработке сайтов. И хоть я по большому счету отдаю предпочтение этой методологии и использую ее при создании сайтов, жестокая правда состоит в том, что может возникнуть множество ситуаций, когда, вероятно, лучше прибегнуть к плавному сокращению возможностей. Как же так?

Кодовую базу сайта www.andthewinnerisnt.com, который мы только что создали, образует HTML5. Устаревшие браузеры, например Internet Explorer версий 6, 7 и 8 (далее по ходу книги они будут называться устаревшими версиями браузера Internet Explorer), были созданы и выпущены в свет до появления HTML5 (который, как вы помните, является неутвержденным стандартом, несмотря на то что становится все более распространенным). По этой причине им непонятно, для чего предназначены элементы `<aside>`, `<section>` и `<footer>`. Таким образом, если мыслить как пурист, можно сказать, что не следует использовать HTML5-элементы. Если для решения проблемы, связанной с базовой функциональностью, добавить фрагмент JavaScript-кода, то разве это будет прогрессивным улучшением?

Несмотря на это, я всегда предпочитаю использовать HTML5 вместо HTML 4.01, если только нет веской причины поступить по-другому. Реальность такова, что если смотреть с позиции работы, которой я занимаюсь неделя за неделей, то HTML5 имеет больше преимуществ, чем недостатков. Поэтому при использовании этой версии языка разметки (что я вам, безусловно, рекомендую делать) давайте всем устройствам возможность проявить себя в полную силу в обработке HTML5-кода, написав его в соответствии с требуемыми стандартами (используйте валидатор по адресу <http://validator.nu/> или <http://validator.w3.org/> для устранения всех ошибок).

Тем не менее вы неизбежно столкнетесь с ситуацией, когда придется сделать часть расширенной функциональности, которую предлагают современные браузеры, доступной в устаревших версиях Internet Explorer. Например, вам потребуется обеспечить поддержку свойства `border-radius`. Однако прежде, чем вы займетесь этим, я хочу еще кое о чем вам рассказать...

9.2. Следует ли вам заботиться о том, чтобы сайт работал в устаревших версиях Internet Explorer?

На данном этапе я хочу еще раз повторить то, что говорил ранее: почти наверняка можно реализовать поддержку большинства HTML5- и CSS3-функций в устаревших браузерах с помощью полизаполнений, однако пользовательское взаимодействие, обеспечиваемое таким образом, будет сильно перегружено JavaScript и может оказаться менее практичным, чем могло бы быть без полизаполнений. Излишне говорить, что важно учитывать последствия в виде снижения производительности, которые повлечет такой выбор. Возможность что-то сделать вовсе не означает, что вы должны это делать!

Кроме того, по своему опыту могу сказать, что даже без полизаполнений (на которые мы вскоре взглянем) на добавление, тестирование и конфигурирование CSS-кода, характерного для устаревших версий Internet Explorer, требуется по меньшей мере столько же времени, сколько и на улучшение внешнего вида того или иного сайта при отображении в современных браузерах, — просто этот процесс будет намного менее увлекательным! Разве так вы или ваш клиент хотите тратить время, отведенное на разработку?

Статистика

Снова обратимся к статистике, которую мы уже рассматривали в главе 1. Признавая, что статистические данные всегда следует воспринимать как ориентировочные, мы отметили, что с июля 2010 по июль 2011 года доля использования мобильных браузеров от общего количества по всему миру (согласно оценке Global Stats по адресу <http://gs.statcounter.com>) увеличилась (с 2,86 до 7,02 %), в то время как доля использования Internet Explorer 7 **уменьшилась (до 5,45 %)**. Статистика за последний месяц 2011 года оказалась еще более показательной: доля Internet Explorer 7 составила всего 4 %, а доля Internet Explorer 6 — лишь 1,78 %. Доля использования мобильных браузеров между тем выросла до 8,04 %.

Еще более интересным является тот факт, что по состоянию на декабрь 2011 года доля использования лишь одного современного браузера Google Chrome (включая обе его версии — 15 и 16) составила 25,7 % от общего количества по всему миру. Если взглянуть на показатели других современных браузеров, например Safari (4,3 %, не считая iOS-версии) и всех версий Firefox (21,01 %), а также соответствующих мобильных браузеров, то нетрудно понять, что обеспечение и улучшение качества взаимодействия для пользователей современных устройств имеет больше смысла, чем латание дыр в устаревших браузерах. По крайней мере, на мой взгляд!

Вывод: доля использования устаревших версий браузера Internet Explorer (6, 7 и 8) снижается, а доля использования современных браузеров (как настольных, так и мобильных) — растет.

Личный выбор

В настоящее время при создании новых сайтов я предпочитаю обеспечивать аккуратное отображение визуальных элементов в текущей версии браузера Internet Explorer (версии 9 на момент написания книги), а также в предыдущей версии (Internet Explorer 8). Если вы решите устранить проблемы с макетом и стилизацией в более старых версиях, то вам, возможно, придется обсудить это со своим заказчиком, поскольку на решение такой задачи потребуется дополнительное время.

Однако это не значит, что я призываю вас попросту игнорировать любые проблемы, касающиеся обеспечения удобства пользования, в таких версиях, как, например, Internet Explorer 7. Я лишь хочу сказать, что нужно ограничить время на разработку, чтобы убедиться в работоспособности базового макета и функциональности, и не обращать внимания на незначительные проблемы с выравниванием и визуальные улучшения вроде фоновых градиентов, скругленных углов, отбрасываемых блочными элементами теней и т. д., которые не поддерживаются в определенном браузере. Эти вещи не влияют на удобство пользования, а по большей части представляют собой лишь прогрессивные улучшения, которые я (равно как и другие пользователи) не ожидаю увидеть в устаревающих браузерах.



ТЕСТИРОВАНИЕ САЙТОВ В РАЗНЫХ БРАУЗЕРАХ

Браузеры, соответствующие стандартам, например Chrome, Safari и Firefox, обычно одинаково обрабатывают веб-страницы на основе HTML5 и CSS3. В настоящее время большинство браузеров, устанавливаемых на смартфонах и работающих в операционных системах Android и iOS, как и их настольные аналоги Safari и Chrome, в качестве основы используют WebKit и к тому же обрабатывают страницы так, как вы того ожидаете. Однако разные версии браузера Internet Explorer значительно отличаются друг от друга, а вы, несомненно, столкнетесь с ситуацией, когда потребуется протестировать созданный дизайн и в них тоже (если только Internet Explorer не окажется вашим браузером по умолчанию, в таком случае я вам сочувствую). Обычно я использую IE Tester (<http://www.my-debugbar.com/wiki/IETester/HomePage>) — бесплатную утилиту, позволяющую эмулировать работу разных версий браузера Internet Explorer на одном компьютере. Однако существует множество альтернативных вариантов, и на сайте Smashing Magazine имеется статья с хорошим обзором некоторых распространенных решений, доступных для выбора: <http://www.smashingmagazine.com/2011/08/07/a-dozen-cross-browser-testing-tools/>.

Вот пример, иллюстрирующий описанный подход: взглянув на сайт, который располагается по адресу www.andthewinnerisnt.com, в браузере Internet Explorer 8, мы сразу же понимаем, что нам еще предстоит провести фундаментальную работу, чтобы всего лишь сделать его функциональным. Воспользуемся отличным JavaScript-инструментом под названием Modernizr и полизаполнением, чтобы устранить пробелы в функциональности устаревших версий Internet Explorer. Я не уверен, что браузер Internet Explorer заслуживает этого после всех тех мук, которые он доставляет, но это лишь мое мнение. Однако перед тем, как сделать это, немного подробнее разберемся в том, что собой представляет Modernizr.

9.3. Modernizr — «швейцарский армейский нож» разработчика клиентских приложений

Способность участников веб-сообщества разбираться со многими разнообразными проблемами кросс-браузерной совместимости и создавать решения для простых смертных вроде меня никогда не перестанет удивлять и восхищать. О Modernizr я кратко упоминал в главах 4 и 8. Снова повторяюсь, что Modernizr — это JavaScript-библиотека с открытым исходным кодом, которая позволяет тестировать браузеры на предмет поддерживаемых ими функций. Фаук Атеш (Fauk Ateş) написал ее первую версию, а в настоящее время в проекте Modernizr также участвуют Алекс Секстон (Alex Sexton) и невероятно талантливый Пол Айриш (Paul Irish), выступающий в качестве ведущего разработчика. Modernizr является любимым инструментом многих известных компаний — Twitter, Microsoft и Google. Я говорю это не просто для того, чтобы превознести команду разработчиков Modernizr (хотя они, вне всякого сомнения, заслуживают этого), а больше для того, чтобы показать, что этот инструмент является *современным*, замечательным JavaScript-решением. Одним словом, это инструмент, которым вам стоит хорошо владеть.

Так что же фактически делает Modernizr? Каким образом этот инструмент позволяет нам устранять пробелы в функциональности устаревших браузеров и прогрессивно улучшать обеспечиваемое пользовательское взаимодействие в современных браузерах и как мы сможем заставить его делать то, что нам нужно? Читайте дальше и все узнаете...

Работа Modernizr по умолчанию немного отличается от процедуры добавления HTML5-прослойки Реми Шарпа (если выбор был сделан в пользу этого решения) с целью включить поддержку структурных HTML5-элементов, например `<aside>` и `<section>`, в не поддерживающие HTML5 браузеры вроде Internet Explorer версии 8 и ниже. Инструмент Modernizr тестирует браузеры на предмет распознаваемых ими функций. В результате он сможет узнать, поддерживает ли соответствующий браузер различные HTML5- и CSS3-свойства. Затем он обеспечит средства, позволяющие предпринимать разные действия в зависимости от сведений, полученных в результате тестирования. Реализация всего остального возлагается на нас. Итак, добавим Modernizr в наши страницы и приступим к работе.

Сначала скачайте Modernizr (по адресу <http://www.modernizr.com>).



КАКУЮ ВЕРСИЮ MODERNIZR ИСПОЛЬЗОВАТЬ — ВЕРСИЮ ДЛЯ РАЗРАБОТКИ ИЛИ ПРОИЗВОДСТВЕННУЮ?

Если вам интересно, как действует этот инструмент, то скачивайте версию Modernizr для разработки (рис. 9.6), поскольку в ней документирована каждая функция/тест. А если вы предпочтете производственную версию, то сможете выбирать только тесты, актуальные для определенного сайта или веб-приложения, которое вы создаете, сохраняя при этом свои файлы довольно компактными.

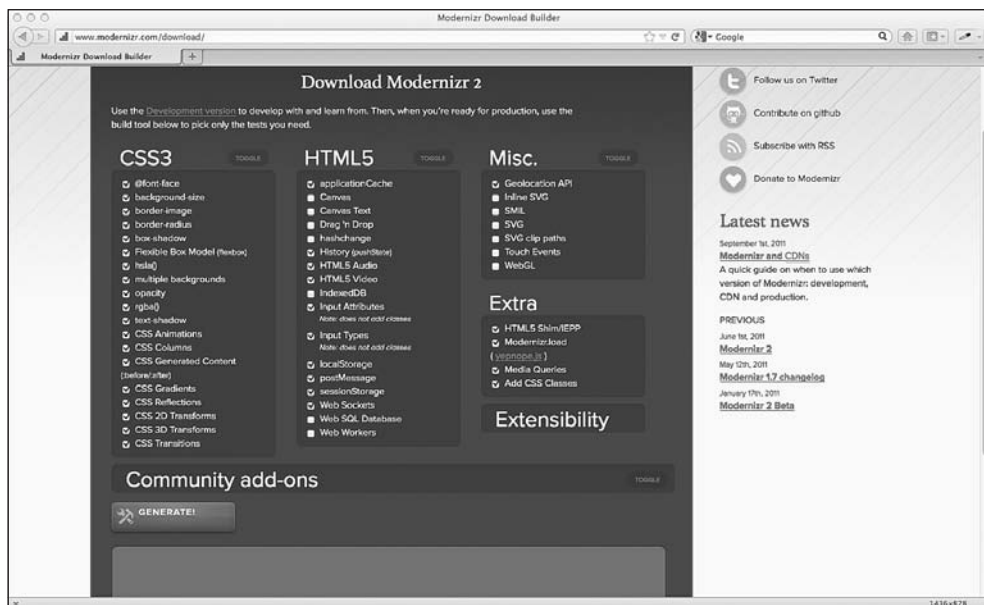


Рис. 9.6. Страница загрузки Modernizr

Далее поместите скачанный файл в подходящую папку (как и раньше, я использовал для этой цели корневой каталог `js`). Затем укажите ссылку на файл в теге `<head>` своей страницы:

```
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initialscale=1.0,maximum-scale=1" />
<title>And the winner isn't...</title>
<link href="css/main.css" rel="stylesheet" />
<script src="js/modernizr.js"></script>
</head>
```

Если вы, добавив Modernizr, взглянете на исходный код страницы в Firebug или чем-то похожем, то увидите множество разных классов, добавленных в тег `<html>`. Вот пример из Firefox 9.01:

```
<html class=" js flexbox geolocation postmessage indexeddb history websockets rgba
hsla multiplebgs backgroundsize borderimage borderradius boxshadow textshadow opacity
cssanimations csscolumns cssgradients no-cssreflections csstransforms
no-csstransforms3d csstransitions fontface generatedcontent video audio localStorage
sessionstorage applicationcache" lang="en">
```

Этот фрагмент очень информативен. В нем последовательно приводятся функции, на предмет которых был проведен тест и которые браузер поддерживает или не поддерживает (если функция не поддерживается, то название снабжается префиксом `no-`). Все это позволит нам сделать две важные вещи — функ-

ция за функцией устранить проблемы со стилизацией в наших CSS-файлах, а также условно загружать дополнительные CSS- и JavaScript-файлы только при необходимости.

Устранение проблем со стилизацией с помощью Modernizr

Наш адаптивный сайт And the winner isn't... предоставляет нам отличную возможность воспользоваться Modernizr для решения возникшей проблемы. Страница с викториной (по адресу <http://www.andthewinnerisnt.com/3Dquiz.html>) прекрасно работает в браузерах, поддерживающих 3D-трансформации (таких как Safari и Chrome), однако в остальных браузерах на этой странице при наведении указателя мыши не наблюдается 3D-эффект. В настоящее время, независимо от того, способен ли браузер пользователя визуализировать 3D-трансформации, на нашей странице выводится следующее примечание: *This page relies on 3D transforms. If the posters don't flip on hover, try viewing in Safari or Chrome* (Эффект на данной странице базируется на 3D-трансформациях. Если постеры фильмов не переворачиваются при наведении на них указателя мыши, используйте для просмотра браузер Safari или Chrome).

Однако благодаря дополнительным классам Modernizr мы теперь можем сделать так, что соответствующее примечание будет отображаться только в том случае, если браузер пользователя *не* поддерживает 3D-трансформации.

```
.note {  
  display: none;  
}  
.no-csstransforms3d .note {  
  display: block;  
}
```

Проанализируем этот фрагмент: сначала мы указываем в CSS-коде, что примечание не должно отображаться по умолчанию:

```
.note {  
  display: none;  
}
```

Это означает, что в браузерах, поддерживающих 3D-трансформации CSS (например, в Google Chrome 16), пользователи не увидят примечание (рис. 9.7).

Затем, во втором правиле, мы задействовали классы, добавленные с помощью Modernizr, для отображения примечания в браузерах, которые не поддерживают 3D-трансформации:

```
.no-csstransforms3d .note {  
  display: block;  
}
```

На рис. 9.8 показана все та же страница, но в браузере Firefox 9.



Рис. 9.7. Примечание не выводится, так как браузер поддерживает 3D-трансформации



Рис. 9.8. Здесь примечание выводится, так как браузер Firefox не позволяет увидеть 3D-эффект

Modernizr дает нам возможность перестать думать о том, какой именно браузер используется, а вместо этого задумываться о том, какие функции он поддерживает.

Modernizr добавляет поддержку HTML5-элементов в устаревшие версии браузера Internet Explorer

Поскольку я выбрал специальную производственную версию Modernizr, включающую HTML5-прослойку, то теперь после обновления нашей веб-страницы в Internet Explorer 8 она выглядит намного лучше, чем раньше (рис. 9.9).

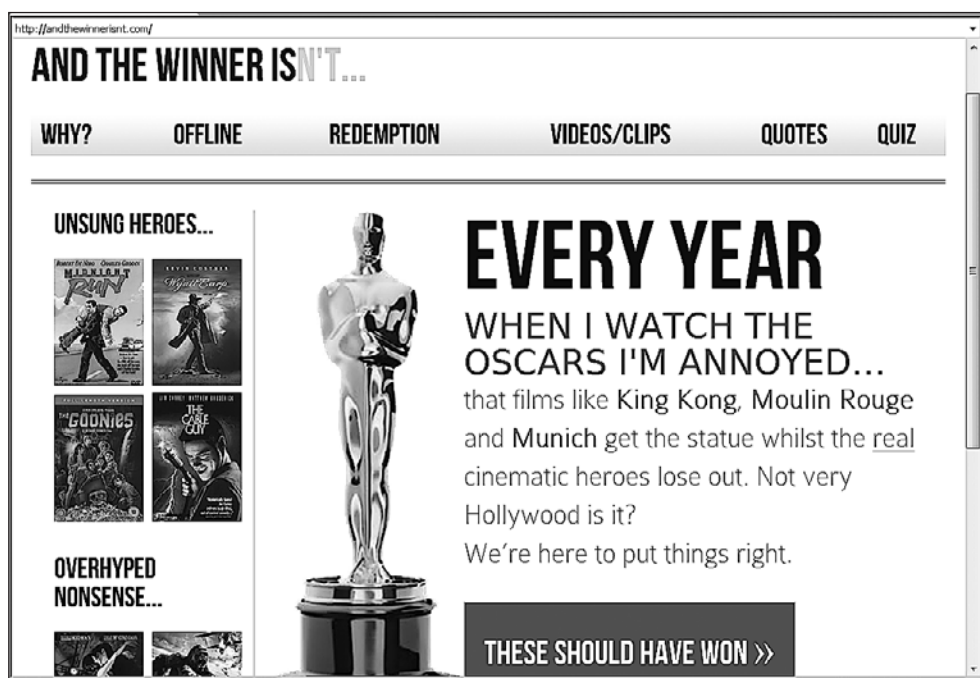


Рис. 9.9. Улучшенный вид нашей страницы в Internet Explorer 8

Мне не потребовалось делать что-либо еще. Поскольку JavaScript-библиотека Modernizr позволила добавить поддержку структурных HTML5-элементов в устаревшие версии браузера Internet Explorer, многие стандартные CSS-стили теперь стали понятны этим версиям, и страница обрабатывается как следует.

На мой взгляд, все выглядит вполне приемлемо. Если вы не знаете, как этот же сайт отображается в современном браузере, то вряд ли заметите какую-либо разницу. Однако, поскольку Internet Explorer 8 не поддерживает CSS3, мы знаем, что при отображении в нем дизайн будет иметь кое-какие очевидные недостатки в визуальном плане, если сравнивать с просмотром в современном браузере. В частности, навигационные ссылки не окрашены в чередующиеся цвета (при необходимости

это можно легко исправить, добавив дополнительный класс для нечетных навигационных ссылок), у кнопки нет скругленных углов, текст и блочные элементы не отбрасывают тени. И, что, возможно, более важно, несмотря на гибкость нашей «резиновой» сетки, отсутствие поддержки CSS3 означает, что медиазапросы тоже не поддерживаются. Нет поддержки медиазапросов — нет значительных изменений макета при отображении в разных по размеру областях просмотра в браузере Internet Explorer версий 6, 7 и 8.

Хотя я никоим образом не считаю наш макет неполноценным, такой инструмент, как Modernizr, также дает нам возможность использовать полизаполнения для устранения пробелов в функциональности устаревших браузеров. В качестве наглядного примера добавим поддержку медиазапросов `min-width` и `max-width`, чтобы наш дизайн адаптировался к разным по размеру областям просмотра в Internet Explorer версий 6, 7 и 8.

Добавление поддержки медиазапросов `min-width` и `max-width` в Internet Explorer версий 6, 7 и 8

Полизаполнение, которое я обычно использую для добавления поддержки медиазапросов в устаревшие версии браузера Internet Explorer, рассчитано только на `min-width` и `max-width`. Существуют более солидные полизаполнения, добавляющие поддержку широкого диапазона медиазапросов, однако для адаптивных веб-дизайнов удобно использовать написанный Скоттом Джелом (Scott Jehl) сценарий `Respond.js`, являющийся простым в применении и быстрым решением, которое всегда хорошо служило мне.

Сценарий `Respond.js` (<https://github.com/scottjehl/Respond>) на самом деле можно использовать без Modernizr — нужно просто подключить его к соответствующей странице и, как выражается сам автор, «запустить браузер Internet Explorer и сделать радостное движение сжатыми в кулаки руками».

Итак, перед тем, как интегрировать `Respond.js` с Modernizr, подключим его. Добавим `Respond.js` прямо в нашу страницу (сразу после файла Modernizr, который мы уже внедрили) и проверим, позволил ли он добиться нужного нам результата в Internet Explorer. Для этого нужно скачать соответствующий файл, поместить его в подходящую папку и указать ссылку на него в разделе `<head>`:

```
<head>
<meta charset=utf-8>
<meta name="viewport" content="width=device-width,initialscale=1.0,maximum-scale=1" />
<title>And the winner isn't...</title>
<link href="css/main.css" rel="stylesheet" />
<script src="js/modernizr.js"></script>
<script src="js/respond.min.js"></script>
</head>
```

Теперь если мы загрузим страницу в Internet Explorer 8 и изменим размеры окна браузера, то вновь получим наш действующий адаптивный веб-дизайн (рис. 9.10).



Рис. 9.10. Страница по-прежнему адаптируется к области просмотра

Отлично, мы добавили полизаполнение, которое обеспечивает поддержку медиазапросов `min-width` и `max-width` в Internet Explorer, однако есть одна загвоздка: оно теперь начнет задействоваться в каждом браузере, который будет загружать нашу страницу независимо от того, нуждается он в нем или нет. Для решения возникшей проблемы можно добавить условный комментарий, касающийся Internet Explorer, как показано далее:

```
<!--[if lte IE 8]>
    <script src="js/respond.min.js"/></script>
<![endif]-->
```

Я уверен, что ранее вы уже сталкивались с условными комментариями. Они представляют собой простой способ обеспечить загрузку CSS- и JavaScript-файлов (или даже содержимого), которые будут использоваться только соответствующей версией Internet Explorer. **Все прочие браузеры будут считать данный код комментарием и игнорировать его.**

В нашем условном комментарии говорится следующее: «Если версия данного браузера *ниже или равна* (часть `lte` — *less than or equal*) Internet Explorer 8 (часть `IE 8`), то следует задействовать указанный сценарий».



ВСЕ ОБ УСЛОВНЫХ КОММЕНТАРИЯХ

Если вы захотите узнать больше об условных комментариях, то по следующему URL-адресу найдите полную информацию: <http://msdn.microsoft.com/en-us/library/ms537512%28v=vs.85%29.aspx>.

Все это отлично срабатывает. Однако разве мы хотим засорять свою разметку условными комментариями, созданными специально для Internet Explorer? И как насчет полизаполнений для других браузеров? Именно здесь Modernizr берет ответственность на себя.

Условная загрузка с помощью Modernizr

Если вы стремитесь сделать свои сайты и веб-приложения как можно более компактными, то Modernizr придется весьма кстати, поскольку он позволяет условно загружать ресурсы (как CSS-, так и JavaScript-файлы). Таким образом, вместо того, чтобы обременять страницы всеми полизаполнениями, которые *могут* потребоваться пользователю (независимо от того, действительно ли он в них нуждается), мы сможем загружать только полизаполнения, которые *действительно* необходимы ему. Это позволит обеспечить как можно большую компактность наших страниц и наименьшее время их загрузки для всех без исключения пользователей.

Итак, добавив библиотеку Modernizr в теги <head> страниц, используем ее для обеспечения условной загрузки нашего полизаполнения Respond.js только в том случае, если конкретный браузер не будет поддерживать CSS3-медиазапросы (например, Internet Explorer версий 6, 7 и 8).

Modernizr включает JavaScript-микробιβотеку под названием YepNope.js (<http://yepnopejs.com/>). Этот инструмент использует простой формат:

```
Modernizr.load({
  test: Modernizr.mq('only all'),
  nope: 'js/respond.min.js'
});
```

Сначала вызывается часть Modernizr, которая загружает ресурсы:

```
Modernizr.load({
```

Сюда входит тест, а также несколько возможных действий в зависимости от его результата. В данном примере мы спрашиваем, поддерживает ли браузер медиазапросы:

```
test: Modernizr.mq('only all'),
```

Если окажется, что он не поддерживает их, то соответствующий ресурс должен будет загрузить наш файл respond.min.js:

```
nope: 'js/respond.min.js'
```

Здесь `only all` — это эквивалент вопроса «поддерживает ли данный браузер медиазапросы?». Устаревшие версии Internet Explorer всегда будут проваливать этот тест, что повлечет за собой `nope` и, следовательно, загрузку соответствующего ресурса. Благодаря этому `respond.min.js` будет загружаться только при необходимости.

Мы также можем выбрать загрузку дополнительных файлов:

```
Modernizr.load({  
  test: Modernizr.mq('only all'),  
  nope: ['js/respond.min.js', 'css/extra.css']  
});
```

В данном примере для добавления файлов `respond.min.js` и `extra.css` используется массив. Мы можем выбрать загрузку CSS-кода таким путем, чтобы обеспечить поддержку отдельных стилей, необходимых только при наличии или отсутствии определенных функций. Не стоит забывать, что также можно загружать разные ресурсы в зависимости от результатов тестирования:

```
Modernizr.load({  
  test: Modernizr.mq('only all'),  
  yep: 'js/pass.js',  
  nope: 'js/respond.min.js' ['fail-polyfill.js', 'fail.css'],  
  both: 'js/for-all.js'  
});
```

Здесь мы загружаем один файл, если браузер проходит тест, два других (в массиве), если он его проваливает, и последний файл, если браузер проходит *или* проваливает тест.

Тесты для условной загрузки кода можно разместить в отдельном JavaScript-файле. В данном случае я присвоил своему файлу имя `conditional.js` и сохранил его в папке `js` наряду с `modernizr.js` и `respond.min.js`. Таким образом, тег `<head>` теперь выглядит так, как показано далее:

```
<head>  
<meta charset=utf-8>  
<meta name="viewport" content="width=device-width,initialscale=1.0,maximum-scale=1"  
>  
<title>And the winner isn't...</title>  
<link href="css/main.css" rel="stylesheet" />  
<script src="js/modernizr.js"></script>  
<script src="js/conditional.js"></script>  
</head>
```

Обратите внимание, что я удалил `respond.min.js` из `<head>`, поскольку теперь он загружается условно по мере необходимости.



ПРИМЕЧАНИЕ

Дополнительная документация со сведениями об условной загрузке ресурсов с помощью Modernizr доступна по адресу <http://www.modernizr.com/docs/#load>.



ГДЕ МОЖНО НАЙТИ ПОЛИЗАПОЛНЕНИЯ

Помните, что на сайте Github имеется отличный репозиторий полезных полизаполнений: <https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills>.

9.4. Преобразование списка навигационных ссылок в раскрывающееся меню (условно)

Распространенная проблема почти всех адаптивных веб-дизайнов состоит в том, что, если на странице имеется много навигационных ссылок, они могут занимать изрядную долю экранного пространства в небольших по размеру областях просмотра.

Например, вот как на текущий момент выглядит любая страница сайта *And the winner isn't...*, имея лишь шесть ссылок на другие страницы, при загрузке в небольшой по размеру области просмотра (рис. 9.11).



Рис. 9.11. Навигационная панель сайта частично скрывается на небольшом экране

Я хотел бы сделать так, чтобы список этих ссылок заменялся раскрывающимся меню, но только если ширина области просмотра в браузере окажется меньше определенной величины. Вы можете добавить собственный JavaScript-код для преобразования ссылок в элементы раскрывающегося меню. Почтенный Крис Койер написал статью о том, как этого можно добиться (<http://css-tricks.com/convert-menu-to-dropdown/>). В качестве альтернативы вы можете прибегнуть к уже готовым сценариям, которые сделают эту работу за вас. Ради краткости и простоты я решил воспользоваться одним из таких сценариев. На рис. 9.12 показано, как все будет выглядеть после преобразования списка наших навигационных ссылок в раскрывающееся меню в меньших по размеру областях просмотра.



Рис. 9.12. Вместо перечня ссылок появилась кнопка

После нажатия кнопки **Select a page** (Выбрать страницу) на экране отобразятся навигационные ссылки, как показано на рис. 9.13.



Рис. 9.13. Нажав кнопку, можно выбрать нужную страницу

Это отличный пример прогрессивного улучшения. Конечно, это не самая важная функциональность, однако она обеспечивает больше видимого без прокрутки содержимого для пользователей в меньших по размеру областях просмотра. Итак, реализуем ее. Прежде всего скачайте сценарий Responsive Menu (<https://github.com/mattkersley/Responsive-Menu>). Как и ранее, сохраните соответствующий файл (`jquery.mobilemenu.js`) в папке `js`. Сначала нужно в разметке снабдить идентификаторами наши навигационные ссылки на каждой странице:

```
<nav role="navigation">
  <ul id="mainNav">
    <li><a href="why.html">Why?</a></li>
    <li><a href="offline.html">Offline</a></li>
    <li><a href="redemption.html">Redemption</a></li>
    <li><a href="video.html">Videos/clips</a></li>
```



```
<li><a href="quotes.html">Quotes</a></li>
<li><a href="3Dquiz.html">Quiz</a></li>
</ul>
</nav>
```

Мы смогли бы не делать этого, однако jQuery-селекторы работают намного быстрее, если указать идентификатор, на который они смогут ориентироваться.

Теперь добавим в файл `conditional.js` следующий код:

```
Modernizr.load([
{
    test: Modernizr.mq('only all'),
    nope: 'js/respond.min.js'
},
{
    // загрузить преобразователь меню, если значением max-width будет 600px;
    test: Modernizr.mq('only screen and (max-width: 600px)'),
    yep : ['js/jquery-1.7.1.js', 'js/jquery.mobilemenu.js'],
    complete : function () {
        // Выполнить это после того, как все в данной группе будет загружено
        // и выполнено, равно как и во всех предыдущих группах
        $(document).ready(function(){
            $('#mainNav').mobileMenu({
                switchWidth: 600,                                // ширина в пикселах, при которой
                                                                // должно осуществляться
                                                                // преобразование
                topOptionText: 'Select a page',                  // текст первого параметра
                indentString: '    '                        // строка для задания отступов между
                                                                // вложенными элементами
            });
        });
    }
}
]);
```

После предыдущей условной загрузки, при которой добавляется `Respond.js` для устаревших версий Internet Explorer, мы указали новый тест:

```
test: Modernizr.mq('only screen and (max-width: 600px)'),
```

В этом тесте мы поставили вопрос о том, поддерживает ли браузер медиазапросы, и если да, то составляет ли максимальная ширина области просмотра в нем 600 пикселей. Если он их поддерживает...

```
yep : ['js/jquery-1.7.1.js', 'js/jquery.mobilemenu.js'],
```

Приведенная выше строка обеспечивает загрузку как библиотеки jQuery, так и файла Responsive Menu:

```
complete : function () {
...еще код...
```



```
<div class="logo">
  <a href="http://benfrain.com/"></a>
</div>
```

А вот CSS-правило, загружающее логотип:

```
#container header[role="banner"] .logo a {
  background-image: url("../img/logo2.png");
  background-repeat: no-repeat;
  background-size: contain;
  display: block;
  height: 7em;
  margin-top: 10px;
}
```

Первоначально логотип выглядел так, как показано на рис. 9.14.



Рис. 9.14. Логотип моего сайта. Изначальный вариант

Логотип имел вполне нормальный вид, однако я захотел, чтобы он выглядел как можно более четким на экранах высокого разрешения. Поэтому я создал две дополнительные версии логотипа (одну для состояния по умолчанию, а другую — для выделенного состояния), вдвое бóльшие по размеру, чем уже имевшийся логотип,

и присвоил им имена соответственно `logo2@x2.png` и `logo20ver@x2.png`. Затем я добавил в свой CSS-код следующий медиазапрос:

```
@media all and (-webkit-min-device-pixel-ratio : 1.5) {  
  #container header[role="banner"] .logo a {  
    background-image: url("../img/logo2@x2.png");  
  }  
  #container header[role="banner"] .logo a:hover {  
    background-image: url("../img/logo20ver@x2.png");  
  }  
}
```

Он позволяет нацелиться на устройства с экранами, для которых `min-device-pixel-ratio` имеет значение 1.5. Следовательно, дисплеи высокого разрешения, как у iPhone версии 4 и выше, попадают в эту категорию и могут обеспечить отображение заданных стилей надлежащим образом. Как видите, приведенное правило включает префикс `-webkit-`. Как и всегда, не забывайте о соответствующих префиксах для устройств, на которые вам понадобится нацелиться.

Теперь при просмотре моего сайта на экранах высокого разрешения будет загружаться более высококачественная версия его логотипа, как показано на рис. 9.15.

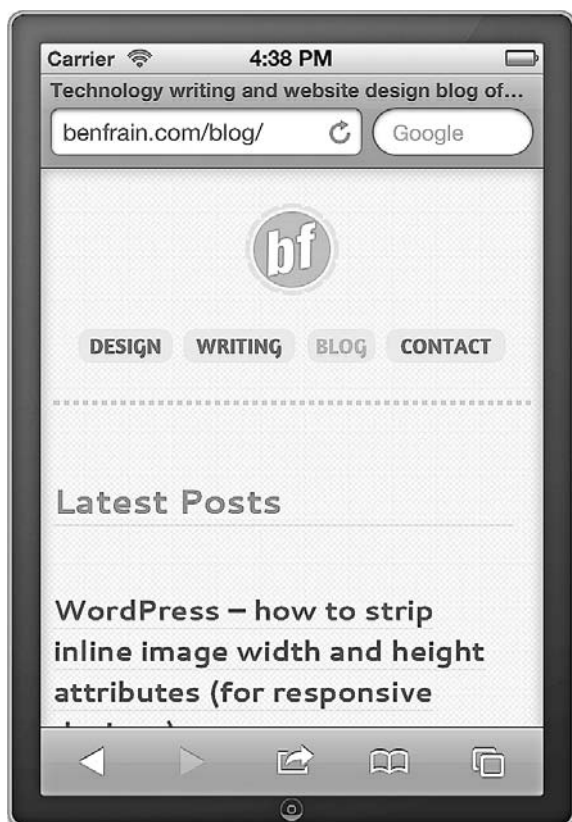


Рис. 9.15. Логотип моего сайта. Вариант для устройств с экранами высокого разрешения

Следует признать, что разницу между отличающимися по качеству версиями логотипа моего сайта трудно заметить на приведенных рисунках. Вероятно, вам лучше самим взглянуть на них вживую. В целом, чем более детализирован рисунок, тем выше вероятность того, что он будет с превосходной четкостью отображаться на экранах высокого разрешения.

Прибегая к такой методике, следует принимать во внимание некоторые особенности. Более крупные по размеру изображения означают более крупные размеры их файлов, а также более длительное время загрузки, в связи с чем повторяюсь, что просто потому, что вы можете что-то сделать, вовсе не значит, что вы должны это сделать.

Если устройство поддерживает формат **Scalable Vector Graphics (SVG)** (Масштабируемая векторная графика), можно устранить многие проблемы с масштабированием изображений, с которыми мы сталкиваемся в настоящее время. Как следует из названия формата, он предназначен для создания рисунков, которые способны четко отображаться независимо от требуемого масштаба. Однако медиазапросы и SVG не помогут, если речь пойдет о встроенных фотографиях для экранов высокого разрешения. В таком случае вам придется прибегнуть к решениям, основанным на JavaScript.

9.6. Резюме

В этой главе мы выяснили, в чем заключается принципиальная разница между прогрессивным улучшением и плавным сокращением возможностей. Затем мы использовали полизаполнение для добавления поддержки медиазапросов в устаревшие версии браузера Internet Explorer, чтобы наш веб-дизайн и в них был адаптивным. И наконец, мы применили Modernizr для условной загрузки CSS- и JavaScript-файлов в зависимости от результатов нескольких тестов браузеров на предмет поддерживаемых ими функций, что позволяет нам задействовать полизаполнения и дополнительные либо альтернативные стили только в том случае, если окажется, что браузеры не поддерживают требуемые функции. В завершение мы кратко рассмотрели технологии, которые получают широкое распространение в ближайшем будущем, а также способы использования **CSS3 для обеспечения дополнительных возможностей** при просмотре сайтов на экранах устройств, которые их поддерживают.

На данном этапе скромный автор этой книги полагает (и надеется), что поведал обо всех методиках и инструментах, которые вам потребуются для того, чтобы приступить к созданию вашего следующего сайта или веб-приложения с применением принципов адаптивного веб-дизайна.

Я твердо убежден, что адаптивные веб-дизайны, создаваемые с использованием HTML5 и CSS3, на данный момент станут наилучшим выбором для разработчиков клиентских приложений при работе с подавляющим большинством сайтов. Предполагая внесение небольших изменений в рабочий процесс, методики и техники, адаптивные веб-дизайны дают нам возможность создавать быстро загружающиеся, гибкие и удобные в сопровождении сайты, которые способны потрясюще выглядеть независимо от размера областей, используемых для их просмотра.

Поскольку количество пользователей мобильных устройств продолжает расти по экспоненте, а новые устройства, о которых мы никогда не задумывались ранее, включаются в конкуренцию в сфере средств просмотра информации, рассматриваемая методология обеспечивает, вероятно, самые надежные и наиболее ориентированные на будущее инструменты для создания веб-дизайнов. Такие веб-дизайны будут хорошо смотреться на экранах любых устройств, в любых по размеру областях просмотра и визуализироваться настолько быстро, насколько это представляется возможным, на применяемых пользователями устройствах, подключаемых к Интернету.

Бен Фрейн

HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств

Перевел с английского В. Черник

Заведующий редакцией

Ведущий редактор

Художник

Корректоры

Верстка

Д. Виноцкий

Н. Гринчик

Л. Адуевская

О. Андросик, Е. Павлович

Г. Блинов

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Подписано в печать 09.08.13. Формат 70×100/16. Усл. п. л. 24,510. Тираж 2000. Заказ 0000.

Отпечатано в полном соответствии с качеством предоставленных издательством материалов
в ГППО «Псковская областная типография». 180004, Псков, ул. Ротная, 34.



Нет времени ходить по магазинам?



наберите:



www.piter.com



Здесь вы найдете:

Все книги издательства сразу
Новые книги — в момент выхода из типографии
Информацию о книге — отзывы, рецензии, отрывки
Старые книги — в библиотеке и на CD



**И наконец, вы нигде не купите
наши книги дешевле!**

ВАМ НРАВЯТСЯ НАШИ КНИГИ? ЗАРАБАТЫВАЙТЕ ВМЕСТЕ С НАМИ!

У Вас есть свой сайт?

Вы ведете блог?

Регулярно общаетесь на форумах? Интересуетесь литературой, любите рекомендовать хорошие книги и хотели бы стать нашим партнером?

ЭТО ВПОЛНЕ РЕАЛЬНО!

СТАНЬТЕ УЧАСТНИКОМ ПАРТНЕРСКОЙ ПРОГРАММЫ ИЗДАТЕЛЬСТВА «ПИТЕР»!



Зарегистрируйтесь на нашем сайте в качестве партнера по адресу www.piter.com/ePartners



Получите свой персональный уникальный номер партнера



Выбирайте книги на сайте www.piter.com, размещайте информацию о них на своем сайте, в блоге или на форуме и добавляйте в текст ссылки на эти книги (на сайт www.piter.com)

ВНИМАНИЕ! В каждую ссылку необходимо добавить свой персональный уникальный номер партнера.

С этого момента получайте 10% от стоимости каждой покупки, которую совершит клиент, придя в интернет-магазин «Питер» по ссылке с Вашим партнерским номером. А если покупатель приобрел не только эту книгу, но и другие издания, Вы получаете дополнительно по 5% от стоимости каждой книги.

Деньги с виртуального счета Вы можете потратить на покупку книг в интернет-магазине издательства «Питер», а также, если сумма будет больше 500 рублей, перевести их на кошелек в системе Яндекс.Деньги или Web.Money.

Пример партнерской ссылки:

<http://www.piter.com/book.phtml?978538800282> – обычная ссылка

<http://www.piter.com/book.phtml?978538800282&refer=0000> – партнерская ссылка, где 0000 – это ваш уникальный партнерский номер

**Подробнее о Партнерской программе
ИД «Питер» читайте на сайте
WWW.PITER.COM**

ИЗДАТЕЛЬСКИЙ ДОМ
 **ПИТЕР®**
WWW.PITER.COM





КНИГА-ПОЧТОЙ



ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **postbook@piter.com**
- по телефону: **(812) 703-73-74**
- по почте: **197198, Санкт-Петербург, а/я 127, ООО «Питер Мейл»**
- по ICQ: **413763617**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

-  Наложением платежом с оплатой при получении в ближайшем почтовом отделении.
-  С помощью банковской карты. Во время заказа Вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
-  Электронными деньгами. Мы принимаем к оплате все виды электронных денег: от традиционных Яндекс.Деньги и Web-money до USD E-Gold, MoneyMail, INOCard, RBK Money (RuPay), USD Bets, Mobile Wallet и др.
-  В любом банке, распечатав квитанцию, которая формируется автоматически после совершения Вами заказа.

Все посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку Ваших покупок максимально быстро. Дату отправления Вашей покупки и предполагаемую дату доставки Вам сообщат по e-mail.

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, факс, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

ИЗДАТЕЛЬСКИЙ ДОМ
ПИТЕР®
WWW.PITER.COM



ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
предлагают профессиональную и популярную литературу по различным
направлениям: история и публицистика, экономика и финансы, менеджмент
и маркетинг, компьютерные технологии, медицина и психология.

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: voronej@piter.com

Екатеринбург: ул. Бебеля, д. 11а
тел./факс: (343) 378-98-41, 378-98-42; e-mail: office@ekat.piter.com

Нижний Новгород: тел.: 8 960 187-85-50; e-mail: nnovgorod@piter.com

Новосибирск: Комбинатский пер., д. 3
тел./факс: (383) 279-73-92; e-mail: sib@nsk.piter.com

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 229-68-09; e-mail: samara@piter.com


УКРАИНА


Киев: Московский пр., д. 6, корп. 1, офис 33
тел./факс: (044) 490-35-69, 490-35-68; e-mail: office@kiev.piter.com


Харьков: ул. Суздальские ряды, д. 12, офис 10
тел./факс: (057) 7584145, +38 067 545-55-64; e-mail: piter@kharkov.piter.com

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163
тел./факс: (517) 208-80-01, 208-81-25; e-mail: minsk@piter.com

 Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок
тел./факс: (812) 703-73-73; e-mail: spb@piter.com

 Издательский дом «Питер» приглашает к сотрудничеству авторов
тел./факс издательства: (812) 703-73-72, (495) 974-34-50

 Заказ книг для вузов и библиотек
тел./факс: (812) 703-73-73, доб. 6250; e-mail: ucchebник@piter.com

 Заказ книг по почте: на сайте www.piter.com; по тел.: (812) 703-73-74, доб. 6225



560 с., 16,5×23,3

БЕСТСЕЛЛЕРЫ O'REILLY

Р. Никсон

Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS. 2-е издание

Научитесь создавать современные динамические веб-сайты, даже если у вас нет опыта в программировании. Если вы умеете писать статические сайты на HTML, то с помощью этого руководства вы освоите динамическое веб-программирование и изучите современные технологии с открытым кодом: PHP, MySQL, JavaScript и CSS. В данном руководстве каждая технология рассматривается отдельно и показывается, как их объединить в одно целое, дается представление о самых современных концепциях веб-программирования. С помощью подробно разобранных примеров и контрольных вопросов, приводимых в каждой главе, вы сможете закрепить изученный материал на практике.



512 с., 16,5×23,3

БЕСТСЕЛЛЕРЫ O'REILLY

Б. Маклафлин

PHP и MySQL. Исчерпывающее руководство

Если у вас есть опыт разработки сайтов с помощью CSS и JavaScript, то эта книга переведет вас на новый уровень веб-разработки — создание динамических веб-сайтов на основе PHP и MySQL. С помощью практических примеров в книге вы узнаете все возможности серверного программирования. Вы прочитаете, как выстраивать базу данных, управлять контентом и обмениваться информацией с пользователями, применяя запросы и веб-формы. • Написание PHP-скриптов и создание веб-форм • Синтаксис PHP и SQL • Создание и управление базой данных • Создание динамических веб-страниц, которые изменяются при каждом новом просмотре • Разработка шаблонов страниц об ошибках, которые будут выводиться пользователям • Использование файловой системы для доступа к данным пользователя, включая иллюстрации и двоичные файлы • Создание административной страницы для управления сайтом.