

Дэн Седерхольм

ПУЛЕНЕПРОБИВАЕМЫЙ

ВЕБ ДИЗАЙН



Dan Cederholm

(3rd Edition)

Bulletproof Web Design:

Improving flexibility and protecting
against worst-case scenarios
with HTML5 and CSS3



New
Riders

VOICES THAT MATTER™

Дэн Седерхольм

3-е издание

ПУЛЕНЕПРОБИВАЕМЫЙ

ВЕБ ДИЗАЙН



 **ПИТЕР®**

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

2012

ББК 32.988.02-018
УДК 004.7
С28

- Седерхольм Д.**
С28 Пуленепробиваемый веб-дизайн. Библиотека специалиста. 3-е изд. — СПб.: Питер, 2012. — 304 с.: ил.

ISBN 978-5-459-01271-2

Эта книга, выходящая уже в третьем издании, посвящена концепции «пуленепробиваемого» веб-дизайна. Она научит вас применять HTML и CSS для разработки современных веб-сайтов, доступных во всех браузерах и устройствах и отличающихся гибкостью и устойчивостью к любым ситуациям. В каждой главе книги рассматривается определенный принцип «пуленепробиваемого» дизайна и описывается, какие именно преимущества дает его использование. В последней главе все ранее изученные методики сводятся воедино для разработки готового макета «пуленепробиваемой» веб-страницы. Все примеры рассматриваются на базе современных веб-стандартов HTML5 и CSS3.

Книга предназначена для веб-дизайнеров, стремящихся освоить современные технологии веб-разработки на базе актуальных стандартов и методик. На практических примерах Дэн Седерхольм, автор нескольких бестселлеров по веб-программированию и дизайну, предлагает новый перспективный подход к созданию гибких и адаптируемых интернет-проектов.

ББК 32.988.02-018
УДК 004.7

Права на издание получены по соглашению с New Riders Publishing. 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 978-0321808356 англ.
ISBN 978-5-459-01271-2

© Daniel Cederholm, 2012
© Перевод на русский язык ООО Издательство «Питер», 2012
© Издание на русском языке, оформление ООО Издательство «Питер», 2012

Оглавление

Благодарности	8
Введение	9
Концепция пуленепробиваемого дизайна	10
Актуальность вопроса	10
Структура книги	11
От издательства	15
Глава 1. Гибкое управление текстом	17
Общепринятый подход	19
Уязвимые места	20
Альтернативные способы	22
Пуленепробиваемый подход	24
Преимущества пуленепробиваемого подхода	26
Что дальше?	26
Ключевые слова и проценты	31
Задание размеров с помощью ем	35
Резюме	39
Глава 2. Навигация	41
Общепринятый подход	43
Уязвимые места	45
Пуленепробиваемый подход	46
Преимущества пуленепробиваемого подхода	56
Как создать градиент в CSS3 без использования изображений	57
Использование ем	60
Дополнительные примеры	62
Резюме	65
Глава 3. Эластичные строки	67
Общепринятый подход	69
Уязвимые места	70
Пуленепробиваемый подход	71
Преимущества пуленепробиваемого подхода	85
Возможности border-radius	86
Другой пример элементов с изменяемыми границами	88
Резюме	93
Глава 4. Креативное перетекание	95
Общепринятый подход	97
Уязвимые места	98

Пуленепробиваемый подход	99
Преимущества пуленепробиваемого подхода	132
Резюме	132
Глава 5. Несокрушимые элементы	135
Общепринятый подход	137
Уязвимые места	139
Пуленепробиваемый подход	140
Преимущества пуленепробиваемого подхода	146
Используем CSS3	147
Другие способы скругления углов	152
Хитрости с боксами	160
Резюме	166
Глава 6. Нет картинок? Нет CSS? Нет проблем!	167
Общепринятый подход	169
Уязвимые места	171
Пуленепробиваемый подход	173
Преимущества пуленепробиваемого подхода	174
Со стилем или без стиля	177
Общепринятый подход	177
Пуленепробиваемый подход	179
DIG DUG-ТЕСТ	181
Инструменты пуленепробиваемости	182
Резюме	191
Глава 7. Преобразование таблиц	193
Общепринятый подход	195
Уязвимые места	196
Пуленепробиваемый подход	197
Преимущества пуленепробиваемого подхода	216
Резюме	217
Глава 8. Резиновые и эластичные макеты	219
Общепринятый подход	221
Уязвимые места	222
Пуленепробиваемый подход	224
Преимущества пуленепробиваемого подхода	256
Верстка на основе ем	256
Резюме	264
Глава 9. Сводим воедино	267
Цель	269
Пуленепробиваемый подход	270
Структура	276
Заключение	302

Книга посвящается Джеку и Тенли

БЛАГОДАРНОСТИ

Выражаю благодарность Майклу Нолану за то, что он познакомил меня с издательством New Riders и помог согласовать концепцию книги. Также благодарю Марджори Баер за руководство и поддержку на всем протяжении работы над этим проектом.

Спасибо Дэйву Робертсу за то, что нашел для меня время.

Я благодарен всем сотрудникам Reachpit, которые приняли участие в создании книги, и в особенности — Ребекке Гулик за руководство проектом и за то, что она смогла максимально отладить весь процесс.

Благодарю Итана Маркотта за то, что он стал отличным научным редактором первого издания книги. Мне было очень комфортно работать с Итаном над книгой, а его комментарии и идеи были бесценными. Спасибо Вирджинии Де-Болт за работу над вторым изданием, а Патрику Х. Локе — за научную редакцию третьего издания.

Благодарю коллег, которые вдохновляют как в виртуальной, так и в реальной жизни. Данная книга не увидела бы свет без самоотверженного труда целого коллектива (а также множества людей, о которых я забыл упомянуть здесь): Джона Оллсоппа, Дэна Бенджамина, Холи Берджвина, Дугласа Боумана, Энди Бадда, Тантека Селика, Джо Кларка, Энди Кларка, Майка Дэвидсона, Тодда Домини, Джейсона Фрида, Джона Галанта, Патрика Гриффитса, Джона Хиккса, Молли Хольцшлаг, Шона Инмана, Райана Айрлана, Ричарда Ишиды, Роджера Йоханссона, Джереми Кита, Яна Ллойда, Дрю МакЛеллана, Эрика Мейера, Кэмерона Мола, Мэтта Мулленвега, Марка Ньюхайза, Данстана Орхарда, Вирле Питерса, Д. Кита Робинсона, Ричарда Раттера, Джейсона Санта Марии, Кристофера Шмита, Дэйва Ши, Райана Симса, Грега Стори, Джеффри Вина, Джоша Уильямса и Джеффри Зельдмана.

Благодарю читателей и клиентов SimpleBits за то, что они позволяют мне заниматься любимым делом.

Благодарю участников лиги Greater Beverly Adult Dodgeball за то, что позволяли мне уходить каждый вторник в 19:30.

Спасибо родителям, которые всегда поддерживают мои начинания, брату Мэтту и его домочадцам, а также остальным членам нашей большой семьи и друзьям.

Я очень благодарен моему верному боевому товарищу — супруге Керри.

Также благодарю вас за то, что вы читаете эту книгу.

ВВЕДЕНИЕ



Должен признаться: не существует абсолютно пуленепробиваемого веб-сайта. Однако это не означает, что книга будет для вас бесполезна.

Как полицейский, который надевает пуленепробиваемый жилет для защиты от пуль, мы тоже можем принять меры для защиты веб-дизайна. В этой книге представлено несколько стратегий защиты веб-сайтов: от повышения гибкости до подготовки к наихудшим ситуациям.

Концепция пуленепробиваемого дизайна

В реальном мире пуленепробиваемый жилет не гарантирует полной защиты, однако от него никто не отказывается. Ведь гораздо лучше надевать жилет, чем не надевать.

Это же применимо к веб-дизайну и методикам, описанным в этой книге. Повышая гибкость веб-сайта и выполняя действия для обеспечения читаемости сайта в разных условиях, мы значительно влияем на результат нашей работы. Это постоянный процесс, который можно упростить благодаря использованию некоторых веб-стандартов, например семантического HTML и CSS-кода, для разработки привлекательных адаптивных сайтов.

Так как макеты на базе CSS становятся все более распространенными, важно научиться эффективно использовать CSS. Цель — использовать преимущества, благодаря которым технология становится мощным инструментом дизайна. Например, такими преимуществами могут быть меньшее количество кода, больше удобства, более простое обслуживание.

Однако использование только CSS и HTML не означает, что веб-сайт автоматически станет лучше. Добавляя к этому гибкость, которую дает разделение контента и дизайна, вы сможете разрабатывать более эффективные дизайны для любых пользователей сети.

Актуальность вопроса

Приблизительно в то же время, когда я начал думать над темой книги, я понял, что для разработки эффективного и привлекательного веб-дизайна необходимы два компонента. Один из них — визуальный компонент, который бросается в глаза любому пользователю, когда он смотрит на страницу. Это графическое оформление, цвета и шрифтовое оформление, которые выбирает дизайнер. Посетите сайт CSS Zen Garden (www.csszengarden.com), вам станет очевидно, что HTML и CSS позволяют разработать привлекательный визуальный дизайн.

Второй (но такой же важный) компонент процесса разработки веб-сайта — это его пуленепробиваемая реализация. Именно на этом компоненте мы подробно остановимся в книге. Вы вполне разумно решили использовать HTML и CSS для создания веб-сайтов, что позволит использовать преимущества, предлагаемые этими технологиями, и готовы использовать данные веб-стандарты для создания привлекательных, гибких, адаптируемых и удобных веб-сайтов.

Так как HTML и CSS применяются все более активно, очень важно обладать ресурсами, которые помогают использовать и реализовывать их возможности наиболее оптимально.

СТРУКТУРА КНИГИ

В каждой главе описывается конкретный принцип пуленепробиваемого дизайна. Мы начнем с анализа представленных в сети вариантов дизайна и укажем, почему они не являются пуленепробиваемыми. После этого мы переработаем код с использованием HTML и CSS, чтобы повысить его гибкость и сократить количество используемого кода.

Многие из представленных в книге примеров включают определенные компоненты страницы. Они помогают лучше объяснить способы защиты. В последней главе «Сводим воедино» мы используем все методики, изученные в главах книги, для разработки макета страницы и вспомним, почему мы выбрали именно эти методики.

В каждой главе примеры представлены поэтапно. Так их проще анализировать, даже если вы недавно стали использовать HTML и CSS в работе. Попутно я буду объяснять, какие преимущества дают указанные веб-стандарты и как конкретно каждый принцип, представленный в главе, может повысить пуленепробиваемость веб-сайта.

ПРИМЕЧАНИЕ

Я использую термин «пуленепробиваемость» отчасти для того, чтобы описать гибкость макета. Иными словами, гибкие дизайны — это дизайны веб-сайтов, которые могут вмещать текст разного размера и любое количество контента; дизайны, которые могут расширяться или сжиматься вместе с контентом.

Кроме того, мы будем говорить о гибкости с точки зрения редактирования, поддержки и развития сайта. Мы упростим процесс редактирования контента, а также обновления и поддержки кода так, чтобы они не влияли на целостность дизайна. Это особенно нужно и важно при решении вопросов интернационализации сайтов, когда объем контента может значительно различаться.





И наконец, мы также будем говорить о гибкости с точки зрения системы. Как дизайн будет влиять на целостность контента веб-сайта и его функциональность? Мы убедимся, что результат нашей работы можно адаптировать для разных ситуаций — для просмотра через браузер, приложение для чтения с экрана или мобильное устройство. Разработка веб-сайта с учетом гибкости означает более эффективную интерпретацию кода разными устройствами и приложениями.

КОНТЕКСТ ПРИМЕРОВ В КНИГЕ

Каждый пример предполагает использование базовой структуры. Та часть HTML и CSS-кода, которая показана в каждом примере, вставлена в код уже существующего документа HTML между тегами `<body>` и `</body>`.

Например, примеры в этой книге имеют следующую базовую структуру:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Заголовок страницы</title>
  <meta charset="utf-8" />
  <style type="text/css">
    ... здесь будет пример CSS ...
  </style>
</head>
<body>
  ... здесь будет пример разметки ...
</body>
</html>
```

Так как CSS-код помещается в заголовок страницы только для удобства, его можно (и, вероятно, нужно) переместить в отдельный файл.

HTML5

Все примеры в книге построены на базе стандарта HTML5, поэтому в книге мы будем использовать новые элементы стандарта HTML5. Можно начать использовать HTML5 прямо сейчас, используя заголовок `DOCTYPE`, как показано выше.

Однако нужно предпринять два небольших шага, чтобы гарантировать, что новые элементы HTML5 будут «узнаваться» всеми браузерами. Новые элементы, которые не существовали до разработки стандарта HTML5, не распознаются браузерами Internet Explorer версий 8 и старше. Самый простой способ «рассказать» браузерам о новых элементах — это использовать небольшой фрагмент

кода JavaScript, разработанный Реми Шарпом (<http://remysharp.com/2009/01/07/html5-enabling-script/>).

Добавьте этот код в заголовок `<head>` документа, который условно загрузит его (из Google) в браузер IE8 и старше, позволяя использовать новые элементы HTML5 в IE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Заголовок страницы</title>
  <meta charset="utf-8" />
  <style type="text/css">
    ... здесь будет пример CSS ...
  </style>
  <!-- разрешить элементы HTML5 в IE7+8 -->
  <!--[if lt IE 9]>
  <script src="http://html5shim.googlecode.com/svn/trunk/html5.js">
    </script>
  <![endif]-->
</head>
```

СБРОС СТИЛЕЙ

Чтобы обнулить поля, отступы и другие элементы, которые браузеры определяют по умолчанию, я часто использую «обнуленную таблицу стилей» — набор правил CSS для создания базы, которую можно использовать для разработки стилей. Такие стили располагаются раньше остальных или наверху основной таблицы стилей, либо подключены до других таблиц при использовании внешних таблиц стилей.

Рекомендую использовать версию Эрика Мейера, которую он обновляет регулярно: <http://meyerweb.com/eric/tools/css/reset>.

Важным этапом работы с обнуленной таблицей стилей при использовании HTML5 является объявление новых элементов HTML5 со свойством `display: block`; для старых браузеров, которые в данный момент о них не знают (в противном случае вы можете получить нежелательные результаты).

Ниже представлена последняя (на момент написания книги) версия обнуленной таблицы стилей от Эрика. В коде я отметил наиболее важные фрагменты HTML5:

```
/* http://meyerweb.com/eric/tools/css/reset/
   Вер. 2.0 | 20110126
   Лицензия: нет (публичный доступ)
*/
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
```





```
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
/* сброс HTML5 display-role для старых браузеров */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
    content: '';
    content: none;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}
```

Таким образом, выполнив этот JS-код для HTML5 и сбросив стили, вы готовы использовать примеры, представленные в книге. Такие установки предполагаются для каждого примера. Это поможет нам сконцентрироваться только на важных аспектах, не повторяя одно и то же каждый раз.

ТЕРМИНЫ, ИСПОЛЬЗУЕМЫЕ В КНИГЕ

Иногда в книге я называю разные версии браузеров сокращенно. Например, гораздо проще сказать IE6/Win, чем «Internet Explorer версии 6 для Windows». Вот несколько из таких сокращений:

- IE = Internet Explorer для Windows;
- WebKit = движок браузера, на базе которого разработаны Safari и Chrome;
- Mozilla = движок браузера, на базе которого разработан Firefox.

При описании старых подходов, о которых я буду рассказывать в примерах, я часто использую понятия «вложенные таблицы» и «GIF-разделители». Они означают общепринятые методики, часто используемые в процессе разработки веб-сайтов. В этих методиках для разработки дизайнов с точностью до пиксела (и очень негибких) использовались таблицы. Вложенные таблицы упрощают точное размещение графики и текста, хотя результатом становится гигантское количество кода вкупе с проблемой доступности.

Термин «GIF-разделитель» означает одно прозрачное GIF-изображение, которое растягивается в разных направлениях для создания пустого пространства, колонок и разделяющих элементов на странице. Веб-сайт, не обладающий пуленепробиваемостью, содержит такие элементы в разметке, делая добавление кода и обслуживание сайта сплошным кошмаром.

Однако есть более эффективные способы добиться визуально такого же результата с помощью экономичной значимой разметки и CSS. Применяя эти веб-стандарты, мы можем разрабатывать привлекательный дизайн, который является гибким и готовым к работе в любой ситуации. Такой принцип называется пуленепробиваемым дизайном.

ОТ ИЗДАТЕЛЬСТВА

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

Мы будем рады узнать ваше мнение!

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

ГЛАВА 1

ГИБКОЕ УПРАВЛЕНИЕ ТЕКСТОМ



Ключевые слова, проценты или единица ем обеспечат максимальную гибкость вашему дизайну, а пользователю дадут возможность настраивать внешний вид страницы под себя

Немногие из дизайнерских задач столь противоречивы, как управление размерами текста на сайте. Этот вопрос беспокоит новичков и является постоянной темой для споров. Возможно, я слегка преувеличиваю, но этот вопрос действительно актуален.

Не претендуя на то, чтобы положить конец этой «войне за размер», в настоящей главе я предлагаю вашему вниманию две гибкие и простые стратегии. Вы сможете использовать их для задания кегля шрифта на веб-сайте и полного контроля над дизайном.

Гибкость настройки текста является одним из ключевых принципов, на котором базируются все примеры в настоящей книге: давая пользователю возможность управлять кеглем шрифта на странице, мы повышаем ее удобочитаемость для всех пользователей. Однако достижение такой гибкости одновременно с разработкой оригинального дизайна — это по-настоящему сложная задача. К концу книги вы сможете научиться решать ее, используя солидный багаж знаний в виде представленных здесь примеров.

Для того чтобы четко понимать, как наилучшим образом обеспечить гибкость при определении кегля шрифта, предлагаю рассмотреть пример и проанализировать его недостатки с точки зрения гибкости.

ОБЩЕПРИНЯТЫЙ ПОДХОД

Чтобы проиллюстрировать общепринятый подход при задании кегля текста, возьмем веб-сайт [eyebuydirect.com](http://www.eyebuydirect.com). Несомненно, пользователи, имеющие проблемы со зрением, найдут приятным для чтения сайт, на котором возможно управлять внешним видом страницы (рис. 1.1).

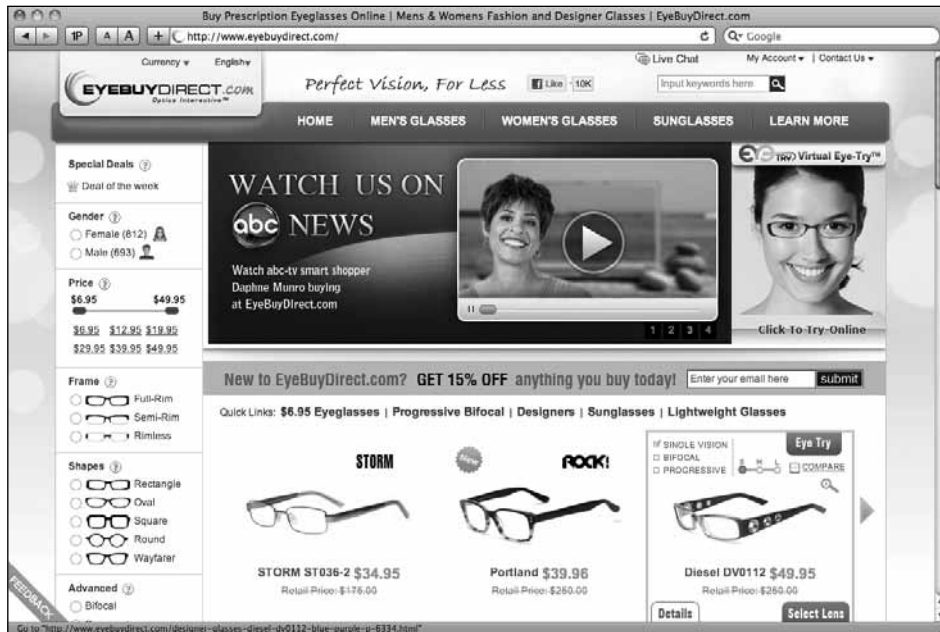


Рис. 1.1. Для иллюстрации общепринятого подхода возьмем веб-сайт [eyebuydirect.com](http://www.eyebuydirect.com) (скриншот сделан в сентябре 2011 года)

ПРИМЕЧАНИЕ

Я вовсе не собирался критиковать дизайнеров [eyebuydirect.com](http://www.eyebuydirect.com). На самом деле сайт довольно неплох с точки зрения дизайна, здесь используются структурированная разметка и CSS. Именно поэтому я и выбрал его. Мне просто был нужен сайт, на примере которого можно поговорить о принципах задания кегля текста, а также обсудить плюсы и минусы, связанные с принятым среди дизайнеров подходом.

На сайте eyebuydirect.com CSS удачно использован во всех элементах дизайна. Как и на многих других хороших сайтах, свойство `font-size` в теге `<body>` позволяет управлять базовым кеглем текста на странице (в пикселах):

```
Body {  
    font-size: 12px;  
}
```

Задавая в теге `<body>` кегль шрифта, дизайнеры гарантируют, что текст будет иметь высоту 12 пикселей — до тех пор, пока это правило не будет отменено другим. Преимущество данного подхода в том, что текст всегда имеет установленный размер, независимо от того, какой браузер или устройство вы используете для просмотра страницы. Именно из-за постоянства и предсказуемости данный способ является таким популярным. При задании абсолютных размеров текста дизайнеры (если требуется точность) предпочитают использовать в качестве единицы измерения пиксели. Однако с этим связана небольшая проблема.

Уязвимые места

Указание фиксированного кегля шрифта в пикселах предоставляет дизайнеру возможность управлять размером текста, но создает проблему для пользователей Internet Explorer под Windows (IE/Win).

Как правило, браузеры позволяют изменять кегль текста на странице, игнорируя установки дизайнера. Это довольно удобно для людей с ослабленным зрением. Для повышения комфорта чтения пользователь может выбрать более крупный размер шрифта (рис. 1.2).

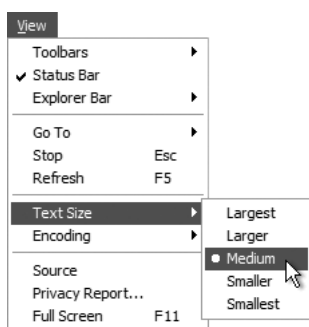


Рис. 1.2. Меню Text Size (Размер шрифта) в IE/Win позволяет пользователю изменять размер текста на странице — увеличивать или уменьшать

Выглядит отлично, но не для пользователей Internet Explorer под Windows — они не смогут масштабировать текст, если дизайнер указал фиксированный кегль

в пикселах. В то же время пользователи других браузеров могут свободно изменять величину текста, независимо от того, какой способ использовал дизайнер.

В Internet Explorer 7 используется функция масштабирования (рис. 1.3), которая позволяет увеличивать страницу целиком, а не только текст. Изменить масштаб страницы можно с помощью клавиш **Ctrl+«+»** («плюс») или **Ctrl+«-»** («минус») или значка в нижней части браузера. Недостатком данного метода является то, что для просмотра увеличенной страницы могут потребоваться горизонтальные и вертикальные полосы прокрутки.

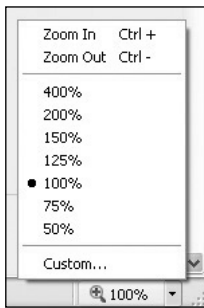


Рис. 1.3. Функция масштабирования в Internet Explorer 7

Изменять кегль текста может потребоваться не только пользователям с проблемами со зрением, ведь таким образом повышается удобочитаемость веб-страниц с мелкими текстовыми блоками. Естественно, дизайнер хочет получить полный контроль над внешним видом страницы. Однако передав часть контроля пользователю (хотя бы над размером текста), вы сможете сделать страницу более удобной и гибкой.

Почему же в IE/Win (вплоть до 9-й версии) нельзя было изменять точно указанный кегль? Скорее всего, разработчики предполагали, что пиксел — это слишком «правильная» единица, чтобы позволить пользователю влиять на установки. Однако в большинстве браузеров процедура обработки команды реализована по-другому, что позволяет изменять даже фиксированный кегль текста с помощью функций, доступных пользователю.

Всем было бы лучше, если бы дизайнеры задавали кегль шрифта в пикселах, а при необходимости пользователи могли его изменить. Однако из-за ограничений IE/Win сделать это невозможно, а значит, такое решение не будет оптимальным. Именно поэтому я предлагаю рассматривать альтернативные способы изменения размера текста. Кроме того, я рекомендую стратегию, которая позволяет отказаться от такой точности, но дает вам важное преимущество — масштабируемость текста во всех браузерах.

ПРИМЕЧАНИЕ

В настоящее время большинство браузеров (включая IE) поддерживают масштабирование, что дает возможность изменять размеры всех элементов на странице, а не только текста. Однако кегль текста имеет большее значение для всех пользователей, а особенно для людей, испытывающих трудности со зрением, так как не влияет на расположение остальных элементов страницы. Масштабирование же может привести к тому, что размер страницы станет больше окна браузера, и для просмотра понадобятся горизонтальные полосы прокрутки.

АЛЬТЕРНАТИВНЫЕ СПОСОБЫ

Помимо фиксированного кегля текста в пикселах, CSS имеет и другие способы управления размерами шрифта.

Значения свойства `font-size` в CSS можно разделить на четыре основные группы.

ПРИМЕЧАНИЕ

С технической точки зрения, пиксел — относительная единица измерения, связанная с разрешением устройства, используемого для просмотра или печати. Фактический размер пиксела зависит от того, каким образом просматривается страница — на мониторе компьютера или распечатанная на бумаге, по сравнению с другими элементами (более подробную информацию см. www.w3.org/TR/REC-CSS2/syndata.html#pizel-units).

РАЗМЕР ШРИФТА

Размер шрифта может быть как относительным, так и абсолютным. Ниже перечислены способы задания относительных размеров шрифта:

- `em` позволяет задать размеры пропорционально используемому шрифту;
- `ex` — пропорционально высоте буквы `x` конкретного шрифта;
- `px` — пропорционально разрешению экрана (наиболее распространенный способ);
- `rem` — значение совпадает с размером текста, рассчитанным по корневому элементу.

Абсолютные единицы измерения используют преимущественно для вывода на печать или когда характеристики и свойства браузера и (или) устройства неизвестны:

- in — дюймы;
- cm — сантиметры;
- mm — миллиметры;
- pt — пункты;
- pc — пики.

Ключевые слова для задания относительного размера

Ключевые слова для задания относительного размера

- larger (англ. «больше»),
- smaller (англ. «меньше»)

позволяют увеличивать и уменьшать кегль шрифта относительно унаследованного значения. Согласно W3C, это означает, что если родительский элемент имеет размер `medium`, то при использовании ключевого слова `larger` кегль шрифта текущего элемента увеличивается до `large` (более подробную информацию см. www.w3.org/TR/CSS21/fonts.html#font-size-props).

Используйте эти ключевые слова точно так же, как поступали со `<small>` и `<big>` в HTML. Чаще всего они используются как команды, учитывающие размеры текста, которые были определены в документе ранее.

Проценты

Проценты позволяют задать относительный размер шрифта. Указав значение `120%`, вы тем самым добавляете 20% к унаследованному значению. Именно проценты (в сочетании с ключевыми словами) лежат в основе пуленепробиваемого подхода к созданию веб-сайта.

Ключевые слова для задания абсолютного размера

Ключевые слова для задания абсолютного размера ссылаются на таблицу кеглей шрифтов браузера или устройства. Существуют следующие ключевые слова:

- `xx-small`;
- `x-small`;
- `small`;
- `medium`;
- `large`;
- `x-large`;
- `xx-large`.

W3C предлагает разработчикам браузеров использовать постоянный коэффициент увеличения, равный 1.5, независимо от физического размера текста. Это означает, что `large` будет всегда в полтора раза больше `medium`, а `small`, соответственно, в полтора раза меньше `medium`.

Простой синтаксис — одно из преимуществ использования ключевых слов для задания абсолютного размера. На этом я более подробно остановлюсь в следующем разделе.

После того как мы рассмотрели все значения свойства `font-size`, можно приступить к обсуждению стратегии, обеспечивающей гибкость и дающей пользователю контроль над кеглем текста.

ПУЛЕНЕПРОБИВАЕМЫЙ ПОДХОД

Хочу предложить вам стратегию, которая уже доказала свою эффективность. В этой стратегии для задания абсолютного размера используются как ключевые слова, так и проценты. Использование ключевых слов в свойстве `font-size` решает проблему, о которой я рассказывал в примере с `eyebuydirect.com`. В отличие от ситуаций, когда кегль шрифта задается в пикселах, при реализации данной стратегии браузеры позволят изменять размер шрифта на странице независимо от того, какой браузер или устройство используется.

КАК РАБОТАЮТ КЛЮЧЕВЫЕ СЛОВА?

Напоминаю, что мы можем использовать семь ключевых слов: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` и `xx-large`. Ниже представлен пример использования ключевого слова при задании размера шрифта в элементе `<body>`:

```
body{
  font-size: small;
}
```

Ключевое слово определяет размер шрифта относительно текущих настроек браузера. Иными словами, если пользователь изменил настройки браузера — уменьшил или увеличил кегль текста по сравнению с используемым по умолчанию, — ключевое слово будет управлять масштабом, ориентируясь на новый базовый размер. Коэффициент масштабирования остается постоянным и не зависит от базового размера шрифта.

Например, на рис. 1.4 показано соотношение размеров текста при использовании разных масштабирующих ключевых слов (результат зависит от кегля шрифта, установленного в браузере по умолчанию). В данном примере я предпочел браузер Safari.

Обратите внимание на то, что хотя параметр `small` будет меняться в зависимости от базового размера, соотношение между текстом с атрибутом `small` и любым другим масштабирующим ключевым словом остается неизменным.

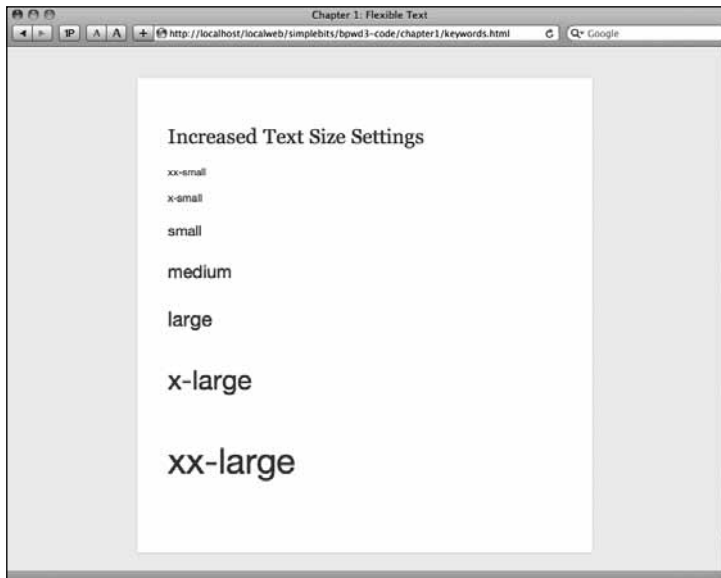


Рис. 1.4. Ключевые слова в Safari. Внизу базовый текст имеет больший размер

ОТКАЗЫВАЕМСЯ ОТ ПИКСЕЛОВ

Основная проблема, которую приходится решать любому дизайнеру при использовании ключевых слов (и вообще любых иных единиц измерения помимо px), заключается в том, что в зависимости от браузера, операционной системы и настроек одни и те же значения могут давать разные результаты.

Например, размер текста с атрибутом `small`, что я проиллюстрировал ранее, будет слегка различаться в разных браузерах и/или операционных системах, даже если во всех случаях использовать одинаковые значения по умолчанию.

В то время как утверждение «пиксел — это всегда пиксел» можно считать веб-законом (впрочем, это не относится к устройствам с высоким разрешением, например Retina-дисплею iPhone), ключевое слово здесь в большей степени — *инструкция* для браузера или устройства, которая определяет, шрифт какого размера они должны отобразить. Если подойти к ее использованию с некоторой долей изобретательности и гибкости, то ваши веб-страница (и, конечно же, пользователи) только выиграют.

ПРЕИМУЩЕСТВА ПУЛЕНЕПРОБИВАЕМОГО ПОДХОДА

Задавая кегль шрифта с помощью относительных параметров (например, ключевых слов), мы отталкиваемся от базового пуленепробиваемого размера. Вместо того чтобы задать значения свойства `font-size` в пикселах, как это реализовано на eyebuydirect.com и миллионах других сайтов, можно использовать атрибут `small` в теге `<body>` для задания значения по умолчанию. Используя ключевое слово, мы гарантируем, что любые браузеры и устройства (включая IE/Win) при необходимости смогут отменить эту инструкцию.

Это важно не только для слабовидящих пользователей, которым для работы в Интернете приходится покупать новые очки, но и для всех, кто ищет новые рецепты, приобретает товары, читает статьи или выполняет другие задачи. Скорее всего это незначительное изменение останется незамеченным, но даст много новых возможностей всем категориям пользователей.

Что дальше?

Пока что мы только узнали, как задавать значения свойству `font-size` для всей страницы. Вполне естественно предположить, что нам может понадобиться возможность задавать кегль шрифта для отдельных элементов страницы — заголовков, списков, подзаголовков, элементов навигации и таблиц. Ведь размер

каждого из них, скорее всего, будет отличаться от размеров текста по умолчанию. Возникает следующий вопрос: каков наилучший способ задать размер шрифта меньше или больше базового?

Лично я рекомендую способ, который считаю самым понятным и простым. Он позволяет легко изменять, добавлять или корректировать размеры шрифтов на протяжении всего существования вашего веб-сайта.

СДЕЛАТЬ И ЗАБЫТЬ

Мы используем только одно ключевое слово для задания абсолютного размера в таблице стилей — для базового текста в элементе `<body>`:

```
body {  
    font-size: small;  
}
```

СОВЕТ

В большинстве браузеров значению `medium` по умолчанию соответствует кегль около 16 пикселей.

Обратите внимание на то, что получающийся кегль текста очень близок к 12 пикселям, использованным на сайте eyebuydirect.com (рис. 1.5). В разных браузерах размеры могут различаться — результат зависит от того, как приложение обрабатывает ключевое слово. Однако в большинстве случаев значение `small` задает базовый размер, достаточно близкий к 12 пикселям (при условии, что пользователь не изменял настройки браузера по умолчанию). Иногда эта информация позволяет дизайнеру найти «точку отсчета», от которой он может отталкиваться при работе с ключевыми словами.

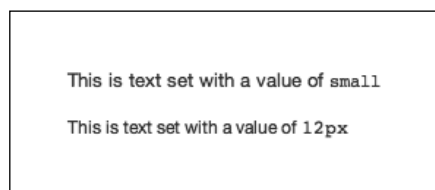


Рис. 1.5. Сравните строку, созданную с помощью ключевого слова `small` (*вверху*), и текст с точно заданным кеглем 12 пикселей (*внизу*) в браузере Safari (OS X). При использовании настроек браузера по умолчанию результаты будут очень близки

ПРОЦЕНТНЫЕ СООТНОШЕНИЯ ДЛЯ ИЗМЕНЕНИЯ КЕГЛЯ ШРИФТА ОТНОСИТЕЛЬНО БАЗОВОГО ЗНАЧЕНИЯ

Используя шрифт с базовым размером `small`, мы можем увеличивать или уменьшать его с помощью процентов. Проценты — простой способ получения необходимого кегля для конкретного элемента. На рис. 1.6 показано, как именно процентное соотношение позволяет увеличивать или уменьшать текст относительно базового.

Например, если мы хотим, чтобы элементы `<h1>` были больше, чем `small` (размер, заданный по умолчанию), можно использовать следующее описание:

```
body {  
    font-size: small;  
}  
h1 {  
    font-size: 150%;  
}
```

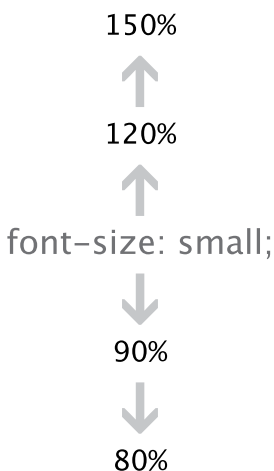


Рис. 1.6. Процентные соотношения можно использовать как для увеличения, так и для уменьшения размера текста относительно базового значения

Так как элемент `<h1>` всегда располагается внутри элемента `<body>`, мы гарантируем, что кегль `<h1>` всегда будет составлять 150% от `small` (рис. 1.7).

Иногда нам может понадобиться, чтобы некоторые абзацы текста были меньше заданного по умолчанию размера, в данном случае — меньше `small`. На рис. 1.8 я применил созданный класс `note` к небольшому абзацу:

```
<p class="note">Это класс "note", который я хочу сделать немного меньше  
основного текста.</p>
```

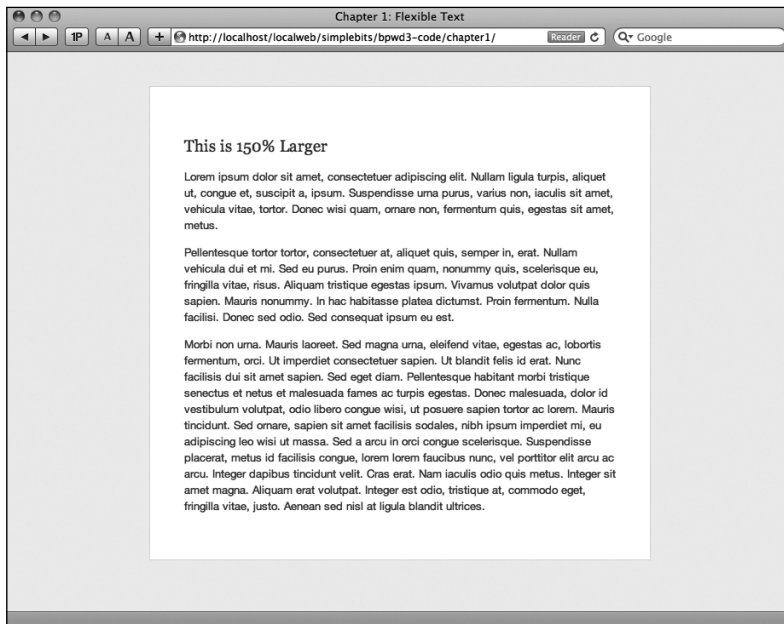


Рис. 1.7. Для элемента `<h1>` указан размер 150% относительно базового small

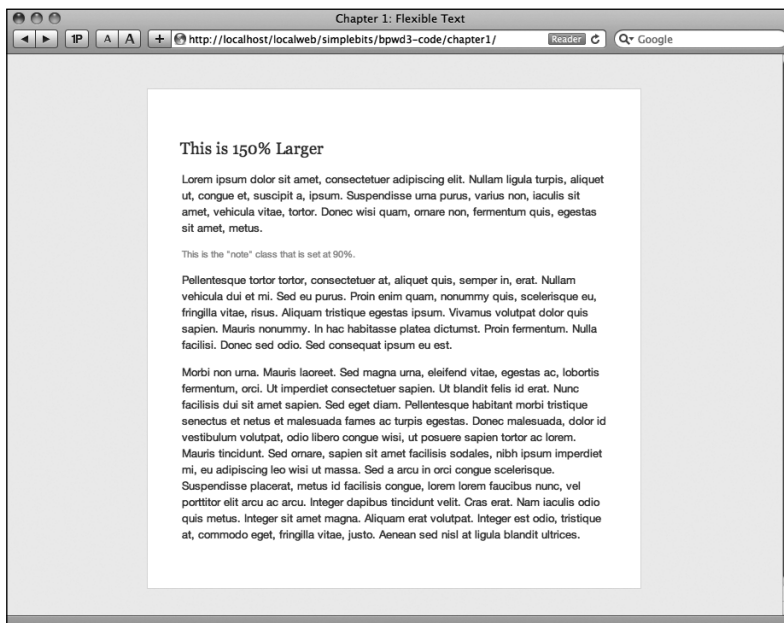


Рис. 1.8. Класс `note` окрашен в серый цвет и имеет размер 90% от базового small

Я хочу сделать шрифт для этого абзаца меньше значения, заданного по умолчанию. Для этого я объявляю класс, в котором определяю значение меньше 100, что и будет соответствовать уменьшению текста:

```
.note {  
    font-size: 90%;  
}
```

Для любого элемента на странице можно определить процентное значение размера, и неважно, какого размера мы хотим сделать этот элемент (меньше или больше базового, определенного в `<body>`).

Упрощенная таблица стилей, в которой размеры некоторых элементов определены в виде процентных соотношений, может иметь следующий вид:

```
body {  
    font-size: small;  
}  
h1 {  
    font-size: 150%;  
}  
h2 {  
    font-size: 130%;  
}  
h3 {  
    font-size: 120%;  
}  
ul li {  
    font-size: 100%;  
}  
.note {  
    font-size: 90%;  
}
```

Мы задали значения для трех уровней заголовков, уменьшили кегль текста маркированного списка и добавили класс `note` для элементов, размер которых должен быть меньше заданного по умолчанию значения `small`. Конечно, это довольно примитивный пример, но он нужен для иллюстрации двухэтапной концепции: 1) определить базовый кегль и 2) использовать процентные соотношения, описывающие увеличение или уменьшение кеглей других текстовых элементов. После изменения базового кегля все остальные элементы изменяются автоматически.

Преимущество модели: если вам когда-либо потребуется изменить значение, заданное по умолчанию, то понадобится поправить только одно объявление в элементе `<body>`. Все остальные значения, установленные в виде процентных соотношений, завязаны на кегле базового элемента. Таким образом, изменение базового размера со `small` на `large` повлияет на все остальные элементы.

Несомненно, это очень удобно, даже если вы решите изменить базовое значение, задав иную единицу измерения.

По этой же причине модель будет удобна пользователям, которые создают собственные таблицы стилей, переопределяющие установки сайта по умолчанию. Изменение базового размера сразу повлияет на всю страницу.

КЛЮЧЕВЫЕ СЛОВА И ПРОЦЕНТЫ

Существует несколько правил, необходимых для более эффективного использования комбинаций ключевых слов и процентов. Далее вы научитесь точной настройке процентных соотношений. Кроме того, я расскажу вам, на что необходимо обращать особое внимание при использовании «вложенных» процентов.

НЕТ ПОДХОДЯЩЕГО КЛЮЧЕВОГО СЛОВА?

Хороший способ задать базовый кегль текста — подстроить значение ключевого слова через контейнер `<div>`. Довольно часто при использовании CSS для разработки макетов `<div>` позволяет реализовывать дизайнерские решения (см. главу 8 «Резиновые и эластичные макеты»). Тем не менее его можно использовать и для управления значением базового ключевого слова (действующего на уровне всей страницы) с помощью процентов.

Для примера предположим, что у вас есть следующий код:

```
<body>
  <div id="container">
    <h1>Это заголовок</h1>
    <p>Это основной текст базового размера.</p>
  </div>
</body>
```

Мы можем не только изменить ключевое слово для базового значения, но и задать увеличенное процентное значение для заголовка `<h1>`:

```
body {
  font-size: small;
}
h1 {
  font-size: 150%;
}
```

Если нам хочется уменьшить кегль текста `small` (если учитывать настройки браузера по умолчанию), то это можно осуществить, изменив процентное значение в объекте `#container`, который оказывает влияние на текст всей страницы, например можно сделать его меньше:

```
body {  
    font-size: small;  
}  
h1 {  
    font-size: 150%;  
}  
#container {  
    font-size: 95%;  
}
```

Посмотрите на рис. 1.9, мы немного уменьшили текст по сравнению со значением по умолчанию, и это изменение повлияло на размер заголовка страницы.

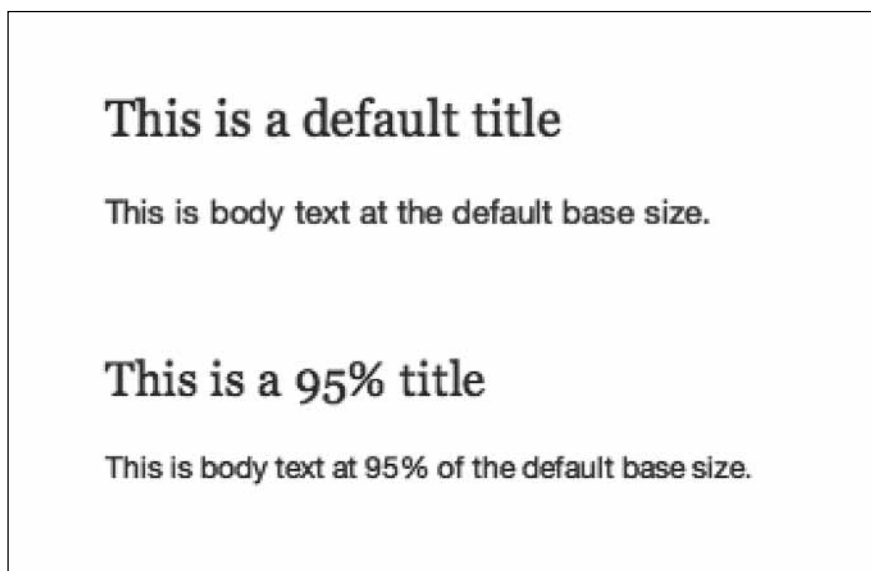


Рис. 1.9. Базовое ключевое слово `small` в сравнении с базовым значением, уменьшенным на 95% с помощью контейнера

СОВЕТ

Вы можете не использовать контейнеры `<div>`, а задать базовый кегль текста в `<html>`, указав меньшее процентное значение в элементе `<body>`.

Для фрагмента внизу мы добавили правило задания `font-size` для объекта `#container`, указав значение 95%, а вверху использовали ключевое слово `small`. Разница практически незаметна.

Этот способ очень удобен, если при задании кегля текста требуется точность, а необходимое значение оказывается в промежутке между ключевыми словами. Используя контейнер `<div>`, который действует на всю страницу, вы сможете подстраивать значение ключевого слова (увеличивая или уменьшая). Таким образом, можно создать новый базовый кегль текста, который будет соответствовать вашим требованиям.

Не следует обращать чрезмерное внимание на точность: небольшая вариативность, связанная с использованием разных браузеров и платформ, вполне допустима. Важно только то, что пользователи получают контроль над кеглем текстовых элементов с помощью встроенных функций браузера.

ОПАСНОСТЬ МНОГОУРОВНЕВЫХ ВЛОЖЕНИЙ

Вернемся к предыдущему примеру, когда мы задавали базовое ключевое слово `small` для всей страницы, а затем уменьшали значение до 95% с помощью контейнера `<div>`. Необходимо проявлять особенную осторожность при работе с «вложенными» процентами. Задание процентного значения через объект `#container` может усугубить ситуацию.

Допустим, мы объявим, что все заголовки `<h1>` должны иметь значение 150%. А так как элементы `<h1>` расположены ниже по иерархии (относительно объекта `#container`), то в действительности мы получаем не 150% от исходного значения `small`, а 150% от 95% от значения `small`. На рис. 1.10 показано, как вложение процентов соотносится с вложением элементов в документах с древовидной структурой.

Сложно? Возможно, вам будет проще разобраться, если заменить ключевое слово `small` на конкретное числовое значение.

Элемент

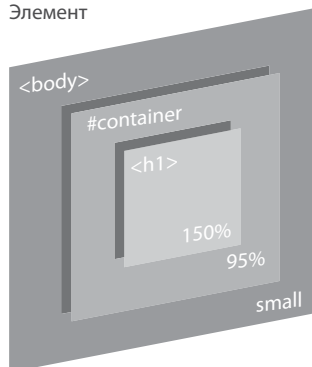


Рис. 1.10. Так рассчитываются процентные значения для элементов при использовании вложений

Если базовый кегль в элементе `<body>` соответствовал 10 (а не `small`), тогда внутри `#container` кегль превратится в 9,5 (или 95% от 10). Заголовки `<h1>` находятся внутри `#container`, а в свойстве `font-size` мы указали для этих элементов значение 150%. Для базового размера, равного 10, кегль заголовка — 15, однако с учетом `#container`, 150% от 9,5 даст 14,25. Если используется многократная вложенность, то расчеты оказываются еще сложнее. В общем, мораль истории такова: при объявлении новых процентных значений нужно быть осторожнее, необходимо учитывать предыдущие процентные настройки. Я редко использую более одного-двух уровней, но понимание того, как вложенность влияет на проценты, поможет избежать ситуации, когда итоговые размеры текста не будут соответствовать вашим ожиданиям.

СОВЕТ

Будьте осторожны с вложенностью, когда указываете проценты менее 100: если пользователь употребит возможности браузера, чтобы уменьшить кегль шрифта, существует вероятность того, что текст станет нечитаемым.

Иными словами, когда вы работаете в расчете на настройки по умолчанию и использовали значение (в процентах), делающее текст более мелким, но читаемым, вполне возможно, что пользователь не сможет прочитать этот блок, если он изменял настройки браузера по умолчанию. Некоторые браузеры устанавливают ограничения минимального кегля, но это вам неподвластно и зависит от конкретного браузера.

Необходимо протестировать настройки в разных браузерах и с разными сценариями изменения размера шрифтов, чтобы убедиться в читаемости текста, независимо от того, какой способ управления кеглями (ключевое слово или процентное соотношение) вы использовали. Этот совет будет полезен, даже если вы используете какие-либо другие способы.

ЭКСПЕРИМЕНТЫ С ПРОЦЕНТНЫМИ СООТНОШЕНИЯМИ

Эксперименты с процентами позволяют обеспечить единообразие отображаемого текста в разных браузерах и на разных платформах. Учитывая, что пользователи просматривают страницу с настройками браузера по умолчанию (настройки, изменяющие размеры текста, отключены), очень важно удостовериться, что ваши эксперименты с процентными значениями, особенно в сторону уменьшения, работают корректно.

Давайте рассмотрим определенный нами класс `note` как пример. Можно сравнить, как изменится свойство `font-size` этого класса при 110 и 115%. Некоторые браузеры могут работать только со значениями процентов, кратными 10 (70, 80,

90 и т. п.), а другие воспринимают только те числа, которые заканчиваются на 5 (75, 85, 95 и т. п.).

На рис. 1.11 и 1.12 показаны страницы в браузерах Safari (*слева*) и IE9/Win (*справа*). Различия между 110% (*вверху*) и 115% (*внизу*) в Safari не бросаются в глаза. В то же время в IE9/Win незначительное увеличение текста более заметно. В данном примере я бы остановился на значении 110%, так как итоговые размеры в двух популярных браузерах будут достаточно близкими.

Поэкспериментируйте с заданием различных значений в процентах для свойства `font-size`. Небольшое изменение (например, 5%) позволит добиться единообразия отображения в разных браузерах.

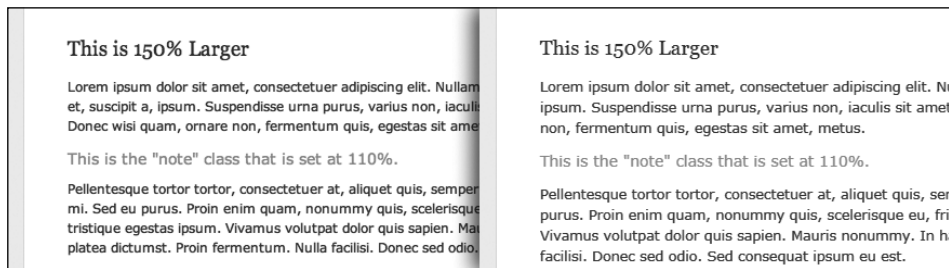


Рис. 1.11. Класс `note`: Safari и IE9/Win при 110%

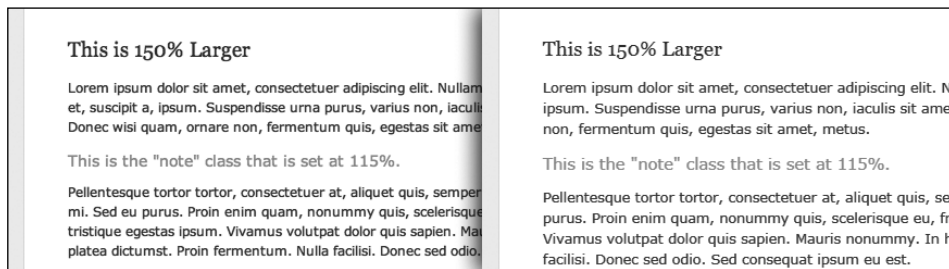


Рис. 1.12. Класс `note`: Safari и IE9/Win при 115%

ЗАДАНИЕ РАЗМЕРОВ С ПОМОЩЬЮ EM

Альтернатива ключевым словам — задание размера текста с помощью `em` — предоставляет те же возможности при работе с IE/Win, что и ключевые слова. Точнее говоря, вы позволяете пользователям изменять размер текста через браузер. `Em` — это относительная единица, позволяющая проводить достаточно

точную настройку текстовых элементов. При работе с единой базовой единицей, заданной по умолчанию, дизайнеры (которые любят использовать пиксели) могут продолжать работать так, как они привыкли.

Давайте вспомним, что `em` — это круглая шпация¹. Роберт Брингхерст, автор книги «Основы стиля в типографике», которая является библией для всех людей, работающих с шрифтами, пишет:

«...круглая шпация — величина переменная. Одна круглая — расстояние, равное кеглю шрифта. В шрифте 6-го кегля одна круглая равна 6 пунктам; для шрифта 12-го кегля она составляет 12 пунктов, для шрифта 60-го кегля — 60 пунктов. Таким образом, пробел в одну круглую пропорционально одинаков при любом кегле шрифта»².

Если вернуться к веб-сайтам, тогда для `font-size`, равного `medium` (в большинстве браузеров это соответствует 16 пикселям), `1em` эквивалентен 16px. Если кегль текста по умолчанию 11px, то `1em` — 11px. Преимущество `em` при задании размеров текста, интерлиньяжа и отступов между элементами заключается в том, что при изменении размера шрифта соответствующим образом изменяются и эти параметры.

Ричард Раттер предлагает хороший способ нормализации базового кегля шрифта с помощью `em` (<http://clagnut.com/blog/348/>), в соответствии с которым размеры всех элементов более или менее привязаны к их пиксельным эквивалентам. Предполагается, что настройки браузера по умолчанию — `medium`, что в большинстве случаев соответствует 16 пикселям.

Способ Ричарда начинается с уменьшения базового кегля шрифта (элемент `<body>`) в свойстве `font-size` до 62,5%:

```
body { font-size: 62.5%; }
```

Эта строка уменьшает размер `medium`, заданный по умолчанию, с 16 до 10 пикселей. Как объясняет Ричард, так следует поступить, чтобы при базовом размере, равном 10 пикселям, вы получили целое значение и могли думать в пикселях, а указывать размеры в `em`.

1 Шпация (в полиграфии) — пробельный материал, применяемый при ручном и монотипном металлическом наборе для отбивки элементов строки друг от друга по горизонтали, а также для создания отступов. Шпация может иметь толщину от 1 пункта до величины кегля шрифта, используемого в строке. В зависимости от толщины шпация называется круглой (кегельной), полукруглой (полукегельной) или тонкой. — *Примеч. перев.*

² Брингхерст Роберт. Основы стиля в типографике. М., 2006.

Например, после того как вы примените 62,5% к элементу `<body>`, 1em примет значение 10px, 1.2em — 12px, .9em — 9px, 1.8em — 18px и т. д. Если задавать разные значения для разных элементов страницы, то настройки могут иметь примерно такой вид:

```
body { font-size: 62.5% }      /* позволяет установить базовое значение,
                               равное 10px */
h1 { font-size: 2em; }        /* 20px */
h2 { font-size: 1.8em; }      /* 18px */
p { font-size: 1.2em; }       /* 12px */
#sidebar { font-size: 1em; }   /* 10px */
```

Этот способ может пригодиться дизайнерам, которые привыкли задавать кегль шрифта в пикселах, но хотят предоставить пользователям возможность изменять размеры текста. Как и с моделью «ключевое слово/процентное соотношение», рассмотренной в этой главе ранее, em требует аккуратности при работе с вложенными структурами (когда вложенность элементов составляет более одного уровня).

Предположим, что нам нужно, чтобы все элементы `<abbr>`, находящиеся в одном абзаце, имели кегль 24px, а не 12px. Для этого мы используем настройку:

```
p abbr { font-size: 2.4em; }
```

Данный код позволяет сделать так, чтобы элементы `<abbr>` предстали в размере 28.8px (так как `<abbr>` наследует от `<p>`, а мы написали, что его кегль должен быть в 2,4 раза больше родительского, который в нашем примере составляет 12px). Элементы `<abbr>` наследуют свойство `font-size` в качестве базового размера. Чтобы изменить это значение, воспользуемся формулой, приведенной в статье Ричарда:

*Размер дочернего объекта в пикселах / размер родительского объекта
в пикселах = размер дочернего объекта в em.*

В нашем примере (где 24px — целевой размер, а 12px — базовое родительское значение) формула будет иметь следующий вид:

$$24 / 12 = 2.$$

Таким образом, для того чтобы в рамках абзаца элементы `<abbr>` предстали в размере 24px, можно написать:

```
p {font-size: 1.2em; }        /* 12px */
p abbr { font-size: 2em; }    /* 24px */
```

При одноуровневым вложениям задача не кажется сложной. Однако ситуация ухудшается с каждым последующим уровнем вложенности. В большинстве случаев вам не понадобится больше одного-двух уровней. Для того чтобы получить полный контроль над ситуацией, не требуются сложные вычисления, но

нужно помнить о том, что если у вас будет несколько уровней вложенности, то сложность расчетов при задании размеров с помощью `em` или ключевого слова/процентного соотношения сильно возрастет.

REM

Не отчаивайтесь, если вам не удалось разобраться с уровнями вложенности элементов `em`. В CSS3 появился новый элемент — `rem` (от англ. «root `em`» — корневой `em`).

В то время как `em` зависит от контекста (пропорционален размерам родительского элемента), `rem` позволяет проигнорировать контекст и вложения, задавая размер текста на базе корневого элемента.

Проиллюстрирую это на примере:

```
body {
    font-size: 62.5%; /* 10px при настройках браузера по умолчанию */
}
h1 {
    font-size: 1.2rem; /* 12px */
}
p {
    font-size: 2.4rem; /* 24px */
}
```

Используя `rem`, мы всегда знаем, как размеры шрифта соотносятся с базовым (корневым) кеглем, независимо от того, в каком месте документа расположен элемент. В то же время `rem` (как и `em`) является относительной единицей и может быть изменен пользователем средствами браузера IE/Win. В каком-то смысле вы гарантируете предсказуемость и обеспечиваете возможность изменения размера текста (при условии использования настроек по умолчанию), как и при использовании `em`. Это отличное пополнение пуленепробиваемого инструментария.

Что касается поддержки `rem` браузерами, необходимо помнить, что `rem` — это новый элемент, хотя его уже поддерживают Safari 5, Chrome, Firefox 3.6+, Opera 12 и IE9. Для более старых версий браузеров Джонатан Снук предлагает решение, которое позволяет использовать `rem` в поддерживающем его браузере, и выполняет пересчет на пиксели, если браузер не поддерживает `rem`.

Решение данной задачи Джонатан Снук изложил в статье «Font sizing with rem» (http://snook.ca/archives/html_and_css/font-size-with-rem):

```
body {
    font-size: 62.5%; /* 10px при настройках браузера по умолчанию */
}
h1 {
    font-size: 12px;
```

```
font-size: 1.2rem; /* 12px */  
}  
p {  
font-size: 24px;  
font-size: 2.4rem; /* 24px */  
}
```

Браузеры, которые не поддерживают `rem`, используют значение, заданное в пикселах, игнорируя значение, заданное `rem`. А те браузеры, которые поддерживают `rem`, будут использовать второе описание, отменяя задание значения в пикселах.

Побочным эффектом будет то, что в IE7 и IE8 кегль текста будет задаваться в пикселах (`rem` не поддерживается), и пользователь не сможет изменить его размер. Вероятность возникновения данной проблемы необходимо предварительно оценить, опираясь на статистику использования браузеров на вашем сайте, и постоянно отслеживать изменение этой информации. Если учесть, что IE9 уже поддерживает `rem`, не забывайте про эту новую возможность.

РЕЗЮМЕ

Из данной главы вы должны сделать как минимум один вывод — пользователь имеет право изменять размеры текста на странице. Я предложил вам несколько стратегий, которые уже доказали свою эффективность: использование ключевых слов и процентов, а также альтернативный вариант — использование `em`. Однако все веб-сайты разные, у каждого сайта есть специфика. Вы должны обеспечить своим проектам максимальную гибкость.

Помните, что:

- пользователи IE/Win не смогут масштабировать текст, если его кегль был задан в пикселах;
- ключевые слова — простой способ задания размеров шрифта, который позволяет пользователям масштабировать текст;
- использование процентов позволяет легко и быстро управлять изменением размеров шрифтов или задавать пользовательские таблицы стилей, при этом понадобится лишь одна команда CSS;
- использование ключевых слов и процентов обеспечивает достаточный уровень точности отображения, если в браузере используются настройки по умолчанию;
- способ, предложенный Ричардом Раттером, упрощает переход от пикселей к относительным размерам шрифтов;

- многоуровневое вложение элементов требует дополнительных расчетов, чтобы обеспечить единообразие отображения;
- элемент `rem`, являющийся частью CSS3, позволяет надеяться, что в будущем задание относительных размеров текста во вложенных элементах станет намного проще.

И хорошая новость: скорее всего, эта глава — наименее интересная часть книги, но она очень важна для формирования надежного (правильнее сказать — гибкого) фундамента для последующих примеров. Далее нам предстоит прийти к выводу, что пользователь должен иметь возможность изменять размеры текста на странице любым доступным ему способом. Учитывая это, мы воспользуемся инструментами CSS и переработаем традиционные элементы дизайна, добиваясь максимальной гибкости. Удачи!

ГЛАВА 2

НАВИГАЦИЯ



Система навигации веб-сайта должна подстраиваться под контент любого размера

Довольно часто система навигации является основой для дизайна веб-сайта. Навигация — такой же важный компонент страницы, как и любой другой. Как правило, разработка системы навигации заключается в том, что веб-дизайнер любовно вырисовывает кнопки, вкладки или текст в графическом редакторе, а затем размещает их во множество вложенных `<table>`. Иногда для создания эффекта при наведении курсора на вкладки используется JavaScript, который заменяет один набор изображений на другой. Конечный результат может выглядеть просто великолепно. Однако то, что скрыто за внешней красотой, как правило, не так привлекательно. Более подробно мы изучим этот вопрос после того, как рассмотрим принятую практику создания систем навигации.

Давайте рассмотрим пример, который хорошо проиллюстрирует типичный подход к разработке графических вкладок. После того как мы разберемся в недостатках данного метода, мы сможем разработать аналогичный (на этот раз пуленепробиваемый!) дизайн, используя экономичную разметку, три небольших изображения и, конечно же, CSS.

ОБЩЕПРИНЯТЫЙ ПОДХОД

В качестве примера я решил использовать основные навигационные вкладки, которые нашел на сайте LanceArmstrong.com (рис. 2.1). Мой выбор обусловлен двумя причинами. Прежде всего, я фанат Лэнса Армстронга. Но, что более важно, дизайн сайта просто кричит, что ему нужна переделка с использованием CSS, которая обеспечила бы необходимую гибкость. Наша цель — использовать CSS с наименьшим количеством кода и создать простой для обслуживания и удобный для масштабирования сайт.

СОВЕТ

Дизайн сайта давно изменен. Поэтому если вы хотите увидеть, как выглядели оригинальные вкладки, просто перейдите по следующей ссылке: <http://web.archive.org/web/20041204124045/http://www.lancearmstrong.com/about.htm>.



Рис. 2.1. Навигационные вкладки со старой версии сайта LanceArmstrong.com в активном и неактивном состояниях

ПОДСВЕТКА

Дизайн вкладок обладает одной интересной особенностью помимо изменения фона и цвета рамки. В каждом из состояний вкладки (активном и неактивном) используется градиентная заливка по вертикали с переходом в сплошную: белую — в активном состоянии и светло-желтую — в неактивном. Верхняя рамка подведена линией более светлого тона, чем остальной абрис. Толщина линии — 1 пиксел. Она добавляет объем и создает эффект, как будто источник света расположен сверху страницы (рис. 2.2).

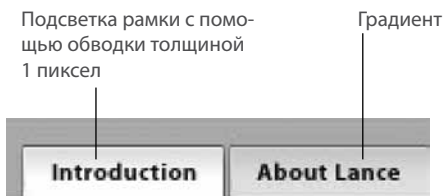


Рис. 2.2. Увеличенное изображение элементов навигации. Обратите внимание на небольшую подсветку рамки в верхней части каждой вкладки. Градиентная заливка используется для имитации объема и создания светотени

Прежде чем перейти к следующему пункту, хочу напомнить, что в целом вкладки реализованы удачно. Даже учитывая, что «хороший дизайн» довольно субъективное понятие (ведь у вас может быть совсем другое мнение на этот счет), все должны согласиться с тем, что дизайнер потратил много времени на разработку системы навигации LanceArmstrong.com, сделав ее привлекательной и функциональной. Цель была достигнута. Именно поэтому я выбрал этот сайт в качестве примера.

На LanceArmstrong.com каждая вкладка представляет собой графический объект. Картинки изображают активное или неактивное состояния, в зависимости от того, какую страницу вы просматриваете. Кроме того, используется графический логотип сайта. На рис. 2.3 показано одно из четырех возможных изображений навигационной системы.

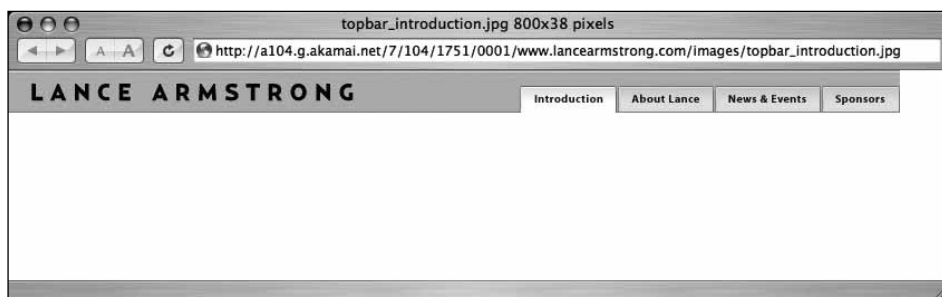


Рис. 2.3. Вся система вкладок и логотип образуют единое большое изображение. На сайте есть еще как минимум три аналогичных изображения с подсветкой других разделов

СТАНДАРТНЫЕ СПОСОБЫ СОЗДАНИЯ ЭФФЕКТОВ

В настоящий момент навигация на LanceArmstrong.com не предусматривает эффектов при наведении курсора. Их можно добавить с помощью JavaScript и второго комплекта изображений, но это потребует еще большего количества строк кода. Мы займемся эффектами для вкладок позднее, когда будем изменять дизайн в разделе «Пуленепробиваемый подход». Но в нашем случае понадобится лишь несколько строк CSS вместо объемного кода, дополнительных изображений и JavaScript.

Итак, мы видим навигационные вкладки, разработанные в соответствии с дизайном сайта. Нажатие на каждую из вкладок перемещает нас на один из главных разделов сайта. Каждой вкладке соответствует текстовая надпись. Теперь давайте рассмотрим систему более скрупулезно и научимся создавать аналогичный дизайн, но другим способом.

Уязвимые места

Что здесь не так? Я не собираюсь критиковать дизайнеров LanceArmstrong.com. На этом сайте используется такой же способ навигации, как и на миллионах других. Я просто рассматриваю его как пример, потому что он интересен с визуальной точки зрения и его довольно легко переделать с использованием CSS и простой разметки. Давайте перечислим особенности сайта, которые являются причиной его уязвимости.

Слишком много кода

Объемный код на JavaScript, использованный для размещения изображений и добавления динамики системе навигации (переключение изображений при наведении курсора), затрудняет работу с самим сайтом. Каждая вкладка представлена изображениями, которые размещаются в серии вложенных таблиц, GIF-разделителями (прозрачными изображениями, используемыми только для заполнения пустого места между элементами) и другими элементами разметки, необходимыми для создания точного макета. В результате генерируется слишком объемный код, который приводит к тому, что сайт становится тяжеловесным и, следовательно, медленно загружается.

Обратимся к LanceArmstrong.com: все четыре вкладки расположены на одном изображении. Для него определена карта ссылок, которая делает каждую вкладку активной. Если бы каждая вкладка была отдельным изображением, размещенным в таблице с GIF-разделителями, потребовалось бы больше кода. Но у каждого из этих способов имеются свои недостатки.

Недоступность сайта для некоторых категорий пользователей

Другим побочным эффектом большого количества кода является то, что он усложняет работу в текстовых браузерах и вспомогательных приложениях, которыми пользуются люди с ограниченными возможностями. Вся система навигации является одним изображением (см. рис. 2.3), но большинство дизайнеров забывают про атрибут `alt`, который необходимо прописывать для каждой зоны ссылки при использовании карты ссылок. Пользователи программ для чтения с экрана или пользователи, отключающие изображения в браузерах для более быстрой загрузки (при медленных соединениях), едва ли смогут комфортно работать с таким сайтом.

Проблема масштабирования

Это очень серьезная проблема. Так как вкладки представлены изображениями, слабовидящие пользователи не смогут изменять размер текста через браузер, чтобы облегчить чтение. Систему навигации, построенную на изображениях, невозможно масштабировать.

ПРИМЕЧАНИЕ

Исключением является функция Масштаб страницы, которую поддерживает большинство современных браузеров. Она изменяет размеры всего документа.

ОТСУТСТВИЕ ГИБКОСТИ

Если в будущем редакторы сайта LanceArmstrong.com захотят изменить текст на вкладке или, например, написать вместо «О Лэнсе» слова «Почему мы любим Лэнса», у них возникнут проблемы. Редактирование, удаление или замена текста на изображениях потребуют создания новой подборки картинок. Это, в свою очередь, приведет к необходимости изменения размерной сетки в карте ссылок. А ведь все это — немалый объем дополнительной работы.

Кроме того, как мы помним, система вкладок представлена одним изображением. Это означает, что для внесения одного изменения потребуется обновление четырех изображений (причем для каждого состояния вкладки) — это потенциальная головная боль для дизайнера.

Как видно, в системе еще многое можно улучшить. Каждый из минусов является достаточной причиной для того, чтобы найти альтернативный способ решения задачи.

ПУЛЕНЕПРОБИВАЕМЫЙ ПОДХОД

Давайте переделаем систему навигации с помощью экономичных методов CSS, учитывая недостатки общепринятой практики создания таких систем, но сохранив понравившийся дизайн. Нашей целью является устранение недостатков с точки зрения пуленепробиваемого подхода.

Как и с любым другим проектом, нужно сначала решить, как сделать разметку более гибкой и удобной. Имеет смысл структурировать список ссылок как обычный список. Используем новый семантический элемент `<nav>` из HTML5 для оформления списка:

```
<nav role="navigation">
  <ul>
    <li id="t-intro"><a href="/">Introduction</a></li>
    <li id="t-about"><a href="about.html">About Lance </a></li>
    <li id="t-news"><a href="news.html">News & Events</a></li>
    <li id="t-sponsors"><a href="sponsors.html">Sponsors</a></li>
  </ul>
</nav>
```

СОВЕТ

Для получения дополнительной информации о значении меток см. стандарт www.w3.org/WAI/PF/aria-practices/#kbd_layout (на английском языке).

Обратите внимание на то, что мы также используем атрибут `role` элемента `<nav>` для введения в систему навигации WAI-ARIA-метки. Роли метки обеспечивают навигацию для вспомогательных приложений. Иными словами, роли значительно повышают доступность сайта путем добавления дополнительной семантики.

ПРИМЕЧАНИЕ

WAI-ARIA (Web Accessibility Initiative—Accessible Rich Internet Applications) — стандарт доступности активных интернет-приложений, определяет подходы к тому, чтобы сделать содержимое сайтов и интернет-приложения более доступными для людей с ограниченными возможностями. Стандарт представлен проектом технической спецификации, опубликованным консорциумом W3C.

Простой маркированный список, который вкладывается в `<nav>`, — единственное, что нам понадобится для создания пуленепробиваемой таблицы в HTML5. Элементарная разметка гарантирует работу навигационной системы во всех браузерах, устройствах и даже вспомогательных приложениях. И это уже не говоря о том, что страница станет существенно «легче». Сравните количество кода, использованного в нашем случае, с общепринятым подходом на базе вложенных таблиц. Разница очевидна.

Обратите внимание на то, что каждому пункту списка присвоен идентификатор. Идентификаторы понадобятся нам позднее, когда нужно будет указать, на какой странице сайта мы находимся, чтобы сделать вкладку активной.

УПРОЩЕНИЕ

Как правило, маркированный список без стилового оформления выглядит как простой список с маркерами в левой части строки (рис. 2.4). Любое устройство или текстовый браузер, не поддерживающие CSS, смогут воспроизвести его.

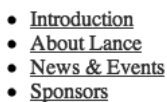
- 
- [Introduction](#)
 - [About Lance](#)
 - [News & Events](#)
 - [Sponsors](#)

Рис. 2.4. Маркированный список без стилового оформления

ДВА НЕБОЛЬШИХ ИЗОБРАЖЕНИЯ

Вы еще помните, что в рамках общепринятого подхода для каждого комплекта вкладок используются отдельные изображения? Фактически нам понадобятся четыре изображения: по одному для каждой зоны сайта, в которой активируется конкретная вкладка. Для того чтобы упростить процесс, предлагаю использовать только пару небольших изображений, которые мы разложим и скопируем по горизонтали на каждой из вкладок. Так как они будут находиться рядом, нам не нужно учитывать ширину или высоту каждого из элементов. К этому вопросу мы вернемся позднее в разделе «Пуленепробиваемый подход».

Если снова посмотреть на исходные вкладки, видно, что вертикальный градиент начинается в верхней части и постепенно переходит в сплошную заливку. Используем Adobe® Photoshop® для создания двух изображений (одно для активных состояний, второе — для неактивных), которые будут воспроизводить градиент, переходящий в прозрачный фон (рис. 2.5). Мы решили заменить цвет в нижней части каждого изображения на прозрачный, а позднее залить прозрачную часть с помощью CSS.

СОВЕТ

Еще одно преимущество списка заключается в следующем: большинство онлайн-приложений используют информацию о количестве элементов в списке. Это позволяет подсказать пользователю, что находится дальше. Например, пользователи популярной программы JAWS от Freedom Scientific могут пропустить список или перейти к следующему элементу на странице (более подробную информацию о навигации в JAWS см. по ссылке www.freedomscientific.com/fs_products/Surfs_Up/Navigating.htm).



Рис. 2.5. Два увеличенных изображения, которые мы будем использовать для создания градиента. Цвет в нижней части каждого изображения заменен на прозрачный (участок с шахматной заливкой), который мы с помощью CSS позднее заполним цветом

Обратите внимание на то, что каждое изображение имеет ширину 10 пикселей и включает 1-пиксельную линию по верху рамки для подсветки вкладки. После нее начинается градиент, переходящий в прозрачный фон.

ПРИМЕНЕНИЕ СТИЛЯ

Теперь с помощью CSS нам нужно свести вместе компоненты системы навигации. Первый этап — определить правила CSS, которые позволят разместить элементы навигации по горизонтали.

Так как роли меток WAI-ARIA используются в документе лишь один раз, воспользуемся ими для создания стиля вкладок с помощью селектора атрибутов, не присваивая идентификаторы или классы (еще одна причина полюбить метки!). Только для элемента `<nav>` можно определить роль маркера `navigation`, для которого зададим следующий стиль:

```
nav[role="navigation"] {  
    margin: 0;  
    padding: 10px 0 0 46px;  
    list-style: none;  
    background: #FFCB2D;  
}  
nav[role="navigation"] li {  
    float: left;  
    margin: 0 1px 0 0;  
    padding: 0;  
    font-family: "Lucida Grande", sans-serif;  
    font-size: 80%;  
}
```

СОВЕТ

Более подробная информация по заданию стилей с помощью атрибутов роли метки приведена в статье Джереми Кита по адресу <http://addactio.com/journal/4267/>.

Не забывайте, что для данных примеров понадобится HTML5 JS и переустановка таблицы стилей. Более подробную информацию можно найти во введении книги.

Используем свойство `float` для горизонтального размещения элементов. Параллельно объявим желтый цвет фона, поверх которого будут располагаться сами вкладки. Желтый фон очень важен, так как он является частью масштабируемой системы навигации (желтый контейнер и вкладки). Кроме того, мы сбросили свойства отступов и заполнения. При этом мы задали 1-пиксельный отступ справа (для разделения вкладок), указали шрифт и установили `font-size` равным 80% от значения по умолчанию. Используя знания, полученные в главе 1 «Гибкое управление текстом», для задания базового размера на странице мы воспользовались ключевым словом `small` в элементе `<body>`. Теперь можно спать спокойно: пользователи IE/Win смогут масштабировать текст на вкладках.

На рис. 2.6. показано, как выглядит наша система навигации в данный момент.



Рис. 2.6. А выглядит система навигации довольно неаккуратно

Понимаю, что она совсем не похожа на то, что вам хотелось бы увидеть. Но до результата осталось лишь несколько строчек кода.

ИСПОЛЬЗУЕМ ПЛАВАЮЩИЕ ЭЛЕМЕНТЫ ДЛЯ ФИКСАЦИИ

Первая проблема, которую необходимо решить: так как элементы `` стали плавающими, они не имеют постоянного положения и не заполняют внешний тег ``, задающий фоновый цвет. Иными словами, так как внутренние элементы становятся подвижными, внешний тег `` не знает свою высоту.

Для решения этой проблемы сделаем `` плавающим, как остальные элементы. Это позволит `` заполнить пространство, растягивая желтый фон. Так как элементы становятся плавающими, их содержимое, условно говоря, «упаковывается» (становится той ширины и высоты, которая необходима в конкретный момент времени). Определим ширину для ``, предполагая (в данном примере), что тег находится внутри макета с фиксированной шириной такого же размера (рис. 2.7).



Рис. 2.7. Делая элемент `` плавающим, мы растягиваем фон позади него

ПРИМЕЧАНИЕ

В теге `` мы указали ширину 720px, а слева добавили отступ 46px для разделения вкладок. Так как отступ является составной частью элемента, общая ширина навигационной панели составит 766px.

СОВЕТ

Так как `ul` превращается в плавающий элемент, необходимо убедиться в том, что свойства последующих элементов сброшены. Каждый последующий элемент, например горизонтальный элемент или блок контента, должен иметь свойство `clear: left`. Это необходимо для того, чтобы элемент оказался под навигационной панелью.

```

nav[role="navigation"] ul {
    float: left;
    width: 720px;
    margin: 0;
    padding: 10px 0 0 46px;
    list-style: none;
    background: #FFCB2D;
}
nav[role="navigation"] ul li {
    float: left;
    margin: 0 1px 0 0;
    padding: 0;
    font-family: "Lucida Grande", sans-serif;
    font-size: 80%;
}

```

Результат стал несколько лучше. Сейчас нужно сделать так, чтобы размер желтого фона изменялся вместе с его наполнением, защищая элементы оформления. Это также означает, что нам не нужно использовать для списка свойство `height` (чего мы и добивались), что делает наш сайт более гибким.

ВКЛАДКИ ОБРЕТАЮТ ФОРМУ

Придадим форму ссылкам, определив отступ, границы и фоновый цвет:

```

nav[role="navigation"] ul li a {
    float: left;
    display: block;
    margin: 0;
    padding: 4px 8px;
    color: #333;
    text-decoration: none;
    border: 1px solid #9B8748;
    border-bottom: none;
    background: #F9E9A9;
}

```

Для элементов `<a>` мы написали правило `display: block;`, превращающее вкладку в элемент, активируемый по щелчку мыши. На уровне блока элементы размещаются на отдельных строках, поэтому снова используем `float` для того, чтобы расположить элементы по горизонтали.

После каждой ссылки мы добавили отступы, изменили цвет и стиль текста, а также добавили границы со всех сторон, кроме нижней. Не забудьте про поля вокруг `<a>`, чтобы сделать активной всю вкладку, а не только текст ссылки.

Удивительно, но после добавления нескольких правил CSS вкладки начинают обретать форму (рис. 2.8).

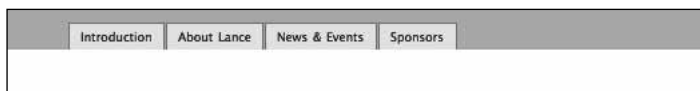


Рис. 2.8. Добавив отступ к элементу `<a>`, мы сделали активной всю кнопку. Теперь навигация по сайту стала еще проще

ВЫРАВНИВАНИЕ ФОНОВЫХ ИЗОБРАЖЕНИЙ

На следующем шаге для каждой вкладки мы используем созданные ранее изображения как фоновые:

```
nav[role="navigation"] ul li a {
    float: left;
    display: block;
    margin: 0;
    padding: 4px 8px;
    color: #333;
    text-decoration: none;
    border: 1px solid #9B8748;
    border-bottom: none;
    background: #F9E9A9 url(img/off_bg.gif) repeat-x top left;
}
```

Мы указали цвет и изображение в одном правиле — теперь цвет (`#F9E9A9`) будет просвечивать сквозь прозрачные фрагменты изображения. Указанный нами цвет — это тот же самый оттенок с нижней части изображения, который мы удалили в графическом редакторе.

Кроме того, мы выровняли изображения по левому верхнему углу вкладки, а свойство `repeat-x` позволило размножить их по горизонтали (рис. 2.9). На тех участках, где изображение отсутствует (или прозрачно), используется фоновый цвет.

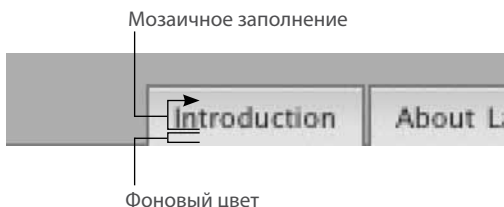


Рис. 2.9. Фоновые изображения размещены по горизонтали для создания подсветки и градиента. Так как они находятся рядом, значения ширины и (или) высоты не повлияют на дизайн

Осталось лишь добавить правило, задающее действия при наведении курсора и изменяюще вкладки в активном состоянии. Но сначала давайте определим

границу под вкладками, которая исчезает, когда вкладка становится активной. Это создаст иллюзию, что вкладка является единым объектом со страницей, расположенной под ней.

Самый простой и удачный способ работы с границами — повторение небольшого изображения с высотой, равной высоте границы. Здесь мы будем использовать третье изображение, оно настолько крошечное, что не потребует дополнительных ресурсов (рис. 2.10).

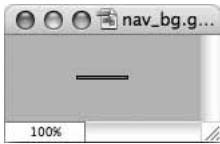


Рис. 2.10. Это изображение с высотой 1 пиксел используется как нижняя граница вкладок и позволяет активной вкладке перекрывать его. Таким образом, создается иллюзия, будто нижняя часть вкладки объединяется с белой страницей

Изображение представлено в формате GIF, а его размер — 1×37 пикселей. Картинка будет иметь горизонтальную ориентацию и, как плитка, укладываться вдоль нижнего края каждой из четырех вкладок. В данном случае ширина изображения была выбрана произвольно, так как мы делаем горизонтальную границу. Я уже не могу вспомнить причину, почему взял именно 37 пикселей.

На рис. 2.11 показан порядок размещения элементов навигации с фоновым цветом, 1-пиксельной границей и контентом вкладок.

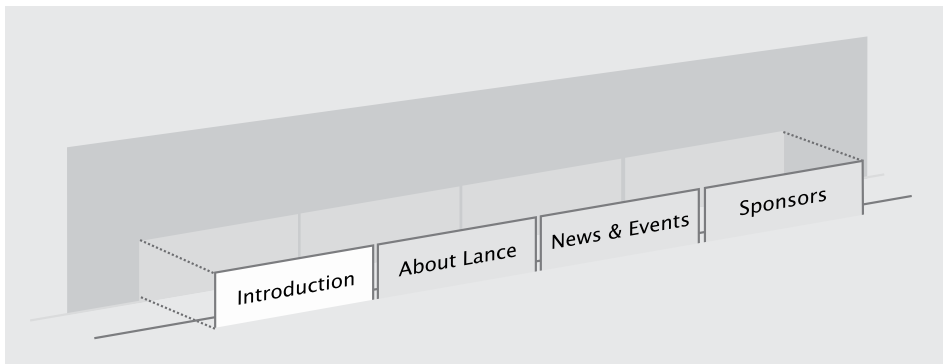


Рис. 2.11. Объемное изображение, демонстрирующее использование фонового изображения для создания нижней границы системы навигации

СОВЕТ

Internet Explorer иногда не слишком уверенно работает с небольшими повторяющимися картинками. Решением проблемы может стать использование фоновых изображений.

ДОБАВЛЯЕМ НИЖНЮЮ ГРАНИЦУ

Как и в прошлый раз, в объявлении `` используем следующий переход к фоновому изображению:

```
nav[role="navigation"] ul {
    float: left;
    width: 720px;
    margin: 0;
    padding: 10px 0 0 46px;
    list-style: none;
    background: #ffcb2d url(img/nav_bg.gif) repeat-x bottomleft;
}
```

В этом коде мы указали, что изображение должно повторяться горизонтально в нижней части. Так как изображение имеет высоту 1 пиксел, желтый фоновый цвет, который мы также здесь задали (`#ffcb2d`), будет просвечивать через остальные части навигационной панели (рис. 2.12).

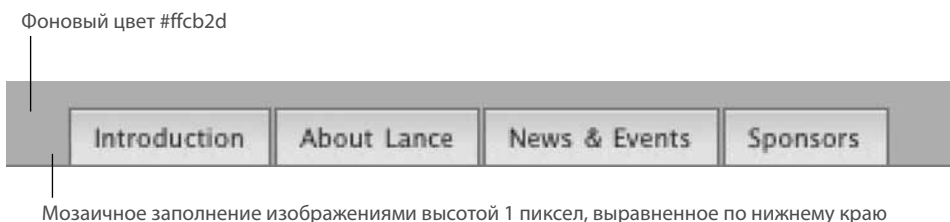


Рис. 2.12. Мы использовали мозаичное заполнение фона для создания нижней границы и приблизились к искомому результату

СМЕНА ИЗОБРАЖЕНИЙ ПРИ НАВЕДЕНИИ КУРСОРА

Давайте займемся активной вкладкой и созданием эффекта при наведении курсора, мы повторим серый фон активной вкладки, использованный на LanceArmstrong.com, заменив бежевое фоновое изображение, выбранное по умолчанию. Для создания обоих сценариев нам потребуется слегка подправить код. Начнем с оформления вкладок при наведении курсора:

```
nav[role="navigation"] ul li a:hover {
    color: #333;
    padding-bottom: 5px;
```

```
border-color: #727377;
background: #FFF url(img/on_bg.gif) repeat-x top left;
}
```

Обратите внимание, что текст и цвет границ мы сделали более темными и использовали серый фон, переходящий в сплошной белый (#FFF). Мы также увеличили нижний отступ на 1 пиксел (с 4 до 5 пикселей). Дополнительный пиксел позволит перекрыть границу фона (высота границы 1 пиксел) вкладкой, создав иллюзию единого пространства (рис. 2.13).

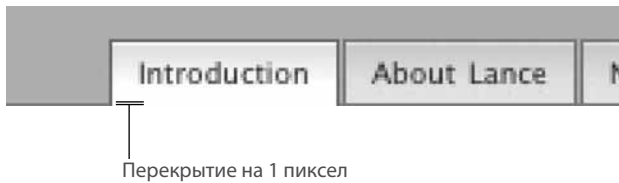


Рис. 2.13. Увеличивая нижний отступ с 4 до 5 пикселей на вкладке, на которой находится курсор, мы закрываем нижнюю границу. Так создается иллюзия, что вкладка является частью страницы

АКТИВНОЕ СОСТОЯНИЕ

Как вкладка узнает, что состояние изменилось? Для того чтобы применить к ней этот же стиль, воспользуемся нисходящим селектором. Мы добавим действие, которое будет выполняться, когда пользователь наводит курсор на вкладку.

```
nav[role="navigation"] a:hover, body#intro #t-intro a
```

Нисходящий селектор позволяет указывать конкретные элементы, основанные на родительских. Перечисляя их через пробел, мы сужаем диапазон целевых элементов, упорядочивая их в соответствии с порядком появления в древовидной структуре документа. Более подробную информацию по селекторам см. www.w2.org/TR/REC-CSS2/selector.html.

Напомним, что мы добавили свойство `id` для элемента `<body>` страницы, присвоив ему значение `intro`:

```
<body id="intro">
```

В этом случае код отдает команду: «на страницах с идентификатором основного текста `#intro` для данной ссылки следует использовать заменяющий фон, более темные цвета и т. д.». Эта возможность CSS позволяет выдавать сообщения «Вы находитесь здесь», работающие в большинстве систем навигации. В данном случае мы объединили состояния изменения вкладки, описав как действие при наведении кнопки мыши, так и активацию вкладки в рамках одного фрагмента кода CSS.

ПРЕИМУЩЕСТВА ПУЛЕНЕПРОБИВАЕМОГО ПОДХОДА

Мы с вами взяли симпатичный дизайн навигационной панели, разрушили его до основания и создали аналогичную систему, но с меньшим количеством кода. Новый код легко обновлять, он работает в большинстве браузеров, устройств и вспомогательных приложений. Кроме того, он стал гибким, так как не зависит от размера и объема размещенного текста.

Мы выровняли фоновые изображения и задали цвета таким образом, что все вкладки стали масштабируемыми. Если пользователь захочет увеличить размер текста в несколько раз для большей удобочитаемости, оформление так же подстроится под новый масштаб (рис. 2.14).



Рис. 2.14. Масштаб этой панели можно увеличивать или уменьшать путем изменения размера текста в браузере

За несколько минут мы сможем изменить текст, добавить или удалить временные вкладки. Для этого нужно лишь отредактировать маркированный список, который был использован для структурирования элементов навигации (рис. 2.15). Просто измените текст в любом пункте списка, описывающего наше меню.

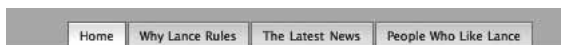


Рис. 2.15. Для изменения текста в каждой вкладке путем обновления простого маркированного списка достаточно нескольких секунд

```
<nav role="navigation">
  <ul>
    <li id="t-intro"><a href="/">Home</a></li>
    <li id="t-about"><a href="about.html">Why Lance Rules</a></li>
    <li id="t-news"><a href="vnews.html">The Latest News</a></li>
    <li id="t-sponsors"><a href="sponsors.html">People Who Like
      ▶ Lance</a></li>
  </ul>
</nav>
```

КАК СОЗДАТЬ ГРАДИЕНТ В CSS3 БЕЗ ИСПОЛЬЗОВАНИЯ ИЗОБРАЖЕНИЙ

До этого момента для создания градиентного фона мы использовали фоновые изображения. Альтернативное решение — использовать градиенты CSS3, но это подходит только для тех браузеров, которые поддерживают эту функцию.

Градиенты CSS3 идеально подходят для ситуаций, когда в таблицах стилей используются изображения с градиентной заливкой. В настоящий момент практически все последние версии браузеров поддерживают использование градиентов. Но мы можем создать резервный сценарий для старых версий браузеров, только вместо градиента будет отображаться сплошная заливка, которой в большинстве случаев более чем достаточно. CSS3 идеально подходит для создания не особо важных, но приятных глазу визуальных элементов.

Написание кода, создающего градиент, — довольно сложная задача. Функции, определяющие градиенты, постоянно менялись на протяжении многих лет, а следовательно, в каждом браузере менялся синтаксис. Настоятельно рекомендую воспользоваться онлайн-инструментом WYSIWYG-типа, в котором уже есть готовые конструкции. Воспроизвести их по памяти практически нереально.

Для этой цели подойдет, например, ColorZilla Ultimate CSS Gradient Generator (www.colorzilla.com/gradient-editor) — удобное приложение, позволяющее создавать градиенты. Программа автоматически генерирует CSS-код для каждого типа браузера (в том числе для IE9 и более старых версий). Каждое правило сопровождается комментариями, подсказывающими, к какому браузеру/ОС данное правило относится (рис. 2.16).

Мы будем использовать цвета из изображений с градиентной заливкой для активного и неактивного состояний:

```
nav[role="navigation"] ul li a {
    float: left;
    display: block;
    margin: 0;
    padding: 4px 8px;
    color: #333;
    text-decoration: none;
    border: 1px solid #9B8748;
    border-bottom: none;
    background: rgb(231,212,154); /* Старые браузеры */
    background: -moz-linear-gradient(top, rgba(231,212,154,1) 0%,
    ▶ rgba(255,230,169,1) 100%); /* FF3.6+ */
    background: -webkit-gradient(linear, left top, left bottom,
    ▶ color-stop(0%,rgba(231,212,154,1)),
```





```

    ▶ color-stop (100%,rgba(255,230,169,1))); /* Chrome,Safari4+ */
background: -webkit-linear-gradient(top, rgba(231,212,154,1) 0%,
    ▶ rgba(255,230,169,1) 100%); /* Chrome10+,Safari5.1+ */
background: -o-linear-gradient(top, rgba(231,212,154,1) 0%,
    ▶ rgba(255,230,169,1) 100%); /* Opera11.10+ */
background: -ms-linear-gradient(top, rgba(231,212,154,1) 0%,
    ▶ rgba(255,230,169,1) 100%); /* IE10+ */
filter: progid:DXImageTransform.Microsoft.gradient
    ▶ ( startColorstr='#e7d49a',
    ▶ endColorstr='#ffe6a9', GradientType=0 ); /* IE6-9 */
background: linear-gradient(top, rgba(231,212,154,1) 0%,
    ▶ rgba(255,230,169,1) 100%); /* W3C */
}
nav[role="navigation"] ul li a:hover, body#intro #t-intro a {
    color: #333;
    padding-bottom: 5px;
    border-color: #727377;
    background: rgb(224,226,238); /* Старые браузеры */
    background: -moz-linear-gradient(top, rgba(224,226,238,1) 0%,
    ▶ rgba(255,255,255,1) 100%); /* FF3.6+ */
    background: -webkit-gradient(linear, left top, left bottom,
    ▶ color-stop(0%,rgba(224,226,238,1)),
    ▶ color-stop (100%,rgba(255,255,255,1))); /* Chrome,Safari4+ */
    background: -webkit-linear-gradient(top, rgba(224,226,238,1) 0%,
    ▶ rgba(255,255,255,1) 100%); /* Chrome10+,Safari5.1+ */
    background: -o-linear-gradient(top, rgba(224,226,238,1) 0%,
    ▶ rgba(255,255,255,1) 100%); /* Opera11.10+ */
    background: -ms-linear-gradient(top, rgba(224,226,238,1) 0%,
    ▶ rgba(255,255,255,1) 100%); /* IE10+ */
    filter: progid:DXImageTransform.Microsoft.gradient
    ▶ ( startColorstr='#e0e2ee', endColorstr='#ffffff',
    ▶ GradientType=0 ); /* IE6-9 */
    background: linear-gradient(top, rgba(224,226,238,1) 0%,
    ▶ rgba(255,255,255,1) 100%); /* W3C */
}

```

ПРИМЕЧАНИЕ

Сплошной фоновый цвет задается до определения правил создания градиента. Более старые версии браузера, которые не воспринимают градиент CSS3, будут отображать вкладку с заливкой основного цвета. Это необходимая мера.

Не бойтесь объемного кода. Обратите внимание на то, что для каждого правила используется комментарий, указывающий на то, к какому браузеру относится

градиент. Префиксы разработчиков позволяют браузерам игнорировать код, который они не поддерживают. Префиксы (-webkit-, -moz- и т. п.) являются важной частью развития стандарта CSS, так как обеспечивают реализацию и проверку некоторых сценариев в реальных ситуациях. Относитесь к этой особенности как к неотъемлемой части рабочего процесса.

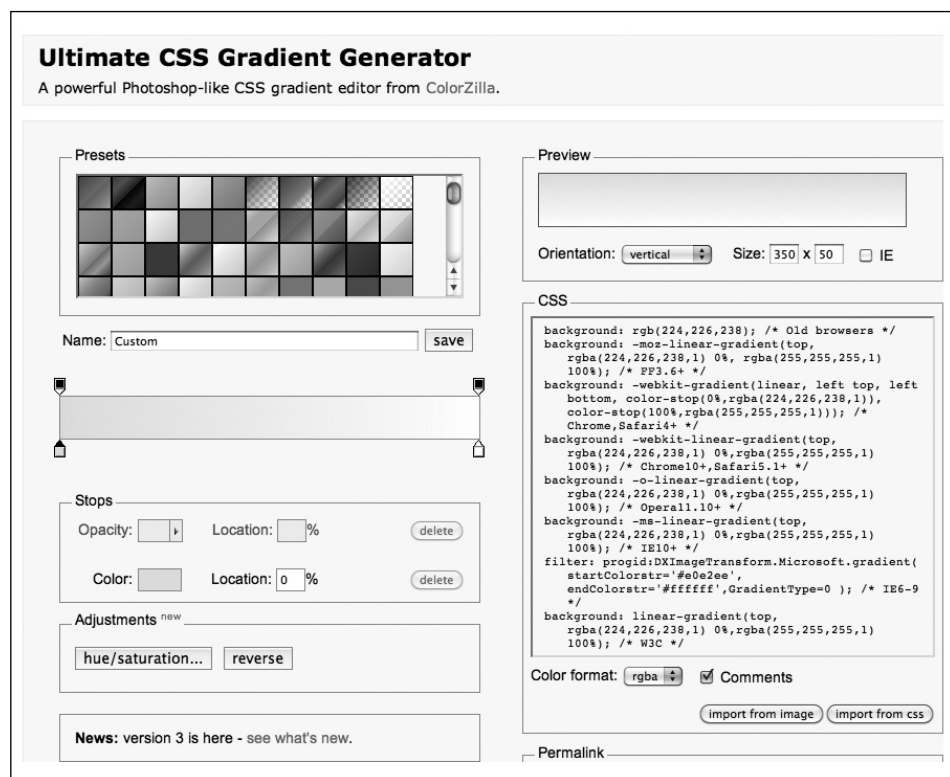


Рис. 2.16. Мастер создания градиента ColorZilla генерирует код для каждого браузера



Рис. 2.17. Увеличенные вкладки с градиентом CSS3, созданным без использования графических изображений

ИСПОЛЬЗОВАНИЕ EM

Для воссоздания вкладок с сайта Лэнса Армстронга мы задавали отступы со всех сторон ссылки в пикселах. В какой-то мере это связано с тем, что нам было необходимо увеличить нижний отступ на один пиксел, обеспечивая перекрытие границ в нижней части панели навигации. Такой подход позволял нам без труда сделать отступ ровно на один пиксел.

Если бы перед нами не стояла задача указать 1-пиксельную границу, мы могли бы использовать `em`, задавая отступы относительно текста ссылки в каждой вкладке. Почему именно `em`? Вспомните первую главу, `em` является величиной, которая зависит от текущего размера шрифта. Таким образом, задавая отступ для каждой ссылки в `em` вместо пикселей, мы сможем сделать так, чтобы масштаб всей вкладки (а не только текста) изменялся в соответствии с размерами шрифта, когда пользователь масштабирует текст.

На рис. 2.18 показана упрощенная версия вкладок — мы удалили границу в нижней части панели навигации и изменили цвет фона на серый.



Рис. 2.18. Упрощенная версия системы навигации без нижней границы

```
nav[role="navigation"] ul {
    float: left;
    width: 50em;
    margin: 0;
    padding: 1em 0 0 5em;
    list-style: none;
    background: #666;
}
```

В этот раз мы указали ширину `50em`, предполагая, что элемент будет находиться в пространстве такого же размера (более подробно о разработке макетов, привязанных к `em`, читайте в главе 8 «Резиновые и эластичные макеты»). Отступы вокруг вкладок также заданы в `em`: `1em` сверху и `5em` слева.

Мы также изменим код `nav[role="navigation"] ul li a`, который определяет стиль каждого элемента навигации, чтобы использовать `em` для задания отступов, следующим образом (также изменим цвет фона на светло-серый и уберем границу возле каждой вкладки):

```
nav[role="navigation"] ul li a {
    float: left;
    display: block;
```

```
margin: 0;
padding: .5em 1em;
color: #333;
text-decoration: none;
background: #ccc;
}
```

Обратите внимание на строку `padding: .5em 1em`. Она означает, что «отступы сверху и снизу должны составлять 0,5 em, а слева и справа — 1 em». Теперь, если изменить размер шрифта, вы увидите, что текст на ссылках и пространство вокруг него изменяются пропорционально (рис. 2.19). Удачно получилось! Вместо того чтобы отводить постоянное количество пикселей, никак не связанное с текущим кеглем шрифта, можно использовать `em` и гарантировать масштабируемость всей системы при увеличении или уменьшении текста.

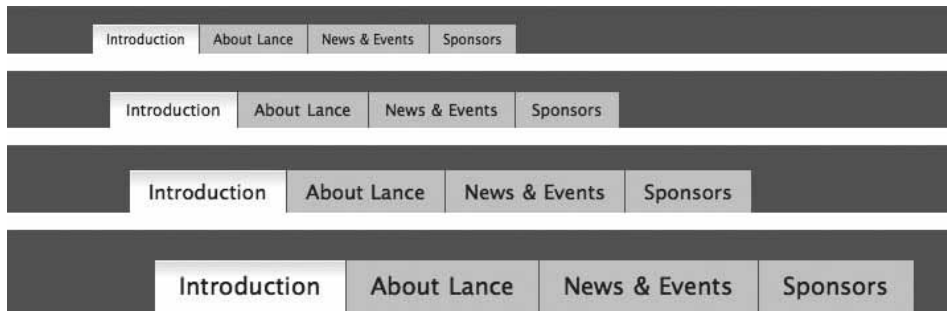


Рис. 2.19. Изменение системы навигации с помощью `em`. Размер текста изменен, чтобы продемонстрировать постоянство пропорций

Я привел этот пример в основном для того, чтобы вы задумались об использовании `em` вместо пикселей при задании отступов, высоты линий и т. п. Конечно, не всегда удастся решить задачу так элегантно (как, например, на вкладках сайта Лэнса Армстронга), однако использование `em` наряду с другими единицами измерения даст вам преимущества, обеспечивая пропорциональность дизайна независимо от размера текста.

ПРИМЕЧАНИЕ

В следующем разделе вы познакомитесь с методами CSS3, которые позволяют создавать элементы со скругленными углами. Новые методы в большинстве случаев помогут обойтись без большого количества графических объектов. Я считаю, что нужно использовать как можно больше гибких решений — тогда вы всегда будете готовы к реализации нетривиальных задач.

Дополнительные примеры

Для получения более полной картины в отношении представленных концепций предлагаю дополнительные примеры — разные варианты дизайнов. В примерах реализованы вкладки, аналогичные нашим, которые обеспечивают гибкость системы навигации.

MOZILLA.ORG

www.mozilla.org

Новый дизайн Mozilla.org использует вкладки со скругленными углами. Их размер изменяется вместе с масштабом страницы или текстового контента (рис. 2.20).

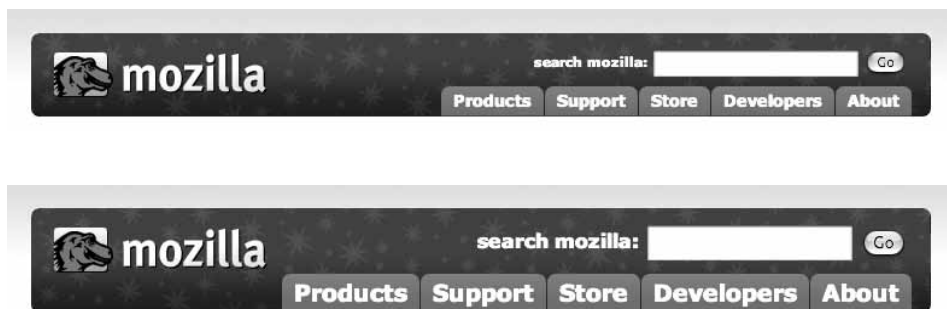


Рис. 2.20. Скругленные углы (подобные представленным на сайте Mozilla.org) создаются способом «раздвижные двери»

Команда Mozilla использовала способ Дугласа Боумана, который он назвал «раздвижными дверями CSS» (www.alistapart.com/articles/slidingdoors/). Боуман предложил интересный вариант использования двух фоновых изображений, раздвигающихся по мере того как увеличивается объем контента между ними. Пара изображений с закругленными углами и другие аналогичные элементы помогут отобразить масштабируемые вкладки и другие контейнеры, сохраняя гибкость и удобство.

Наклонные черты

www.simplebits.com/bits/bulletproof_slants.html

При работе над одним из проектов у меня возникла необходимость разработать гибкие элементы навигации, разделенные наклонными чертами. В результате

я использовал одно фоновое изображение и управлял его видимой областью, показывая большую (или меньшую) часть изображения, в зависимости от размера шрифта текста, находящегося внутри (рис. 2.21).

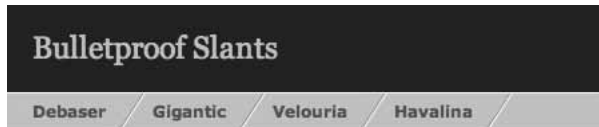


Рис. 2.21. При использовании фонового изображения с наклонной чертой, размер которого больше, чем необходимо, можно добиться масштабируемости при любом изменении размера текста, расположенного между чертами



Рис. 2.22. Увеличенное изображение наклонной черты. Видно, что размер изображения больше, чем необходимо

ESPN.com

www.espn.com

Старый дизайн сайта ESPN.com использовал вкладки для переключения фильтров, реализованные способом, аналогичным тому, который мы изучили в данной главе. В качестве фона использовалось более крупное изображение с градиентной заливкой. Дополнительные части изображения проявлялись, когда пользователь укрупнял текст. На рис. 2.23 можно увидеть, как меняются вкладки при увеличении текста.



Рис. 2.23. В предыдущей версии сайта ESPN Search вкладки увеличивались вместе с изменением размеров текста

Давайте начнем с неактивного состояния вкладок. На вкладке предусмотрена небольшая подсветка размером в 1 пиксел слева и сверху фонового изображения. Поскольку подсветка используется только с двух сторон, можно взять одно фоновое изображение, сделав его достаточно большим для того, чтобы оно могло вместить текст большого объема.

СОВЕТ

Используя описанный ранее инструмент градиента CSS3, можно реализовать такое оформление вкладок без использования изображений. Попробуйте выполнить это задание самостоятельно.

На рис. 2.24 показан внешний вид вкладки при использовании размера текста по умолчанию. Обратите внимание, что виден только левый верхний фрагмент фонового изображения, тогда как на рис. 2.25 при увеличении размера текста становится видна большая часть фонового изображения. Разместив изображение в левой верхней части элемента `<a>` вкладки, мы можем создать эффект подсветки и градиент независимо от объема и кегля текста.

При проектировании этих вкладок необходимо учитывать, что ширина текстового блока может изменяться, так как в каждой надписи отображается результат (число в скобках). Для этого нам понадобится пространство, позволяющее «дышать», т. е. расширяться или сжиматься по необходимости, обеспечивая целостность дизайна.



Рис. 2.24. Вкладка при размере текста по умолчанию. Видна только верхняя левая часть фонового изображения



Рис. 2.25. При увеличении текста становится видна большая часть фонового изображения, а подсветка слева и сверху остается

РЕЗЮМЕ

Пристальный взгляд на наиболее популярный метод создания графических вкладок выявил его недостатки. Мы поняли, в чем преимущества простого маркированного списка, и разобрались, как с помощью CSS позволить элементам навигации не только менять свой масштаб при изменении размера текста, но и размещать в них стабильный контент.

При разработке дизайна системы навигации помните:

- навигация на базе изображений не позволит пользователям, которые испытывают проблемы со зрением, изменять размер текста, более того, данный подход мешает индексации сайта в поисковиках;
- простая разметка (маркированный список в элементе `<nav>`) делает сайт доступным для более широкого диапазона браузеров, устройств и приложений;
- текстовые блоки в навигационной системе легко обновляются и редактируются, а при внесении изменений вам не придется каждый раз создавать новые изображения;

- нестандартное размещение фоновых изображений добавляет изюминку дизайну системы навигации и в то же время обеспечивает ее гибкость, используйте градиенты CSS3, чтобы отказаться от не особо важных изображений;
- `em` позволяет задавать отступы, интерлиньяж и т. п., что облегчает масштабирование всей страницы (а не только шрифта) при любом размере текста;
- в ряде случаев невозможно отказаться от изображений — например, при использовании фирменных шрифтов или из-за ограниченного экранного пространства. В любом случае, это еще не конец света.

Обратите внимание на последний пункт. В реальном мире требования к шрифтам и (или) ширине и высоте элементов навигации могут исходить от заказчика. В таком случае бегите от него как можно дальше! Шучу-шучу, ведь теперь вы можете предложить ему более гибкую систему. Не существует неправильных решений, просто что-то подходит лучше, а что-то — хуже. Необходимо учитывать конкретные условия.

Ну а теперь, после того как мы разработали гибкую концепцию системы навигации, перейдем к другим компонентам сайта.

ГЛАВА 3

Эластичные строки



Старайтесь не задавать точную высоту — масштабирование горизонтальных элементов страницы может повлиять на их размеры по вертикали

Горизонтальные элементы — заголовки, поля регистрации, строки навигации и поиска — стандартные элементы любой страницы веб-сайта. Обычно их размещают в верхней части страницы. Они могут содержать как графику (фоновую или иную), так и текст. Как правило, при разработке данных элементов предполагается, что увеличение их размера по вертикали недопустимо. Разработчики считают, что более крупный шрифт или большее количество контента либо невозможны, либо повлияют на дизайн. В то же время разделы со статьями или большим количеством текста позволяют размещать контент любого размера, поэтому важно (и это вполне реально) предоставить такие же возможности и другим горизонтальным элементам.

В этой главе мы проанализируем общепринятый подход к дизайну полей регистрации и рекламы, которые занимают верхнюю часть веб-страницы. Мы возьмем конкретный пример, проанализируем его, а затем переделаем так, чтобы горизонтальные элементы вмещали текст или контент любого размера.

ОБЩЕПРИНЯТЫЙ ПОДХОД

Для того чтобы понять, что подразумевается под изменением вертикальных размеров горизонтальных компонентов, давайте рассмотрим пример — сайт магазина, который носит гордое название «Самый лучший магазин» (на самом деле такого магазина не существует). «Самый лучший магазин» — обычный коммерческий сайт, предлагающий покупателям товары для дома. Мы придумали его для данной главы, однако дизайн его сайта основан на методах, которые используются в реальности. Для демонстрации общепринятого подхода давайте внимательно посмотрим на фрагмент страницы сайта «Самого лучшего магазина» (рис. 3.1). В верхней части страницы расположена цветная полоса со строкой логина и адреса ближайших магазинов. Под ней еще одна полоса, с текстом обновляемого рекламного сообщения. Каждая полоса содержит одну строку текста.

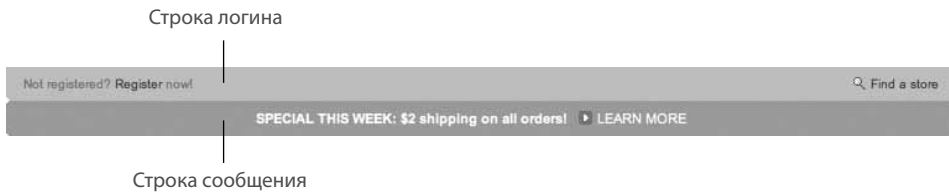


Рис. 3.1. Изображение верхней части сайта магазина «Самый лучший магазин»

Эти полосы (выровненные по ширине страницы) реализованы с помощью вложенных таблиц. Графические элементы (например, скругленные углы полос) и текст находятся в ячейках таблицы.

На рис. 3.2 показано, как можно структурировать ячейки таблицы для верхней полосы — отдельные ячейки обведены линиями. Каждый раздел полосы, как и графические элементы со скругленными углами, находится в отдельной ячейке таблицы. Это приблизительная структура таблицы, но обратите внимание, что таблицы, GIF-разделители и небольшие графические изображения сгруппированы так, что образуют две строки.

Not registered? Register now	Find a store
SPECIAL THIS WEEK: \$2 shipping on all orders! LEARN MORE	

Рис. 3.2. Каждый раздел полосы помещен в отдельную ячейку таблицы

Использование таблиц и GIF-разделителей для размещения графики и текста — это стандартная методика, доведенная до совершенства. С ее помощью построено большинство сайтов. Многие дизайнеры гордятся, что могут воспроизвести

любой макет веб-страницы с точностью до пиксела. Если вы смогли придумать дизайн и распечатать «картинку» на бумаге, то сможете создать на его основе веб-страницу.

Но мы хотим научиться более эффективному веб-дизайну. Для этого нам понадобятся методы, которые позволят повысить удобочитаемость и удобство сайта с помощью экономичной разметки и CSS. Мы применим их к двум горизонтальным полосам на сайте «Самого лучшего магазина», но сначала поговорим о том, почему существующий дизайн не является пуленепробиваемым.

Уязвимые места

Процесс создания горизонтальных полос можно усовершенствовать, чтобы повысить гибкость страницы. Давайте перечислим недостатки существующей реализации, чтобы разобраться, что необходимо изменить в дизайне для повышения его пуленепробиваемости.

Незначащая графика

Помните, что горизонтальные полосы были созданы с использованием вложенных таблиц? Каждый фрагмент полосы связан с отдельной ячейкой таблицы. Незначащая графика, например скругленные углы, была добавлена при разметке «по соседству» с текстом. Такая графика создаст проблемы для людей, которые пользуются текстовыми браузерами или ридерами. Несомненно, с проблемой поможет справиться свойство `alt` (или пустой атрибут `alt=""`) для графики, однако дизайнер «Самого лучшего магазина» про него забыл.

Термин «незначащая графика» относится к графическим элементам, которые не выполняют какой-либо функции и не дают пользователям дополнительной информации. Скругленные углы являются элементами оформления, поэтому чуть позже мы переместим их в таблицу стилей.

Размышления о фиксированной высоте

Если вы попытаете увеличить размер текста, то сразу заметите, что оформление горизонтальных полос нарушается (рис. 3.3). Увеличение размера текста — хороший способ проверки дизайна на гибкость. Вы сможете не только оценить «удобочитаемость» увеличенного текста, но и понять, может ли дизайн корректно обрабатывать контент независимо от его объема и размера шрифта. Иными словами, если в будущем редактор решит сделать две строки рекламного текста вместо одной, как предполагалось ранее при разработке макета, то принцип пуленепробиваемости автоматически увеличит высоту строки. Хороший способ сэкономить время и деньги.



Рис. 3.3. Когда размер текста увеличивается, статичные изображения не изменяются вместе с остальными элементами дизайна

На рис. 3.3 видно, что происходит, когда изображения со скругленными углами вставлены для обрамления строки с фиксированной высотой. Контент большего размера будет выходить за пределы высоты полосы, нарушая целостность дизайна. В данной главе (и многих других) я постараюсь вам объяснить, как важно изменить свое отношение к фиксированной высоте. Увеличение текста, объема контента и т. п. всегда можно компенсировать за счет высоты элемента.

Избыточный код

В соответствии с принципами веб-дизайна для создания таких полос понадобится большое количество кода. Как вы помните из главы 2 «Навигация», вложенные таблицы добавляют в разметку много кода. Избыточный код затрудняет работу серверов, каналов связи и мешает работе других приложений и устройств. К счастью для нас, существует более простой и гибкий способ, позволяющий реализовать наш дизайн.

Пуленепробиваемый подход

Для того чтобы обеспечить пуленепробиваемость полос, мы прежде всего должны обратить внимание на структуру разметки. После чего добавим цвет, разместим элементы и фоновые изображения. В результате у нас будет достаточно гибкий продукт, который сможет обрабатывать текст и контент любого размера. Начнем с разметки, сокращая избыточный код.

СТРУКТУРА РАЗМЕТКИ

Для структурирования двух полос необходимо разобраться с их наполнением. Какие элементы смогут обеспечить решение наших задач? Какие элементы

наиболее важны? При создании разметки «с нуля» я начинаю с ответов на эти вопросы, а затем использую наиболее подходящую структуру. Однозначного ответа на эти вопросы может и не быть, но необходимо задуматься обо всем этом еще до начала работы.

В данном случае нам понадобятся два контейнера — по одному для каждой из строк. Очевидно, что по краям верхней строки будут располагаться два текстовых фрагмента, которые можно оформить как элементы списка, а на второй строке будет находиться абзац текста.

Опишем полную структуру разметки:

```
<ul>
  <li>Не зарегистрированы? <a href="/register/">Зарегистрироваться
    ▶ </a>сейчас!</li>
  <li><a href="/find/">Найти магазин</a></li>
</ul>
<div>
  <p><strong>Специальное предложение:</strong>
    ▶ для всех заказов стоимость
    ▶ доставки составит два рубля! <a href="/special/">ПОДРОБНО</a></p>
</div>
```

Таким образом, на верхней строке находится список, состоящий из двух пунктов, а на второй строке — контейнер `<div>` с одним абзацем текста. Данную структуру легко реализовать, а выглядит она довольно хорошо. Так что мы переходим на экономичную разметку, избавляясь от избыточного кода, связанного с вложенными таблицами.

Кроме того, мы достигли цели — сделали контент более доступным. Независимо от того, какое устройство или приложение вы используете для просмотра страницы, наши строки всегда будут представлять собой список и абзац текста.

Идентификация компонентов

Теперь пора приступить к идентификации элементов, которым мы собираемся присвоить стили. Используя несколько идентификаторов `id`, мы сможем разместить элементы, раскрасить их и вставить изображения, превратив обычную разметку в заверченный дизайн.

```
<ul id="register">
  <li id="reg">Не зарегистрирован?
    ▶ <a href="/register/">Зарегистрироваться</a> сейчас!</li>
  <li id="find"><a href="/find/">Найти магазин</a></li>
</ul>
<div id="message">
```

```

<p><strong>Специальное предложение:</strong>
  ▶ стоимость доставки всех заказов
  ▶ составит два рубля! <a href="/special/">ПОДРОБНОСТИ</a></p>
</div>

```

Мы только что определили идентификаторы для списка в целом, для каждого элемента списка по отдельности и для контейнера второй строки `<div>`. Они пригодятся нам уже через минуту. Возможно, вы удивляетесь, почему мы не можем просто присвоить идентификатор `#message` тегу `<p>` вместо использования `<div>` с вложенным `<p>`. Для завершения нашего дизайна нам понадобятся оба этих элемента, а еще один контейнер позволит разработать гибкий и качественный проект страницы. Но всему свое время.

БЕЗ СТИЛЕЙ

На рис. 3.4 показано, как выглядит наша структура в браузере, когда из стилевой разметки задействован только базовый шрифт (на сайте «Самого лучшего магазина» используется Arial).

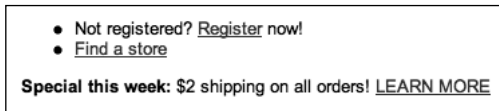


Рис. 3.4. Вид нашего пуленепробиваемого дизайна без применения стилей. Любое устройство, получившее доступ к сайту, сможет показать этот текст

ПРИМЕЧАНИЕ

Перед тем как мы начнем использовать стили, давайте считать, что наши строки находятся на странице «Самого лучшего магазина» шириной 768 пикселей. Иными словами, эти строки располагаются внутри контейнеров (`<div>`, `<table>` и т. п.) с конкретной шириной.

Такой вид будут иметь наши строки, если их попытается отобразить браузер, не поддерживающий CSS. Любое устройство сможет обработать данную разметку. А теперь настало время стилей.

В элементе `<body>` мы задали базовый размер шрифта с помощью ключевого слова `small`.

```

body {
  font-family: Arial, sans-serif;
  font-size: small;
}

```

СОВЕТ

В то время как на сайте «Самый лучший магазин» намеренно используется основной шрифт Arial, истинные шрифтоманы предпочитают шрифт Helvetica (знаменитый шрифт, на базе которого был создан Arial). На большинстве компьютеров Mac имеется Helvetica. Чтобы создать для них иллюзию приоритета, вы можете сначала указать Helvetica, а затем Arial: `body { font-family: Helvetica, Arial, sans-serif; }.`

Большинству людей разница между Arial и Helvetica не бросается в глаза, однако педанты заметят эти незначительные отличия за километр.

Если вы хотите узнать больше, зайдите на сайт дизайнера шрифтов Марка Симонсона: <http://www.ms-studio.com/articlesarialsid.html>.

ДОБАВЛЕНИЕ ФОНА

Пора заняться стилями. Давайте зададим цвет фона для каждой строки — это поможет нам определиться с размерами.

```
#register {
    background: #BDDDB62;
}
#message {
    background: #92B91C;
}
```

Результаты наших действий показаны на рис. 3.5.



Рис. 3.5. Фоновые цвета помогают визуально разделить строки

РАЗМЕЩЕНИЕ КОНТЕНТА

Разместим контент: давайте расположим два пункта списка по краям верхней строки, а сообщение о скидке на доставку — в центре нижней строки.

На рис. 3.6 показаны результаты применения следующего кода CSS:

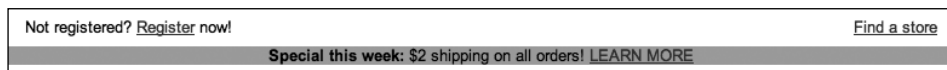


Рис. 3.6. Мы разместили пункты списка по левому и правому краям верхней строки

```
#register {
  margin: 0;
  padding: 0;
  list-style: none;
  background: #BDDDB62;
}
#reg {
  float: left;
  margin: 0;
  padding: 8px 14px;
}
#find {
  float: right;
  margin: 0;
  padding: 8px 14px;
}
#message {
  clear: both;
  text-align: center;
  background: #92B91C;
}
```

Мы удалили верхнее и нижнее поля списка `#register` и скрыли маркеры списка с помощью правила `list-style: none;`.

Элементы списка мы выравнивали по краям. Первый элемент встал слева (`#reg`), а второй — справа (`#find`). В результате оба пункта оказались на одном уровне, но с противоположных концов строки (рис. 3.7).



Рис. 3.7. Метод противопоставления плавающих элементов — удобный способ выравнивания контента по обеим сторонам контейнера

Это очень удобный метод размещения плавающих элементов, позволяющий привязать объекты к сторонам контейнера.

Посмотрите на результат. Мы не только сделали элементы списка плавающими, но и добавили поля вокруг пунктов списка. Теперь нам хватит места, чтобы поставить значок с лупой слева от ссылки **Find a store** (Найти магазин).

Для нижней строки мы добавили свойство `clear: both;`, которое отменяет обтекание с обеих сторон в верхней строке. Для центрования текста мы использовали `text-align: center;`.

ПРОПАВШИЙ ФОН

Куда пропал фон верхней строки? Вспомните: мы уже сталкивались с такой же проблемой в главе 2. Когда внутренние элементы (в данном случае это два тега ``) становятся плавающими, они перестают принадлежать обычной структуре документа. Таким образом, для внешнего `` пункты списка больше не существуют, и `` «не знает», какой высоты и ширины должна быть фоновая полоса.

Давайте поступим так же, как мы действовали в главе 2 «Навигация», — сделаем тег `` плавающим. Кроме того, нам придется использовать `width`, чтобы растянуть полосу на ширину страницы. Большинство разработчиков браузеров слишком буквально интерпретировали спецификации CSS2.0, в которых говорится о том, что «плавающий элемент должен иметь четко определенную ширину» (www.w3.org/TR/REC-CSS2/visuren.html#floats). Если мы не укажем значение, ширина полосы будет зависеть от контента (в данном случае от двух текстовых строк).

```
#register {
    float: left;
    width: 100%;
    margin: 0;
    padding: 0;
    list-style: none;
    background: #BDDDB62;
}
#reg {
    float: left;
    margin: 0;
    padding: 8px 14px;
}
#find {
    float: right;
    margin: 0;
    padding: 8px 14px;
}
#message {
    clear: both;
    text-align: center;
    background: #92B91C;
}
```

На рис. 3.8 показаны наши строки с фоном.

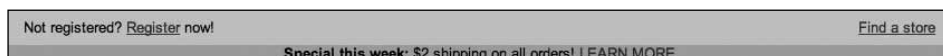


Рис. 3.8. Когда мы делаем элементы плавающими, восстановить фон можно, если контейнер тоже будет плавающим

ДОБАВЛЯЕМ ДЕТАЛИ

Для завершения дизайна нам осталось поработать над деталями. Начнем с верхней строки и добавим закругленные углы к нижним краям фоновой полосы (рис. 3.9).

Обратите внимание, что скругленный угол — это всего лишь треугольник белых пикселей. При просмотре в масштабе 100% (а не при увеличении, как на рис. 3.9) создается иллюзия того, что углы полосы скруглены.

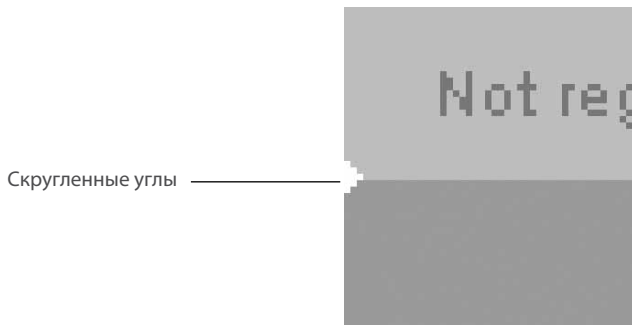


Рис. 3.9. Скругление на самом деле представляет собой уголок из нескольких точек белого цвета, прикрепленный к углу цветной полосы

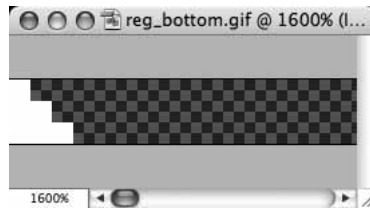


Рис. 3.10. Посмотрите на уголок GIF-файла шириной 768 пикселей (при 1600%-м увеличении)

Удалить окрашенные точки, чтобы создать имитацию закругленных углов, — хорошая идея, а реализовать ее можно с помощью крошечных изображений, окрашенных в фоновый цвет, заданный CSS.

Начнем с того, что в Photoshop (или любом другом графическом редакторе) создадим изображение. Поскольку нам точно известна ширина (768 пикселей), можно ограничиться одним изображением, которое будет содержать левый и правый углы, а затем использовать его как фон в CSS-коде.

На рис. 3.10 показан увеличенный вид левого края, созданного нами изображением. Так что для каждого края полосы можно нарисовать белый треугольник, используя инструмент **Pencil** (Карандаш) с толщиной линии 1px, а остальная часть изображения будет прозрачной. Белые участки элемента будут наложены поверх фоновой цвета, который был задан в CSS. Это позволит создать иллюзию того, что углы цветной плашки скруглены.

Давайте вернемся к элементу `#register` (содержащего ``) и добавим еще одно правило:

```
#register {  
    float: left;  
    width: 100%;  
    margin: 0;  
    padding: 0;  
    list-style: none;  
    background: #BDDDB62 url(img/reg_bottom.gif) no-repeat bottom left;  
}
```

Мы указали цвет фона и разместили поверх него изображение, отключив повтор и выровняв его по левому нижнему углу. Прозрачные части изображения позволят фоновому цвету просвечивать через них, а белые углы будут перекрывать его. Выравнивание изображения по нижнему краю гарантирует, что при любой высоте плашки (любом кегле шрифта или объему контента) углы всегда будут находиться в нужных местах (рис. 3.11).



Рис. 3.11. 3D-изображение, иллюстрирующее порядок наложения элементов

На рис. 3.12 представлен результат нашей работы. Теперь скругленные углы расположены в нижней части верхней плашки.



Рис. 3.12. Фоновый цвет, белые и прозрачные фрагменты изображения создают иллюзию скругленных углов

СОВЕТ

В последних версиях браузеров (включая IE9) CSS3 может работать с несколькими фоновыми изображениями. Теперь с одним элементом вы можете связать множество изображений. Для более подробной информации см. <http://www.css3.info/preview/multiple-backgrounds/>.

ЧЕТЫРЕ СКРУГЛЕННЫХ УГЛА

Во второй плашке скругленные углы находятся как сверху, так и снизу. Не забудьте, что плашка может менять высоту. Для решения этой задачи мы будем использовать два фоновых изображения: одно — такое же как в предыдущем случае (нижняя часть верхней плашки), а другое — верхний край — будет зеркальным отображением первого по вертикали. Так как нам нужны два фоновых рисунка, потребуется два элемента для работы с ними.

К счастью, у нас уже есть эти два элемента. В разметке для второй плашки мы использовали контейнер `<div>` с вложенным `<p>` для контента:

```
<div id="message">
  <p><strong>Специальное предложение:</strong> для всех заказов
  стоимость доставки
    ▶ составит два рубля! <a href="/special/">ПОДРОБНО</a></p>
</div>
```

Давайте свяжем фоновые изображения с этими элементами. Перевернутое изображение белых углов будет привязано к `#message`, а прямое изображение (как в верхней плашке) — к абзацу `<p>`:

```
#message {
  clear: both;
  text-align: center;
  background: #92B91C url(img/mess_top.gif) no-repeat top left;
}
#message p {
  margin: 0;
  padding: 8px 14px;
  background: url(img/reg_bottom.gif) no-repeat bottom left;
}
```

Привязывая верхние углы к элементу `#message` (внешний `<div>`), а нижние углы к `<p>` (рис. 3.13), мы гарантируем, что все четыре угла будут находиться в нужных местах, независимо от размера и объема текста в абзаце. Если мы вставим текст большего размера или большее количество текста, верхние углы всегда останутся сверху, а нижние — снизу.

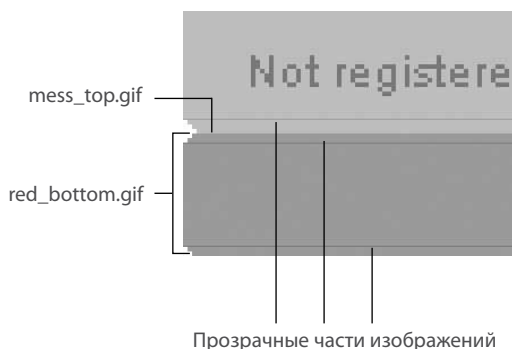


Рис. 3.13. Для нижней плашки используются два изображения, пропускающие фон сквозь прозрачные фрагменты GIF

На рис. 3.14 можно оценить результаты нашей работы. Так же как и в случае с верхней плашкой, прозрачные участки изображения позволяют просвечивать фоновому цвету, а белые уголки маскируют только четыре фрагмента.



Рис. 3.14. С добавлением фоновых цветов полосы начинают принимать форму

СВОЙСТВА ТЕКСТА И ССЫЛКИ

Для завершения работы осталось добавить стили для цветового оформления ссылок и текста. Также нужно добавить изображение со стрелкой для ссылок **Find a store** (Найти магазин) и **Learn more** (Подробнее).

Сначала займемся цветами ссылок и текста на плашках, задав соответствующие правила для всех используемых стилей:

```
#register {
    float: left;
    width: 100%;
    margin: 0;
    padding: 0;
    list-style: none;
    color: #690;
    background: #BDD662 url(img/reg_bottom.gif) no-repeat bottom left;
}
#register a {
    text-decoration: none;
    color: #360;
}
#reg {
    float: left;
    margin: 0;
    padding: 8px 14px;
}
#find {
    float: right;
    margin: 0;
    padding: 8px 14px;
}
#message {
    clear: both;
    font-weight: bold;
    font-size: 110%;
    color: #FFF;
    text-align: center;
    background: #92B91C url(img/mess_top.gif) no-repeat top left;
}
#message p {
    margin: 0;
    padding: 8px 14px;
    background: url(img/reg_bottom.gif) no-repeat bottom left;
}
#message strong {
    text-transform: uppercase;
}
#message a {
    margin: 0 0 0 6px;
```





```
padding: 2px 15px;
text-decoration: none;
font-weight: normal;
color: #FFF;
}
```

Мы задали цвета ссылок для каждого элемента строки `#register`, а также значения по умолчанию для свойств текста `font-size` и `color`, а еще для ссылки в строке `#message` (рис. 3.15).



Рис. 3.15. Вот как выглядят наши строки. Обратите внимание, что мы предусмотрели место для небольших значков, которые будут располагаться слева от ссылок `Find a store` (Найти магазин) и `Learn more` (Подробнее)

Рекламный текст **Special this week:** (Специальное предложение:) мы выделили с помощью тега ``, а свойство `text-transform` помогло нам сделать эти буквы прописными, в то время как сам текст мы не трогали. Зачем это нужно? Сейчас мы хотим выделить слова **Special this week:** (Специальное предложение:) и **Learn more** (Подробнее) прописными буквами, но нам не всегда будет нужно изменять регистр. Вдруг заказчик захочет, чтобы текст был набран в нижнем регистре. Свойство `text-transform` позволяет легко менять регистр текста, оставляя исходные материалы неизменными.

Это всего лишь пример альтернативного сценария, который может вам понадобиться в будущем. Не трогайте исходный текст. Вместо этого используйте свойство `text-transform`, изменяющее регистр при необходимости.

Последний этап

Последним этапом создания пуленепробиваемых полос будет добавление графических изображений для ссылок **Find a store** (Найти магазин). Мы могли бы вставить изображения прямо в разметке, но чтобы упростить задачу обновления, давайте добавим их с помощью CSS как фоновые изображения.

Прежде всего добавим значок с лупой в список верхней строки и выровняем, указав значения `0 50%`, чтобы разместить его слева от соответствующего пункта списка и опустить вниз на 50% (чтобы отцентрировать изображение по вертикали):

```
#find {
  float: right;
  margin: 0;
```

```
padding: 8px 14px;
background: url(img/mag-glass.gif) no-repeat 0 50%;
}
```

На рис. 3.16 показаны результаты вставки значка для ссылки Find a store (Найти магазин).



Рис. 3.16. Фоновое изображение находится слева от пункта списка, отступ для которого был задан заранее

И наконец, добавим стрелку перед ссылкой Learn more (Подробнее). Снова укажем значения 0 50%, которые соответствуют позиции стрелки слева и по центру:

```
#message a {
margin: 0 0 0 6px;
padding: 2px 15px;
text-decoration: none;
font-weight: normal;
color: #FFF;
background: url(img/arrow.gif) no-repeat 0 50%;
}
```

На рис. 3.17 показаны результаты вставки значка со стрелкой слева от ссылки Learn more (Подробнее) путем задания элемента <a>, который располагается в информационной полосе.

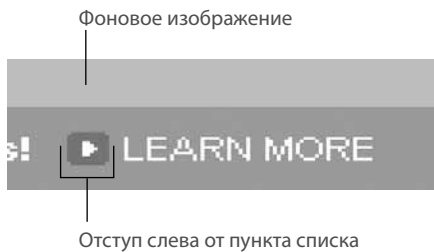


Рис. 3.17. Как и в случае с верхней строкой, мы разместили стрелку левее слов Learn more (Подробнее), только в этот раз связали ее с элементом <a>

На рис. 3.18 показан окончательный результат. Теперь у нас есть две строки, практически идентичные строкам с сайта «Самого лучшего магазина», однако разметка, а также дополнительные меры, которые мы предприняли для размещения фона и текста, делают их пуленепробиваемыми. Нам осталось лишь использовать еще один трюк, необходимый для IE7.



Рис. 3.18. Окончательная пуленепробиваемая версия полос

Специально для IE7

Если вы будете просматривать окончательный дизайн в Internet Explorer 7, то увидите, что над текстом рекламной строки появился дополнительный отступ (рис. 3.19). Причина такого поведения IE7 неизвестна, но другие современные браузеры (а также предыдущая версия, IE6) так себя не ведут. Самый простой (и наименее разрушительный) способ решения проблемы — снова использовать плавающие элементы, которые уже были описаны в этой главе. Мы сделали контейнер `#reg <div>` плавающим, для того чтобы разместить в нем другие плавающие элементы, а теперь, чтобы решить проблему IE7, нам придется применить свойство `float: left;` к контейнеру `#message <div>`.



Рис. 3.19. Окончательный вид страницы в Internet Explorer 7: над контентом второй полосы образовалось пустое пространство

Измененное объявление выглядит следующим образом:

```
#message {
    float: left;
    width: 100%;
    margin: 0;
    padding: 0;
    font-weight: bold;
    font-size: 110%;
    color: #FFF;
    text-align: center;
    background: #92B91C url(img/ship_top.gif) no-repeat top left;
}
```

Мы опять используем `width: 100%`, чтобы раздвинуть плашку на ширину контейнера (в данном случае 768px). Этот метод быстрый, простой и безопасный для всех браузеров. В данном случае, вероятно, это наилучший способ решения проблемы. Но не забывайте, что существуют и иные методы решения проблем с плавающими элементами. В следующей главе мы обсудим другой способ «самоочистки» плавающих элементов, который позволяет контейнерам оставаться независимыми от того, что находится в структуре документа после них.

ПРЕИМУЩЕСТВА ПУЛЕНЕПРОБИВАЕМОГО ПОДХОДА

Упростив разметку и разместив небольшие фоновые изображения, мы удачно переделали информационные строки с помощью пуленепробиваемых методов. Теперь поговорим о том, зачем это было нужно.

РАЗДЕЛЕНИЕ СТРУКТУРЫ И ОФОРМЛЕНИЯ

Мы отказались от таблиц и несущественной графики, а вместо избыточного кода использовали экономичный структурированный HTML-код. Теперь большинство устройств и приложений смогут «понять» нашу разметку, даже если они не поддерживают CSS.

Мы перенесли описание изображений, образующих дизайн-макет веб-страницы, из разметки в таблицу стилей. Если нам понадобится внести изменения в дизайн, то сделать это будет существенно проще, не говоря уже о том, что для этого понадобится значительно меньше кода.

Для создания цветовых переходов из зеленого цвета в красный, синий или любой другой цвет понадобится изменить нескольких правил CSS. И вы сразу же увидите результат.

СКАЖЕМ «НЕТ» ФИКСИРОВАННОЙ ВЫСОТЕ

Вместо того чтобы предположить, что строка должна быть x пикселей в высоту, мы использовали фоновые изображения так, чтобы сохранить целостность дизайна, позволяя строкам и плашкам менять размеры при необходимости. Такой подход позволяет изменять размеры текста (как было описано в главе 1 «Гибкое управление текстом»), независимо от того, какие единицы измерения размера текста мы используем.

На рис. 3.20 показаны перестроенные строки с увеличенным шрифтом. Обратите внимание на то, что скругленные углы и фон остаются неизменными.



Рис. 3.20. При увеличении шрифта строка так же расширяется, однако это не влияет на целостность дизайна

Допустим, редактор хочет добавить два рекламных сообщения на вторую плашку. Мы можем просто вставить еще одну строку текста, не нарушая целостности дизайна. Плашка расширится, чтобы вместить новое сообщение (рис. 3.21). Данный пример иллюстрирует преимущество пуленепробиваемых методов — дизайн учитывает даже непредусмотренные ситуации.



Рис. 3.21. Добавление еще одной строчки на плашку #message не требует вмешательства дизайнера, так как еще в макете мы предусмотрели неограниченное пространство для текста

СОВЕТ

Помните, как мы выравнивали фоновые изображения с лупой и стрелкой по вертикали? Обратите внимание на то, что независимо от размера текста изображение будет отцентрировано по вертикали относительно текста.

Представьте, что клиент или менеджер говорит: «Нам нужна только одна строка текста». И вы начинаете работу в соответствии с требованиями. Но через неделю к вам могут вернуться со словами: «Мы подумали и решили, что нам нужно место для двух строк текста!» Учитывая принцип пуленепробиваемости, вы предусмотрели и эту ситуацию. Конечно, вы можете сказать, что вам нужна еще неделя. Но лучше просто продемонстрируйте им, что вы все предусмотрели, и проиллюстрируйте возможности CSS, позволяющие снизить затраты на обслуживание.

ВОЗМОЖНОСТИ BORDER-RADIUS

В исходном дизайне для скругления углов использовались графические изображения. В качестве альтернативы мы можем еще упростить оформление, удалив

белые треугольники, и использовать CSS3 — свойство `border-radius`. Браузеры, не поддерживающие `border-radius`, будут отображать нескругленные углы. Действительно ли так необходимы скругленные углы? Если ответ отрицательный, то использование CSS вместо изображений станет отличной альтернативой. Его преимущества — большая гибкость, простота изменения, минимальные затраты на техподдержку.

Вернемся к главе 2 и вспомним, как мы использовали особый синтаксис для создания градиента в навигационных вкладках. Свойство `border-radius`, как и градиенты, поддерживается современными браузерами (Safari, Firefox, Opera и IE9+). Но для того чтобы скругленные углы обрабатывались разными браузерами, необходимо использовать префиксы разработчиков.

Попробуем `border-radius` на деле, чтобы посмотреть, как работает это свойство. Сначала уберем изображения, которые мы использовали ранее. Для этого понадобится изменить объявления скруглений в `#register` и `#message`. Также удалим изображения из правил `background`.

```
#register {
    float: left;
    width: 100%;
    margin: 0;
    padding: 0;
    list-style: none;
    color: #690;
    background: #BDD862;
    -webkit-border-bottom-left-radius: 5px;
    -webkit-border-bottom-right-radius: 5px;
    -moz-border-radius-bottomleft: 5px;
    -moz-border-radius-bottomright: 5px;
    border-bottom-left-radius: 5px;
    border-bottom-right-radius: 5px;
}

#message {
    float: left;
    width: 100%;
    margin: 0;
    padding: 0;
    font-weight: bold;
    font-size: 110%;
    color: #FFF;
    text-align: center;
    background: #92B91C;
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    border-radius: 5px;
}
```

СОВЕТ

CSS-правила без префиксов рекомендуется размещать в структуре документа последними. Даже если спецификация изменится, вы сможете гарантировать, что последняя версия команды выполнится, отменяя (иногда) экспериментальную версию, которая используется с префиксом разработчика.

Теперь мы используем правильный синтаксис `border-radius`. Данный синтаксис поддерживается браузерами WebKit (Safari, Chrome) и Mozilla (Firefox). Кроме того, мы используем правила без префиксов, которые «поймут» Opera и IE9+ (и все последующие браузеры, поддерживающие `border-radius`).

Обратите внимание на синтаксис префиксов `-webkit-` и `-moz-`. Если вы не можете его запомнить, используйте сайт <http://border-radius.com>.

На рис. 3.22 видно различие во внешнем виде плашек при просмотре с помощью Safari (*вверху*) и квадратные нескругленные плашки при просмотре через IE8 (*внизу*), который не поддерживает `border-radius`. Вы действительно думаете, что если не все браузеры смогут скруглить углы плашек, то наступит конец света? Скорее всего, нет. В то же время гибкость, которую вы обеспечиваете, удалив декоративные элементы, зачастую перевешивает недостатки, связанные с отсутствием поддержки «красивостей» в старых браузерах.

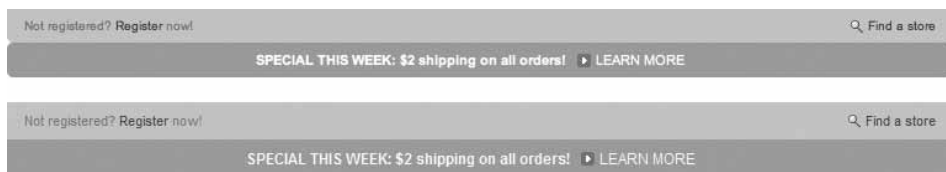


Рис. 3.22. Скругленные углы при просмотре через Safari (*вверху*) и квадратные углы плашек при просмотре через IE8 (*внизу*)

ДРУГОЙ ПРИМЕР ЭЛЕМЕНТОВ С ИЗМЕНЯЕМЫМИ ГРАНИЦАМИ

Для иллюстрации другого сценария, в котором используются расширяемые плашки, я хочу описать процесс, который взял на вооружение при создании гибкого заголовка в шаблоне TicTac для Blogger — популярного блога Google. При создании страницы блог позволяет пользователям выбрать шаблон. Было очень важно сделать шаблоны пуленепробиваемыми, так как от заголовка требовалось, чтобы он вмещал название любой длины.

На рис. 3.23 показан фрагмент шаблона Sample Blog. Графические фрагменты заголовка являются фоновыми изображениями, размещенными средствами CSS. Название сайта должно оставаться текстом, чтобы пользователи Blogger могли использовать шаблон без дополнительных графических элементов.



Рис. 3.23. Образец шаблона ТiсТас для Blogger, в котором гибкость была необходима для изменения текста заголовка

Если заголовок помещается в одну строку, то оформление выглядит отлично. А если название блога будет длиннее (рис. 3.24)? Если я использую фиксированную высоту заголовка, то длинные заголовки (или повышение кегля) перестанут помещаться на плашке, станут нечитаемыми и некрасивыми.



Рис. 3.24. Когда фиксированная высота используется в элементе, вмещающем переменное количество текста, результат оставляет желать лучшего

Неотъемлемой составляющей хорошего дизайна является возможность работы с любым контентом. Для того чтобы заголовок шаблона Blogger мог варьироваться по размеру, я использовал CSS, внедрив два фоновых изображения.

РАЗМЕТКА

Прежде чем приступить к стилям или графике, я занялся структурой разметки заголовка. Мне нужна была возможность подключить два фоновых изображения, а также два элемента разметки, к которым я мог бы их привязать:

```
<header role="banner">  
  <h1>Sample Blog</h1>  
</header>
```

Для заголовка блога я выбрал тег `<h1>`. Но если вы собираетесь использовать его в иных целях, можно взять любой другой тег. Важно, чтобы у вас были два элемента. В элементе HTML5 `<header>` используется `<h1>`. Обратите внимание на то, что я опять использую стандарт WAI-ARIA. Строка `role="banner"` объявляет заголовок `<header>` (элемент, который может повторяться на странице более одного раза) основным баннером — шапкой страницы. Такая роль не только обеспечивает доступность, но и вместо абстрактного `id` позволяет использовать уникальный элемент стиля.

СОЗДАНИЕ ДВУХ ИЗОБРАЖЕНИЙ

Для создания эффекта увеличения размера я использовал два изображения. Высота одного из изображений больше, чем требуется (как я предполагаю). Пользователи Blogger видят, что им виден больший (или меньший — в зависимости от длины заголовка) фрагмент изображения. Второе изображение соответствует нижнему краю заголовочного блока. Оно всегда будет располагаться ниже текста заголовка. На рис. 3.25 показано большое изображение, заполненное повторяющимся узором.

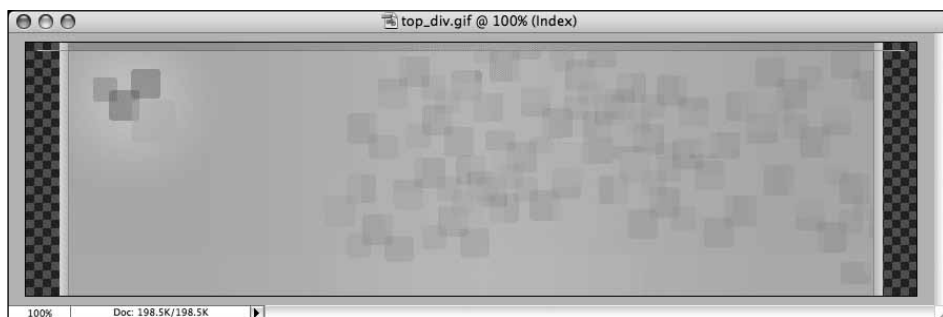


Рис. 3.25. Высота верхнего изображения гораздо больше, чем планировалось

На рис. 3.26 показано второе изображение — фрагмент, который всегда будет располагаться под текстом заголовка. Обратите внимание на то, что верхняя часть этого изображения прозрачная. Это позволило мне разместить изображения друг над другом. При этом `top_div.gif` всегда будет просвечивать сквозь `top_h1.gif`.

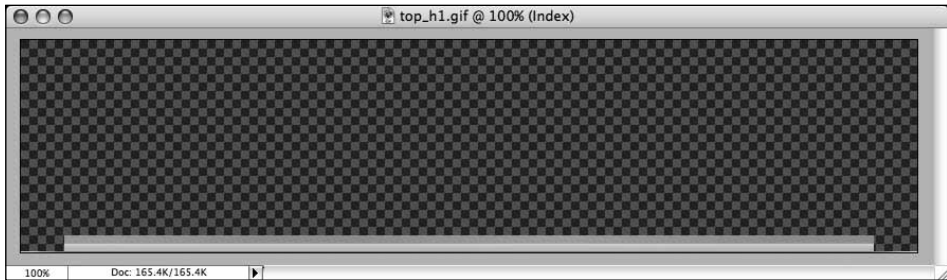


Рис. 3.26. Изображение `top_h1.gif` представляет собой нижний край заголовочного блока и имеет прозрачную заливку в верхней части, чтобы просвечивало подложенное под него изображение

ПРИМЕНЕНИЕ CSS

Для сведения всех фрагментов воедино я использовал два объявления CSS. Сначала я добавил правила для объекта `<header>`:

```
header[role="banner"] {
    margin: 0;
    padding: 0;
    font-family: "Lucida Grande", "Trebuchet MS";
    background: #E0E0E0 url(img/top_div.gif) no-repeat top left;
}
```

Как видите, свойство `font-family` использует значение по умолчанию, а `top_div.gif` располагается сверху слева на светло-сером фоне.

Затем я добавил объявление для заголовка:

```
header[role="banner"] {
    margin: 0;
    padding: 0;
    font-family: "Lucida Grande", "Trebuchet MS";
    background: #E0E0E0 url(img/top_div.gif) no-repeat top left;
}
header[role="banner"] h1 {
    margin: 0;
    padding: 45px 60px 50px 160px;
    font-size: 200%;
    color: #FFF;
    background: url(img/top_h1.gif) no-repeat bottom left;
}
```

С помощью этого кода я указал `font-size` и задал отступ от текста (`padding`), а также разместил `top_h1.gif` внизу слева от заголовка. Так как высота `<header>` зависит от контента, видимая часть фона будет отображаться в зависимости от размера заголовка.

На рис. 3.27 показано, как фрагменты связаны друг с другом.



Рис. 3.27. Верхняя часть заголовка увеличивается пропорционально размеру контента

РАСШИРЕНИЕ

Для тестирования введем длинный текст заголовка и посмотрим, что произойдет. Как показано на рис. 3.28, при использовании большого заголовка становится видна большая часть `top_div.gif`, а `top_h1.gif` опускается вниз, оставаясь под заголовком.



Рис. 3.28. Планирование с учетом неизвестного объема контента гарантирует успех вашей работе

Если текст будет занимать только одну строку, а владелец сайта захочет понизить кегль заголовка, то заголовочный блок уменьшится. Теперь у нас есть красиво оформленный заголовок, который можно использовать как шаблон.

РЕЗЮМЕ

В английском языке есть выражение «пекарская дюжина» (*baker's dozen*). Умный пекарь всегда добавлял дополнительную буханку. Если одно из изделий подгорело или развалилось, всегда оставалось достаточное количество до полной дюжины. Таким образом, пекарь учитывал непредвиденные ситуации. Почему бы не испечь дополнительную пару буханок, если на кону целая партия? Иногда дополнительное изделие пригодится, а иногда — нет.

Когда мы создаем расширяющиеся по вертикали горизонтальные компоненты, мы печем «пекарскую дюжину» — оставляем место для изменения текста или контента. В конце концов, мы экономим время и обеспечиваем пользователю (и редакторам) больший контроль и больше удобств.

При создании горизонтальных элементов дизайна:

- привяжите плавающие элементы к сторонам контейнера;
- уберите незначущую графику из разметки и используйте возможности CSS для работы с фоновыми изображениями, это позволит снизить количество необязательного кода;
- когда объем контента, который будет располагаться в компоненте, неизвестен, используйте два фоновых изображения, для того чтобы границы компонента могли изменяться;
- используйте `border-radius` для скругления углов, если это необходимо для дизайна;
- будьте предусмотрительны — оставляйте больше места, чем, по вашему мнению, может понадобиться.

ГЛАВА 4

КРЕАТИВНОЕ ПЕРЕТЕКАНИЕ



Помните, в предыдущей главе в подразделе «Пуленепробиваемый подход» мы сначала решали, как именно структурировать компоненты? Мы выбирали, какая разметка лучше всего подходит для нашего контента. Я считаю этот этап очень важным при разработке веб-сайта. Именно для того, чтобы вы научились выбирать наиболее подходящие элементы, я предлагаю вам использовать «стилевые маркеры», которые характерны для дизайна с помощью CSS. Но, что более важно, выбор адекватной разметки означает, что контент будет правильно отображаться в различных браузерах и устройствах. Разметка — это основа, и она должна нести идею, независимо от дизайнерского оформления, которое мы можем реализовать с помощью CSS. Кроме того, если мы разделим задачи создания оптимальной структуры (значимый HTML-код) и дизайн (CSS), то изменение дизайна не потребует от вас поиска иголки в стоге сена.

Тем не менее оптимальная структура не должна ограничивать способ отображения контента. Как мы узнаем в этой главе, float позволяет структурировать контент не хуже таблицы, а кода для этого понадобится гораздо меньше. Используя минимум разметки, мы гарантируем, что все браузеры, приложения и устройства смогут отображать контент без каких-либо проблем. Кроме того, мы упрощаем работу дизайнеров и разработчиков, которым понадобится изменять и редактировать компоненты сайта.

Давайте начнем с элемента, объединяющего изображение, название и описание. Это довольно стандартный шаблон, используемый на многих сайтах, и мы можем найти для него элегантное решение, задействующее минимальную разметку и CSS.

ОБЩЕПРИНЯТЫЙ ПОДХОД

Возможно, вам уже попадались такие элементы, ведь они часто встречаются на сайтах. Такой блок состоит из изображения, названия и описания — это могут быть аннотации статей, продуктов, описания файлов и т. п. Часто в таком блоке присутствует изображение, с боку от которого располагаются название и краткое описание (рис. 4.1).



Рис. 4.1. Блок, состоящий из изображения, названия и описания, довольно часто встречается на веб-страницах

На сайте могут использоваться сразу несколько таких блоков. Каждый из них является ссылкой на статью, продукт или другой объект. Как правило, для их структурирования используют тег `<table>` и GIF-разделитель, управляющий пространством между элементами (рис. 4.2).



Рис. 4.2. Таблицы и GIF-разделители позволяют управлять размещением элементов

Конечно, можно сказать, что мы имеем дело с табличными данными (такими, как электронные таблицы, календари, статистика), но я не собираюсь рассуждать об уместности таблиц. Мы просто используем конкретный компонент с реального сайта как основу, чтобы переделать его, сократив объем разметки, и использовав возможности CSS для достижения результата, похожего на

таблицу. Давайте откажемся от лишней разметки и ненужных графических элементов, чтобы создать более гибкий, доступный и управляемый (с редакторской точки зрения) сайт.

На рис. 4.3 показан фрагмент страницы сайта Furniture Shack (выдуманного интернет-магазина домашней мебели), который я и предлагаю реконструировать. Данный модуль содержит три блока с аннотациями товаров, продающихся в магазинах Furniture Shack.

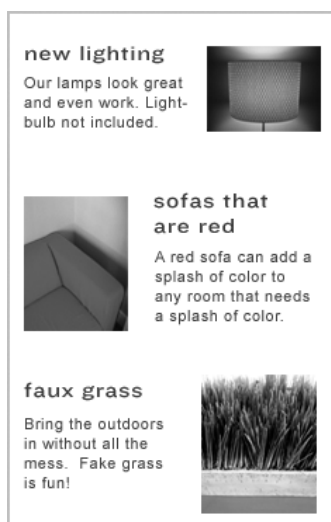


Рис. 4.3. На домашней странице Furniture Shack используются блоки с аннотациями

Каждый блок содержит изображение продукта, его название и краткое описание, а оформление соответствует фирменному стилю компании.

Если заглянуть внутрь, мы увидим, что модуль был создан на базе вложенных таблиц и GIF-разделителей. Название товара также представлено в графическом виде. Таким образом, для создания этого макета понадобился тяжеловесный объемный код.

Уязвимые места

В данном примере количество кода является достаточной причиной для того, чтобы поискать другой способ реализации (рис. 4.4). Сокращение кода не только позволит уменьшить размер файла (а, значит, освободит место на сервере и ускорит процесс загрузки), но и упростит редактирование информации. Когда мы решаем, какую разметку лучше использовать, необходимо помнить, что простой

код всегда лучше как для серверов, так и для редакторов сайта. Воспринимайте его как основу гибкого обслуживания.

```

spacer.gif" width="1" height="3" alt="" border="0"><br><a href="http://ww2.furniturehackshack.com/cat/
themeindex.cfm?cid=thmgorsaltsrc=shpcfur%7Crshop" onMouseOut="MM_swapImgRestore()"
onMouseOver="MM_swapImage('editoeg3','','http://a451.g.akamai.net/7/451/1713/0001/image2.styleinamerica.com/
fsimgs/images/bld-20050401-004/common/arr_right45_white.gif',1)">new lighting</a><br><br><a href="http://ww2.furniturehackshack.com/view.cfm?pg=body/orrguide"
onMouseOut="MM_swapImgRestore()" onMouseOver="MM_swapImage('editoeg4','','http://a451.g.akamai.net/7/451/
1713/0001/image2.styleinamerica.com/fsimgs/images/bld-20050401-004/common/arr_right45_white.gif',1)">Our
lamps look great and even work. Lightbulb not included.</
a><br><br></font></a></td>
</tr>
</table></div><div align="center">
<table border="0" cellpadding="0" cellspacing="0" width="199">
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td valign="top" width="188"><table border="0" cellpadding="0" cellspacing="0"><tr><td><br><font
face="Verdana, Geneva, Arial, Helvetica" size="1" color="666666" class="text"><a href="http://
ww2.furniturehackshack.com/cat/roomindex.cfm?cid=romliv" onMouseOut="MM_swapImgRestore()"
onMouseOver="MM_swapImage('edit5sss','','http://a451.g.akamai.net/7/451/1713/0001/image2.styleinamerica.com/
fsimgs/images/bld-20050401-004/common/arr_right45_white.gif',1)">sofas that are red</a><br><br>

```

Рис. 4.4. Избыточный код

Из-за избыточности кода общепринятый подход снижает доступность сайта для некоторых приложений и устройств. Доступ к громоздкой конструкции из вложенных таблиц и GIF-разделителей, несомненно, станет испытанием для любого пользователя. Избавиться от необязательного кода и повысить доступность сайта не означает пренебречь дизайном.

ПУЛЕНЕПРОБИВАЕМЫЙ ПОДХОД

Для упрощения структуры давайте заменим таблицы на минимально необходимую разметку. Затем обратимся к CSS, чтобы разработать похожий на таблицу шаблон аннотации (блока, состоящего из изображения, названия и описания). Дизайн при этом останется столь же привлекательным.

Как всегда, сначала нужно решить, как структурировать модуль с помощью разметки.

БЕСКОНЕЧНЫЕ ВОЗМОЖНОСТИ РАЗМЕТКИ

Перед тем как приступить к выбору разметки, давайте вспомним нашу цель и кратко опишем, что нам нужно получить.

На рис. 4.5 представлена модель нашей структуры. Нам понадобится внешний контейнер для модуля, чтобы разместить все элементы и определить границы. Внутри модуля будут располагаться три аннотации, содержащие изображение (с выравниванием по одной из сторон), название и краткое описание.

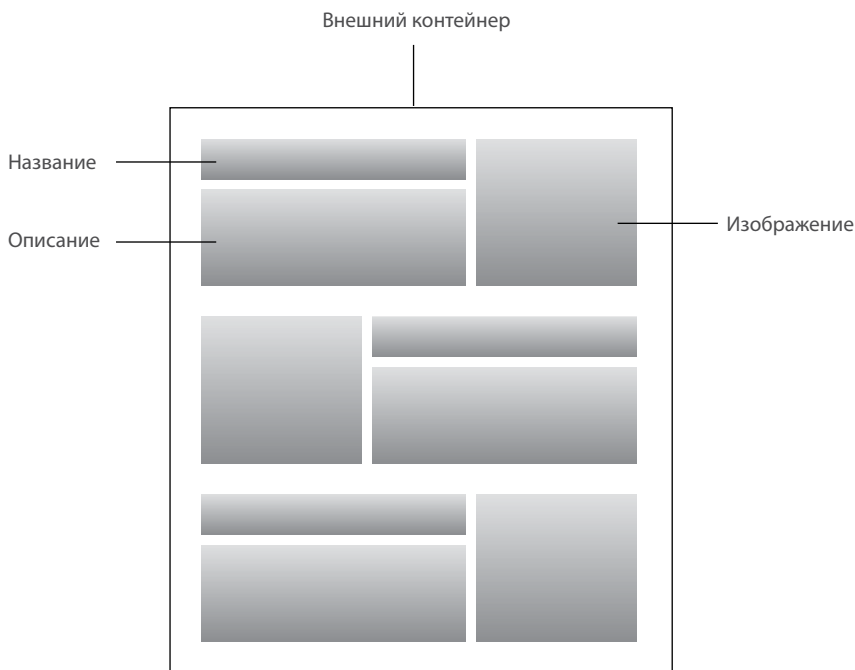


Рис. 4.5. Модель структуры поможет создать макет

Я уже знаю, что нас ожидает, поэтому могу заранее сказать, что нам понадобится отдельный элемент, в который будет «вкладываться» каждая аннотация. Этот элемент позволит объединить изображение, название и описание в единый блок. Да и с точки зрения семантики имеет смысл изолировать каждую аннотацию с помощью элемента-контейнера.

Давайте теперь проанализируем возможные варианты разметки. Как и в большинстве случаев, не существует однозначно правильного ответа, есть только более или менее удачные варианты.

ИСПОЛЬЗОВАНИЕ СПИСКА ОПРЕДЕЛЕНИЙ

По моему мнению, списки определений используются до обидного редко. В особенности это касается ситуаций, когда не существует очевидных групп из названия и описания. Список определений содержит элемент `<dl>`, объединяющий любое количество терминов `<dt>` и описаний `<dd>`:

```
<dl>
  <dt>Это термин</dt>
  <dd>Это описание.</dd>
</dl>
```

W3C (<http://www.w3.org/TR/html5/grouping-content.html#the-dl-element>) предлагает и другие варианты использования этих элементов. Там отмечено, что: а) списки определений могут состоять из нескольких терминов и (или) определений, и б) списки определений могут использоваться для терминов и определений, метаданных, тем и значений, вопросов и ответов или любых других групп данных, состоящих из названий и значений. Так что таким образом можно записать даже диалог:

```
<dl>
  <dt>Молодой полицейский</dt>
  <dd>В машине было что-то ценное?</dd>
  <dt>Парень</dt>
  <dd>А, да, да... магнитола, несколько пленок Creedence
    ▶ и... эээ... мой портфель.</dd>
  <dt>Молодой полицейский</dt>
  <dd>[многозначительная пауза] А в портфеле?</dd>
  <dt>Парень</dt>
  <dd>Мм, бумаги, просто рабочие бумаги.</dd>
  <dt>Молодой полицейский</dt>
  <dd>И чем же вы занимаетесь?</dd>
  <dt>Парень</dt>
  <dd>Я безработный.</dd>
</dl>
```

Мне (и другим дизайнерам) нравится пункт б. Я использую списки определений, чтобы структурировать информацию в разных ситуациях.

Я решил остановиться на списках определений для структурирования каждого блока, состоящего из изображения, названия и описания.

СТРУКТУРА РАЗМЕТКИ

Для того чтобы задание стало интереснее, вместо контента сайта Furniture Shack я буду использовать собственный контент — несколько фотографий, сделанных во время моей поездки в Швецию. Каждая аннотация будет состоять из списка определений, содержащего название — `dt`, а также изображения и описания — `dd`.

Как я уже говорил, нам понадобится внешний контейнер для задания ширины и создания границы вокруг компонента.

Структура разметки будет выглядеть следующим образом:

```
<article id="sweden">
  <dl>
    <dt>Стокгольм</dt>
    <dd></dd>
    <dd>This was taken in Gamla Stan
      ▶ (Old Town) in a large
      ▶ square of amazing buildings.</dd>
  </dl>
  <dl>
    <dt>Gamla Uppsala</dt>
    <dd></dd>
    <dd>The first three Swedish kings are buried here,
      ▶ nder ancient burial mounds.</dd>
  </dl>
  <dl>
    <dt>Perpetual Sun</dt>
    <dd></dd>
    <dd>During the summer months, the sun takes forever to
      ▶ o down. This is a good thing.</dd>
  </dl>
</article>
```

Мы присвоили внешнему контейнеру `<article>` (новый элемент HTML5) идентификатор `id` со значением `sweden`. Внутри контейнера находятся три списка определений. Каждый список содержит название, изображение и краткое описание. Возможно, вы удивитесь, почему я использовал три отдельных тега `<dl>` вместо одного большого списка. Причину я поясню немного позднее.

Простой стиль

Без стилового оформления наша структура при просмотре в браузере будет иметь вид, представленный на рис. 4.6.

Как правило, браузер сдвигает элементы `<dd>`, делая более наглядными связи между ними и заголовками `<dt>`. Так как мы выбрали простую экономичную разметку, любое устройство или браузер сможет без проблем реализовать наш замысел. Но не такого результата мы хотим добиться. Следующим шагом станет добавление стилей.

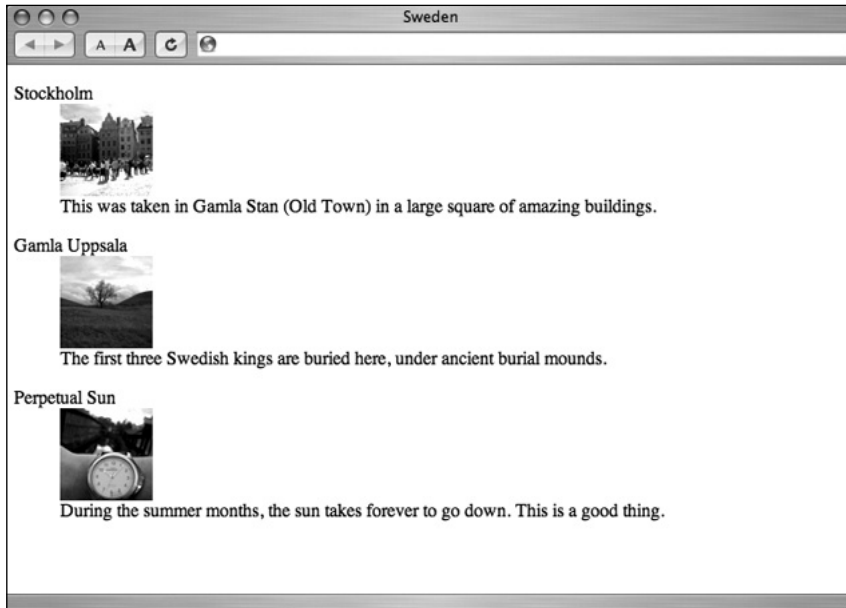


Рис. 4.6. Даже если браузер не может использовать CSS, наша структура вполне читабельна и понятна

ПРИСВОЕНИЕ СТИЛЯ КОНТЕЙНЕРУ

Начнем с декларации, которая задаст ширину и создаст голубую рамку вокруг списка. Добавим стилевое оформление внешнему контейнеру `<article>`, которому мы присвоили значение `id="sweden"`.

ПРИМЕЧАНИЕ

Шрифт Arial был по умолчанию назначен для всей странице, а в `font-size` мы использовали ключевое слово `small`.

```
#sweden {
  width: 300px;
  border: 2px solid #C8CDD2;
}
```

Указав ширину `300px` и задав цветную рамку для всего блока, мы получим результат, представленный на рис. 4.7.



Рис. 4.7. Зададим ширину 300 пикселей для всего блока

РАСПОЗНАВАНИЕ ИЗОБРАЖЕНИЙ

Чтобы упростить управление сайтом, нам нужно сделать один шаг — добавить `class` каждому `<dd>`-элементу, который содержит изображение. Поскольку мы сделаем изображения (но не текстовые описания) плавающими, нам понадобится идентификация этих элементов в разметке, чтобы позднее назначить для них стиль с помощью CSS:

```
<article id="sweden">
  <dl>
    <dt>Стокгольм</dt>
    <dd class="img"></dd>
    <dd>This was taken in Gamla Stan (Old Town)
      ▶ in a large
      ▶ square of amazing buildings.</dd>
  </dl>
  <dl>
    <dt>Gamla Uppsala</dt>
    <dd class="img"></dd>
    <dd>The first three Swedish kings are buried here,
      ▶ under ancient burial mounds.</dd>
  </dl>
```

```

<dl>
  <dt>Perpetual Sun</dt>
  <dd class="img"></dd>
  <dd>During the summer months, the sun takes forever to
    ▶ go down. This is a good thing.</dd>
</dl>
</article>

```

Теперь, когда каждому элементу `<dd>`, содержащему изображение, присвоен класс `class="img"`, можно двигаться дальше.

ПРИМЕНЕНИЕ БАЗОВЫХ СТИЛЕЙ

Давайте применим базовые стили для каждой аннотации, а размещением изображений займемся чуть позже.

Для того чтобы задать поле шириной 20 пикселей вокруг аннотаций и внутри модуля (рис. 4.8), давайте уберем отступы между контейнером `<article>` и аннотациями. Прежде всего добавим 10-пиксельные поля над и под контейнером `<article id="sweden">`. Затем добавим 10-пиксельные поля над и под каждым элементом `<dl>` и 20-пиксельные поля слева и справа от каждого элемента `<dl>`.

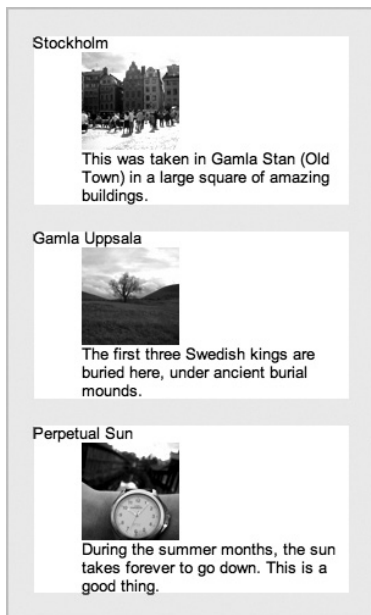


Рис. 4.8. Наша цель — последовательно задать отступ 20 пикселей (выделенный серым) по внутренней стороне модуля вокруг аннотаций

```
#sweden {
  width: 300px;
  padding: 10px 0;
  border: 2px solid #C8CDD2;
}
#sweden dl {
  margin: 10px 20px;
  padding: 0;
}
```

На рис. 4.9 показан внешний вид нашего блока после добавления отступов. Мы также обнулили все значения полей, которые могут быть связаны со списками определений. Теперь модуль выглядит значительно привлекательнее.

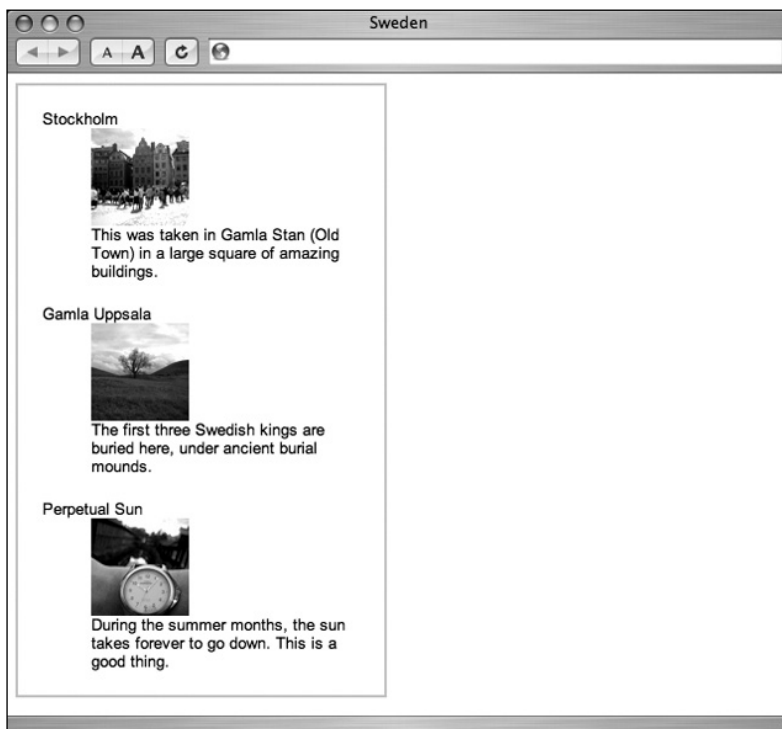


Рис. 4.9. Мы распределили отступы по элементам: теперь модуль обретает форму

Пора добавить цвет и заняться обработкой названия каждой аннотации, применив стиль к элементам `<dt>`:

```
#sweden {
  width: 300px;
  padding: 10px 0;
```

```
border: 2px solid #C8CDD2;
}
#sweden dl {
margin: 10px 20px;
padding: 0;
}
#sweden dt {
margin: 0;
padding: 0;
font-size: 130%;
letter-spacing: 1px;
color: #627081;
}
```

ПРИМЕЧАНИЕ

В этом примере я использовал браузер Safari с эффектом сглаживания текста под Mac OS X. Аналогичный результат дает функция ClearType в Windows. Пользователи других операционных систем могут увидеть другие результаты.

На рис. 4.10 видно, что мы укрупнили название, изменили цвет и увеличили межбуквенное расстояние с помощью свойства `letter-spacing`, повторяя оформление Furniture Shack, где вместо текста с присвоенным стилем использовались изображения.

Давайте добавим стилевое оформление элементам `<dd>`, взяв за основу мелкие серые буквы Furniture Shack:

```
#sweden {
width: 300px;
padding: 10px 0;
border: 2px solid #C8CDD2;
}
#sweden dl {
margin: 10px 20px;
padding: 0;
}
#sweden dt {
margin: 0;
padding: 0;
font-size: 130%;
letter-spacing: 1px;
color: #627081;
}
#sweden dd {
margin: 0;
```





```
padding: 0;
font-size: 85%;
line-height: 1.5em;
color: #666;
}
```

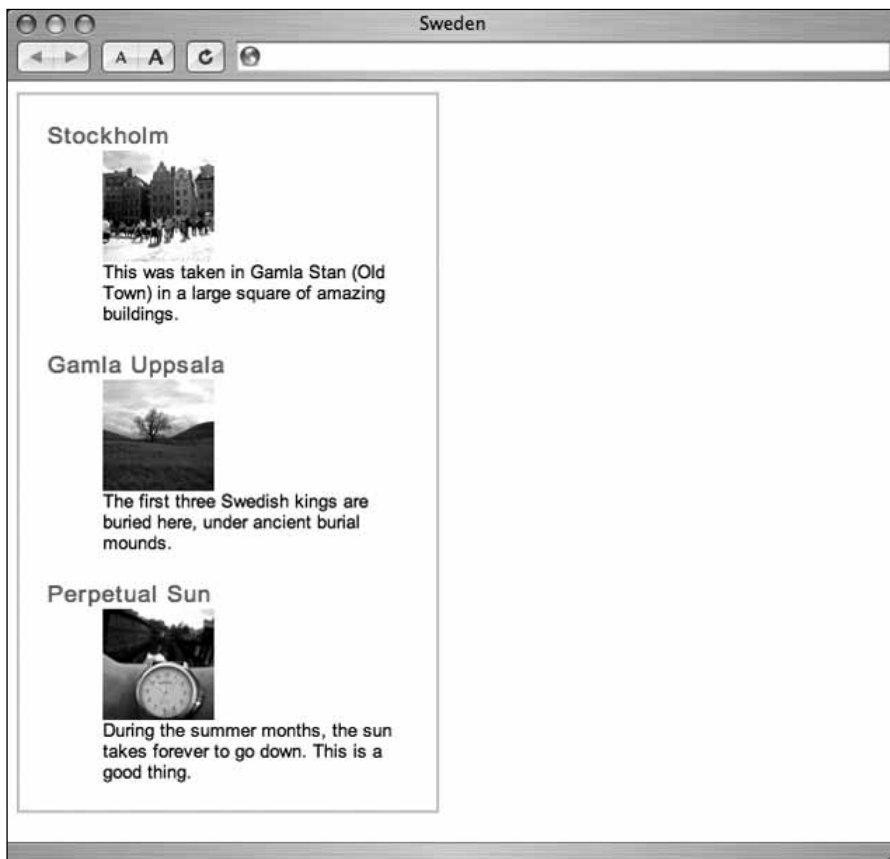


Рис. 4.10. Простое назначение тексту стиля творит чудеса и устраняет необходимость использования шрифтов в виде изображений

На рис. 4.11 показан результат нашей работы. Мы понизили кегль и добавили серый цвет краткому текстовому описанию. Мы также увеличили `line-height` (интерлиньяж) до показателя в 1,5 раза больше высоты обычного текста. Это делает описание более привлекательным. Не забывайте удалять отступы, которые браузер по умолчанию применяет к элементам `<dd>`. Мы их просто обнулили.

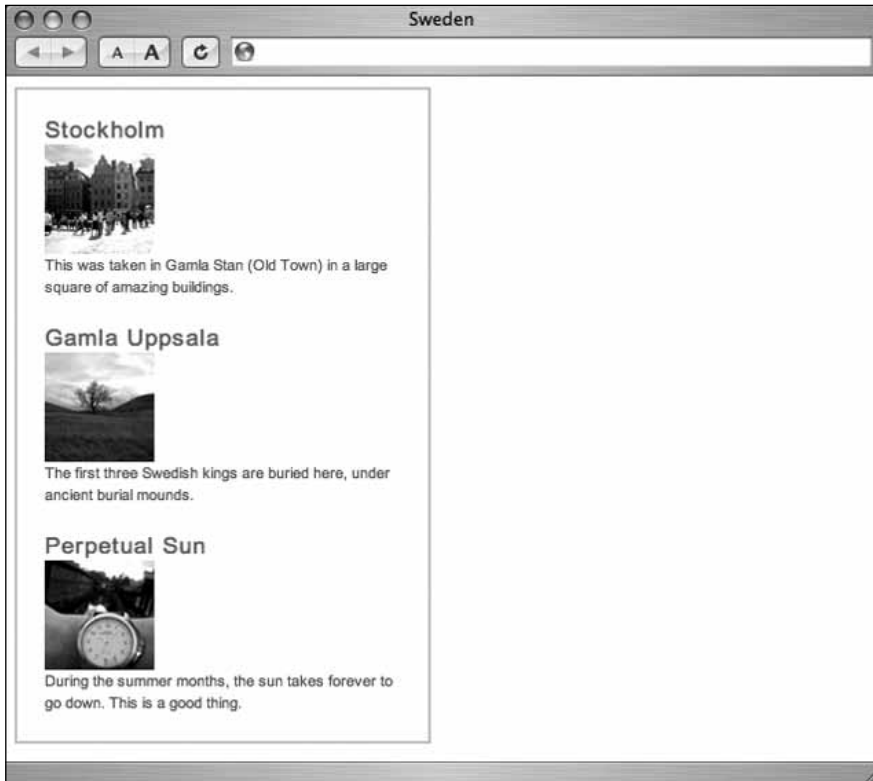


Рис. 4.11. Для повторения оформления сайта Furniture Shack мы уменьшили текст описания и покрасили его в серый цвет

РАЗМЕЩЕНИЕ ИЗОБРАЖЕНИЙ

Давайте разместим изображение сбоку от названия и описания. Но сначала мы попробуем просто выровнять элементы по левому краю. После того как мы разберемся с этой задачей, можно будет повторить оформление Furniture Shack.

Из-за порядка следования элементов (название, изображение, а лишь затем описание), если мы просто сделаем изображение плавающим и выровняем его по левому краю, название окажется над всеми остальными элементами (рис. 4.12). Но нам нужно, чтобы верхний край изображения и название находились на одном уровне.

В предыдущих случаях я просто изменял порядок элементов, размещая изображение в `<dt>` и используя `<dd>` для названия и описания. Когда изображение стоит первым, то мы легко можем прибить его к краю и добиться нужного результата. Однако в этом случае лучше определить название как `<dt>`, за которым

следуют изображение и текст описания (как мы и сделали в нашем примере). Для достижения результата — создания оптимальной разметки и размещения изображения на одном уровне от названия и описания — осталось лишь немного CSS-кода.

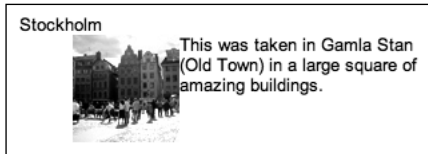


Рис. 4.12. Так как в разметке изображение идет после названия, если мы просто прибьем его к левому краю, название будет находиться над всеми элементами

ВЫРАВНИВАНИЕ ПО КРАЯМ

Возможно, вы помните как мы выравнивали по краям плавающие элементы в главе 3 «Эластичные строки». Мы использовали `float` для размещения элементов на противоположных сторонах контейнера. Здесь мы прибегнем к тому же методу, выравнивая изображение по краю и располагая его слева от названия и описания, чтобы сохранить оптимальный порядок разметки.

Мы специально определили для элементов `<dd>` свойство `class="img"` — это позволит сделать изображения внутри элементов плавающими, выровняв их по краю и оставив описание на месте.

Для начала заставим элементы `<dt>` сдвинуться правее, а изображения — левее:

```
#sweden {
    width: 300px;
    padding: 10px 0;
    border: 2px solid #C8CDD2;
}
#sweden dl {
    margin: 10px 20px;
    padding: 0;
}
#sweden dt {
    float: right;
    margin: 0;
    padding: 0;
    font-size: 130%;
    letter-spacing: 1px;
    color: #627081;
}
```

```
#sweden dd {
  margin: 0;
  padding: 0;
  font-size: 85%;
  line-height: 1.5em;
  color: #666;
}
#sweden dd.img img {
  float: left;
}
```

Выравнивая плавающие элементы по краям, мы можем разместить изображение слева от названия и описания, хотя название по-прежнему остается на первом месте в структуре разметки (рис. 4.13).



Рис. 4.13. Выравнивание по краям в действии. Верхний край изображения и название находятся на одном уровне

Обратите внимание, что хотя изображение находится на нужном месте, текстовое описание оказалось между изображением и названием. Для исправления этой ситуации придется произвести некоторые подсчеты (рис. 4.14).

Нужно задать ширину элементов `<dt>` таким образом, чтобы они занимали все свободное пространство в верхней части. Для расчета ширины возьмем общую ширину модуля (300 пикселей), вычтем поля вокруг списка определений (дважды по 20 пикселей), вычтем ширину изображения (80 пикселей). Результат: 180 пикселей.

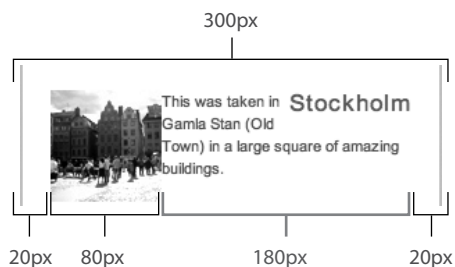


Рис. 4.14. Определение ширины для элементов `<dt>` требует расчетов

Если в декларации элементов `<dt>` мы укажем ширину 180px, то гарантируем, что все элементы окажутся на своих местах:

```
#sweden dt {
    float: right;
    width: 180px;
    margin: 0;
    padding: 0;
    font-size: 130%;
    letter-spacing: 1px;
    color: #627081;
}
```

На рис. 4.15 можно увидеть результаты нашей работы. Мы разместили изображение, название и описание, воспользовавшись выравниванием плавающих элементов по краям.



Рис. 4.15. После задания ширины для элементов `<dt>` все элементы встали на свои места

НЕ БУДЕМ ОГРАНИЧИВАТЬ РАЗМЕРЫ ОПИСАНИЯ

Пока что в нашем примере размер описаний не создавал проблем — текст заполнял практически всю область до нижнего края изображения. Нам не нужно было думать об очистке плавающих элементов. Чтобы понять, что может произойти, если описание окажется короче, посмотрите на рис. 4.16. Очевидно, что это решение нельзя назвать пуленепробиваемым.



Рис. 4.16. Короткий текстовый блок в сочетании с плавающим изображением дают неожиданный результат

Если вы только начинаете пользоваться плавающими элементами, задача правильной очистки может сильно осложнить вашу жизнь. Когда элемент плавает, он изымается из системы обтекания документа и перестает как-либо влиять на вертикальные блоки, расположенные далее. Плавающие элементы всегда выходят за границы высоты своего контейнера. Если описание окажется длинным и будет заканчиваться на одном уровне с нижним краем плавающего элемента или даже продолжаться за пределы этой границы, все будет в порядке. Проблемы возникают, когда контент, расположенный рядом с плавающим изображением, оказывается не достаточно большим.

Обратите внимание на один из элементов на рис. 4.17, который обведен в рамку. Изображение существенно больше по высоте, чем текстовый блок из названия и аннотации. Так как изображение сдвинуто влево, то следующий за ним спи-

сок определений попытается его обтечь. Значит, нам нужно очистить область плавающего изображения, перед тем как начнется следующий анонс. В старые времена для этого было достаточно написать `<br clear="all">`. Конечно, этот способ продолжает работать, но постарайтесь его избегать, тем более что атрибут `clear` перестал функционировать в последних версиях HTML. Для очистки плавающих элементов лучше использовать CSS (а не разметку). Давайте рассмотрим способы реализации такой очистки.



Рис. 4.17. Рамка показывает реальную границу элемента `<dl>`

Самоочистка плавающих блоков

Существует несколько способов очистки плавающих элементов с помощью CSS. Чтобы ознакомиться с некоторыми из них, почитайте статью Эрика Мейера «Containing Floats» (www.complexspiral.com/publications/containing-floats/).

Давайте рассмотрим три популярных метода самоочистки плавающих элементов (три варианта использования CSS с контейнером, который содержит плавающие элементы). Самоочистка плавающих элементов позволяет контейнеру оставаться независимым, при этом неважно, что находится до или после него в структуре документа, — вот ключевой принцип пуленепробиваемости. Если мы сделаем «блоки» (контейнеры с мини-макетами) независимыми, то сохраним их стабильность при перемещении или редактировании. Самоочистка позволяет разбить структуру, использующую плавающие элементы, на модули и не требует дополнительной разметки (например, `<br clear="all">`). Вот эти методы:

- **Использование плавающих элементов для фиксации плавающего элемента** (описывается в статье Эрика Мейера и уже использовалось в книге). Данный метод зависит от того, что располагается после контейнера на странице, его достоинства — кроссбраузерность и простая реализация.
- **Простая очистка плавающих элементов с использованием `overflow`**. Возможно, это самый простой способ, но и у него есть недостатки (www.sitepoint.com/blogs/2005/02/26/simple-clearing-of-floats/).

- Простая очистка с помощью сгенерированного контента (<http://positioniseverything.net/easyclearing.html>). Для реализации метода в Internet Explorer требуются танцы с бубнами. Однако после того как вы поймете суть, скорее всего, вы согласитесь, что это самый надежный способ.

Давайте начнем с первого метода — плавающие элементы для фиксации плавающего элемента — и используем его в нашем примере.

Плавающий элемент для фиксации плавающих элементов

Если сам контейнер является плавающим элементом, он может растягиваться, охватывая все плавающие элементы, расположенные внутри него. В соответствии с рекомендациями Эрика Мейера, нам нужно выровнять все элементы `<dl>` по левому краю, для того чтобы каждый следующий анонс оказался под предыдущим плавающим изображением. Кроме того, нам следует сделать плавающим весь контейнер `<article>`, чтобы создать границу для всех плавающих элементов:

```
#sweden {
    float: left;
    width: 300px;
    padding: 10px 0;
    border: 2px solid #C8CDD2;
}
#sweden dl {
    float: left;
    margin: 10px 20px;
    padding: 0;
}
#sweden dt {
    float: right;
    width: 180px;
    margin: 0;
    padding: 0;
    font-size: 130%;
    letter-spacing: 1px;
    color: #627081;
}
#sweden dd {
    margin: 0;
    padding: 0;
    font-size: 85%;
    line-height: 1.5em;
    color: #666;
}
#sweden dd.img img {
    float: left;
}
```

Добавив к нашему примеру эти правила, мы корректно очистили плавающие элементы. Теперь все анонсы располагаются друг под другом независимо от размера описания (рис. 4.18).



Рис. 4.18. Очистка плавающих элементов работает даже с короткими аннотациями

Последние штрихи

Для окончательной отладки нашего примера добавим интервал между изображением и текстом и позволим изображениям «уплывать» как влево, так и вправо.

Интервал между изображениями и аннотациями появится, если мы зададим нашему изображению отступ справа, а затем вычтем это значение из ширины, указанной для `<dt>`. Давайте добавим границу вокруг каждого изображения, для этого достаточно дописать несколько строк в основную таблицу стилей. CSS-код остается достаточно компактным:

```
#sweden {
    float: left;
    width: 300px;
    padding: 10px 0;
```

```

        border: 2px solid #C8CDD2;
    }
    #sweden dl {
        float: left;
        width: 260px;
        margin: 10px 20px;
        padding: 0;
    }
    #sweden dt {
        float: right;
        width: 162px;
        margin: 0;
        padding: 0;
        font-size: 130%;
        letter-spacing: 1px;
        color: #627081;
    }
    #sweden dd {
        margin: 0;
        padding: 0;
        font-size: 85%;
        line-height: 1.5em;
        color: #666;
    }
    #sweden dd.img img {
        float: left;
        margin: 0 8px 0 0;
        padding: 4px;
        border: 1px solid #D9E0E6;
        border-bottom-color: #C8CDD2;
        border-right-color: #C8CDD2;
        background: #FFF;
    }

```

На рис. 4.19 можно увидеть результат применения новых стилей. Мы добавили 8-пиксельный отступ справа каждому плавающему изображению и создали рамку вокруг изображений — 4-пиксельный отступ и 1-пиксельная рамка. Так как мы увеличили ширину изображения, необходимо сложить все приращения, а затем вычесть полученное значение из общей ширины, которая используется элементами `<dt>`. Расчет будет следующим: 8 (отступ справа) + 4 (отступы с обеих сторон) + 1 (рамка с обеих сторон) = 18 пикселей. Таким образом, мы уменьшили ширину элементов `<dt>` до 162 пикселей, чтобы учесть дополнительное пространство, необходимое изображению.

Чтобы создать эффект объема, мы симитировали тени — сделали рамку справа и снизу немного темнее, чем слева и сверху. Таким образом, создается ощущение, что изображение лежит поверх модуля (рис. 4.20).

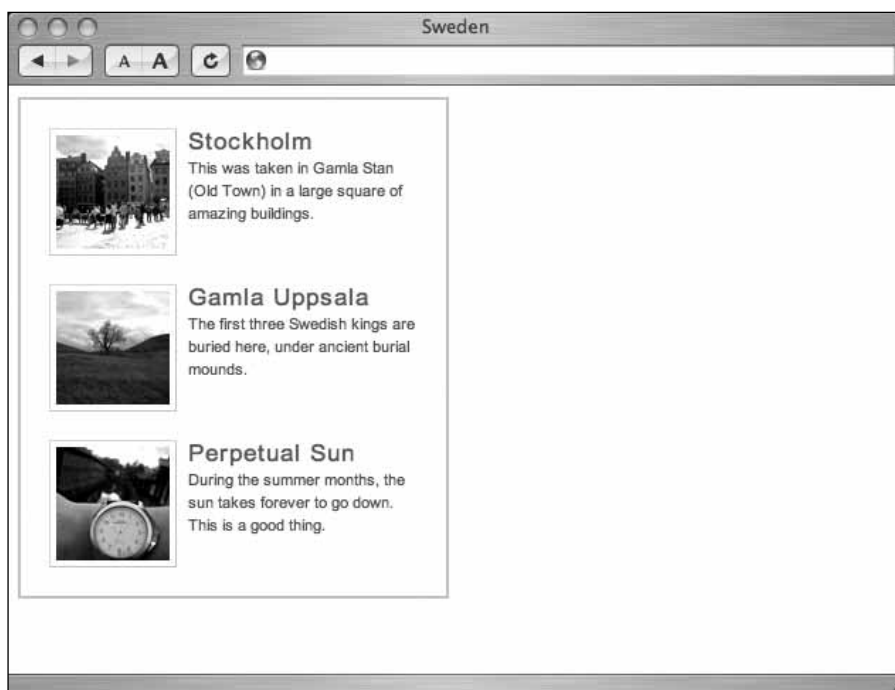


Рис. 4.19. Мы задали правильное расстояние между изображениями и текстом



Рис. 4.20. Простая уловка для создания эффекта объема — сделайте правую и нижнюю границы изображения более темными

МЕНЯЕМ ВЫРАВНИВАНИЕ

Давайте повторим еще один элемент оформления Furniture Shack — распределим изображения в шахматном порядке: первое выравнивается по левому краю, вто-

рое — по правому и т. п. Нам нужно при необходимости изменять положение картинки. Для этого добавим класс `class="alt"` для элемента `<dl>`.

Начнем с `<dl>`, направление выравнивания которого нужно изменить, — это будет второй анонс:

```
<article id="sweden">
  <dl>
    <dt>Stockholm</dt>
    <dd class="img"></dd>
    <dd>This was taken in Gamla Stan (Old Town)
      ▶ in a large square of amazing buildings.</dd>
  </dl>
  <dl class="alt">
    <dt>Gamla Uppsala</dt>
    <dd class="img"></dd>
    <dd>The first three Swedish kings are buried here,
      ▶ under ancient burial mounds.</dd>
  </dl>
  <dl>
    <dt>Perpetual Sun</dt>
    <dd class="img"></dd>
    <dd>During the summer months, the sun takes forever to
      ▶ go down. This is a good thing.</dd>
  </dl>
</div>
```

После добавления ко второму анонсу класса `alt` можно задать несколько правил в конце таблицы стилей. Они будут отменять правила, действующие по умолчанию, которые выравнивают изображение по левому краю, а стиль `alt` будет задавать новое выравнивание. Этот класс при желании можно изменять, что позволяет редактору полностью контролировать макет.

После последних правил добавьте следующий CSS-код:

```
/* изменяем выравнивание */
#sweden .alt dt {
  float: left;
}
#sweden .alt dd.img img {
  float: right;
  margin: 0 0 0 8px;
}
```

Мы даем команду браузеру, чтобы он выполнил следующие действия:

- элементы `<dt>`, расположенные внутри `<dl>` с классом `alt`, должны обтекать изображение слева (а не справа, как это определено по умолчанию);

- изображения с классом `alt` должны выравниваться по правому краю (а не по левому, как определено по умолчанию);
- 8-пиксельный отступ справа заменяется на такой же отступ слева.

На рис. 4.21 показаны результаты добавления этих правил в таблицу стилей. Так как второй анонс относится к классу `alt`, соответствующее изображение смещается к правому краю. Теперь мы можем использовать этот класс, чтобы изменить выравнивание конкретного изображения.

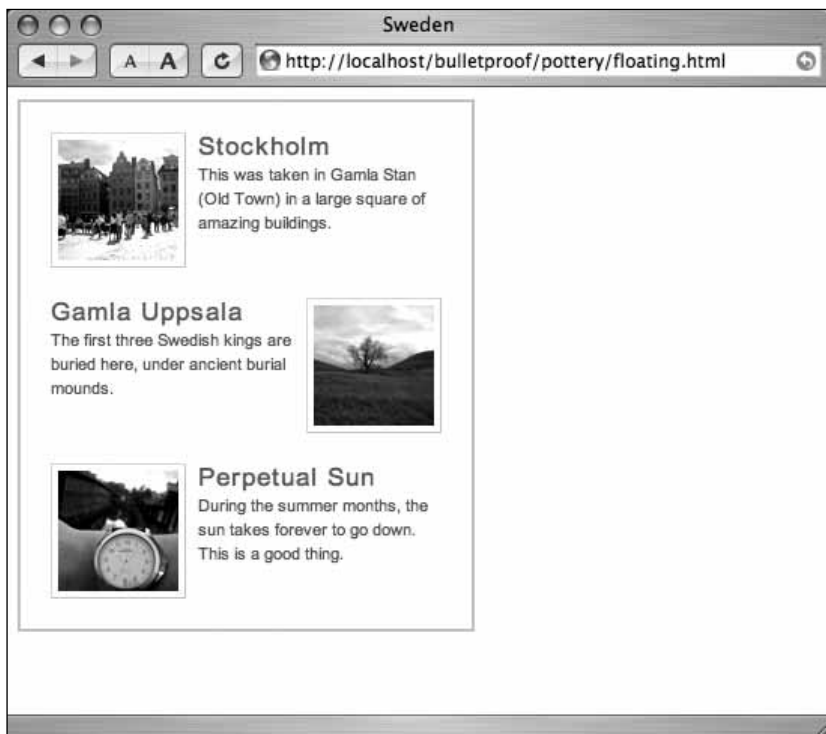


Рис. 4.21. Изменить выравнивание изображения просто — достаточно добавить или удалить класс `alt`

ЭФФЕКТ ТАБЛИЦЫ

Представим, что в нашем примере были использованы длинные аннотации (или пользователь увеличил размер текста), тогда окончание текста описания окажется под изображением снизу. В этом смысл плавающего элемента — он занимает ровно столько места, сколько ему необходимо, и позволяет контенту обтекать его (рис. 4.22).

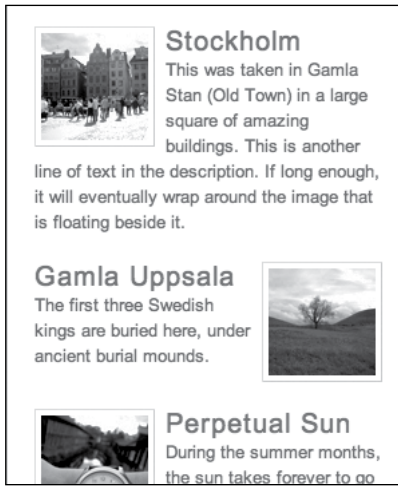


Рис. 4.22. Длинные текстовые блоки обтекают плавающие изображения

В некоторых случаях нас это устраивает, но сейчас нам необходимо создать эффект таблицы. Если добавить отступ к аннотации, тогда можно будет зафиксировать взаимное расположение текста и изображения.

Ширина отступа должна быть равна ширине изображения с учетом дополнительных интервалов между изображением и аннотацией (рис. 4.23).

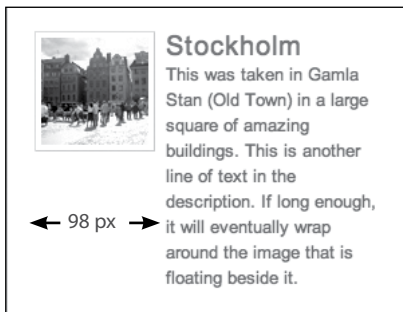


Рис. 4.23. Ширина изображения + отступ + заданное расстояние + ширина рамки = 98 пикселей

Если ко всем элементам `<dd>` добавить слева отступ 98px и сделать этот отступ нулевым для элементов `<dd class="img">` (мы ведь не хотим, чтобы отступ применялся к изображению, находящемуся внутри контейнера `<dd>`), мы создадим эффект таблицы.

Аналогичное изменение отступов для класса `alt`, когда изображение выравнивается по правому краю, можно описать в еще одном правиле, добавив его в раздел альтернативного выравнивания в конце таблицы стилей:

```
#sweden {
    float: left;
    width: 300px;
    padding: 10px 0;
    border: 2px solid #C8CDD2;
}
#sweden dl {
    float: left;
    width: 260px;
    margin: 10px 20px;
    padding: 0;
}
#sweden dt {
    float: right;
    width: 162px;
    margin: 0;
    padding: 0;
    font-size: 130%;
    letter-spacing: 1px;
    color: #627081;
}
#sweden dd {
    margin: 0 0 0 98px;
    padding: 0;
    font-size: 85%;
    line-height: 1.5em;
    color: #666;
}
#sweden dl dd.img {
    margin: 0;
}
#sweden dd.img img {
    float: left;
    margin: 0 8px 0 0;
    padding: 4px;
    border: 1px solid #D9E0E6;
    border-bottom-color: #C8CDD2;
    border-right-color: #C8CDD2;
    background: #FFF;
}
/* reverse float */
#sweden .alt dt {
    float: left;
}
```

```
#sweden .alt dd {
    margin: 0 98px 0 0;
}
#sweden .alt dd.img img {
    float: right;
    margin: 0 0 0 8px;
}
```

Обратите внимание на то, что мы добавили объявление, обнуляющее отступ для элементов `<dd>`, помеченных тегом `class="img"`. Таким образом, мы запрещаем правый отступ при альтернативном выравнивании, описанном в конце таблицы стилей. Данное объявление позволяет не повторять отмену отступа после объявления `#sweden .alt dd`, задающего правый отступ, который не нужен элементам `<dd>` с плавающим изображением.

На рис. 4.24 показаны результаты добавления правил в таблицу стилей. Даже если увеличить кегль текста и (или) использовать объемную аннотацию, текст и изображение останутся в своих «колонок», как будто для создания макета мы использовали таблицу.

АЛЬТЕРНАТИВНЫЙ ФОН

Добавим последний штрих — изменим сплошную голубую рамку вокруг модуля на градиентную фоновую заливку с переходом от голубого к белому цвету. Давайте создадим в Photoshop изображение, ширина которого составит 304 пиксела (300 пикселей + 2-пиксельные границы по обеим сторонам).

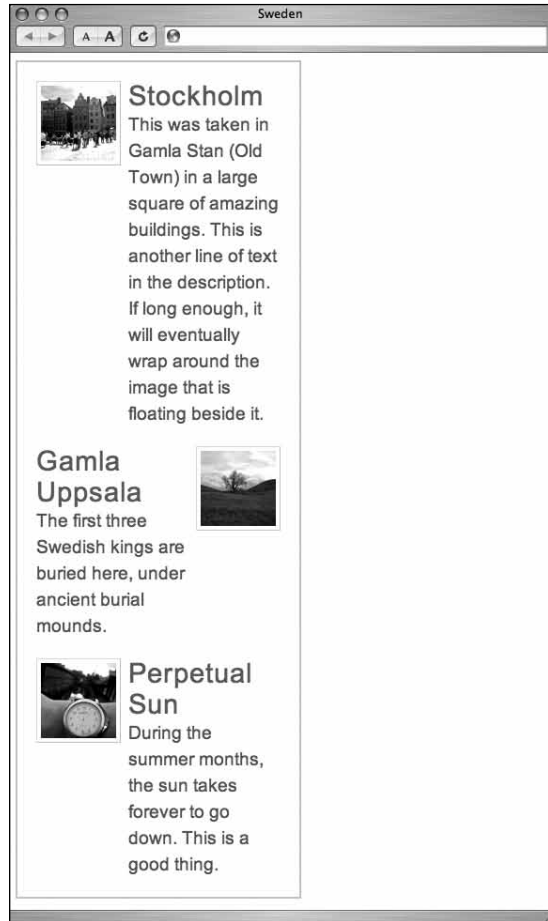


Рис. 4.24. Добавляем отступ к описанию, чтобы выглядело так, как будто текст находится в колонках таблицы

На рис. 4.25 показано окончательное изображение, полученное за счет заливки прямоугольника светло-голубым цветом, и инструмента **Градиент** (рис. 4.26) с переходом от белого к прозрачному фону снизу вверх.



Рис. 4.25. Мы создали голубой фон с помощью градиентной заливки

Для ссылки на изображение в таблице стилей изменим объявление для основного контейнера `<article>`:

```
#sweden {  
    float: left;  
    width: 304px;  
    padding: 10px 0;  
    background: url(img/bg.gif) no-repeat top left;  
}
```

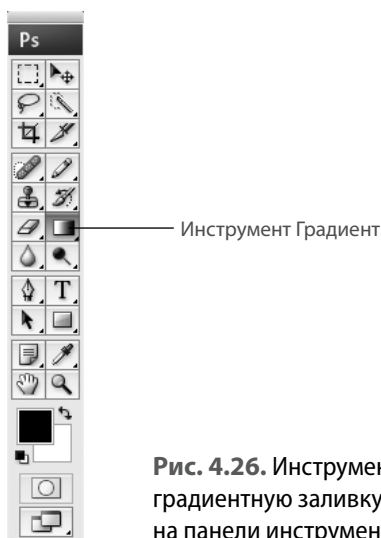


Рис. 4.26. Инструмент, выполняющий градиентную заливку, находится на панели инструментов Photoshop

Мы изменили ширину контейнера с 300 на 304 пиксела, учитывая 2-пиксельную границу (которая сейчас является частью фонового изображения), и выровняли фоновое изображение по верхнему и левому краям. Поскольку заливка постепенно переходит в белый цвет (фоновый цвет страницы) и выровнена по верхней части модуля, нам не нужно думать о заливке при заполнении рамки. Мы сможем работать с контентом любой высоты и всегда будем получать красивый результат (рис. 4.27).



Рис. 4.27. Завершенный пуленепробиваемый пример

Если увеличить верхнее изображение, вокруг изображений станет видна белая рамка (рис. 4.28). Для ее создания мы задали `background: #FFF`; для отступа от плавающих изображений. Если бы мы не указали фоновый цвет, то голубая заливка просвечивала бы сквозь рамку.



Рис. 4.28. Совмещение отступа и фонового цвета позволяет создать рамку вокруг изображения

ЭФФЕКТ ТЕНИ С BOX-SHADOW

Для создания дополнительного эффекта для каждого изображения будем использовать `box-shadow` (CSS3). Свойство `box-shadow` — очень гибкий способ создания теней для элементов, не требующий дополнительной разметки или графических элементов. Свойство `box-shadow` поддерживается последними версиями браузеров, но является необязательным дополнением, от которого можно легко отказаться, если браузер его не поддерживает.

Для добавления тени к изображениям в нашем примере нужно задать несколько правил:

```
#sweden dd.img img {
  float: left;
  margin: 0 8px 0 0;
  padding: 4px;
  border: 1px solid #D9E0E6;
  border-bottom-color: #C8CDD2;
  border-right-color: #C8CDD2;
  background: #FFF;
  -webkit-box-shadow: 1px 1px 3px rgba(0,0,0,.12);
  -moz-box-shadow: 1px 1px 3px rgba(0,0,0,.12);
  box-shadow: 1px 1px 3px rgba(0,0,0,.12);
}
```

Обратите внимание, что сначала мы добавляем `box-shadow` с префиксами разработчиков для браузеров WebKit и Mozilla и в самом конце записываем свойство без префиксов.

Синтаксис `box-shadow` предполагает указание координаты верхней левой точки начала тени относительно элемента (в данном случае 1px сверху и 1px слева). Третье значение указывает браузеру глубину размытки тени (в данном случае 3px). Последним параметром является цвет тени в палитре RGBA, которая позволяет использовать прозрачность, чтобы тень могла сливаться с любым фоном. Для `box-shadow` я практически всегда использую палитру RGBA. В данном случае выбраны значения 0, 0, 0 (RGB-значение черного цвета), а коэффициент непрозрачности — .12 (или 12%).

На рис. 4.29 показаны результаты нашей работы после добавления правил `box-shadow` в браузере Safari. Если браузер не поддерживает `box-shadow` (например, IE версии не старше 8), эти правила игнорируются, и тень не отображается.

Преимущество `box-shadow` для создания теней состоит в том, что для создания теней не нужно отводить дополнительное место в макете. Тени располагаются на отступах или других участках элементов, которые не нужно задавать явным образом в таблице стилей. В старые времена, чтобы задать точку, с которой начинается тень, нужно было использовать отступ, указывать дополнитель-

ную разметку и отводить дополнительное пространство. При использовании `box-shadow` тень сама добавляется там, где это нужно, или не добавляется (если браузер не поддерживает это свойство). Вот что такое абсолютная пуленепробиваемость.



Рис. 4.29. Увеличенный вид изображения с тенью при просмотре в Safari

ЕЩЕ НЕМНОГО ПОРАЗВЛЕКАЕМСЯ С ПЛАВАЮЩИМИ ЭЛЕМЕНТАМИ

Мы закончили работу над примером, используя плавающие элементы для фиксации плавающих элементов — метод, который мы ранее применяли в книге. Все `<dl>` стали плавающими, что позволило очистить расположенные в их недрах плавающие элементы.

Это простой кроссбраузерный метод, но он накладывает на нас определенные обязательства — нам следует задать ширину контейнера (чтобы разместить плавающие контейнеры по вертикали), мы также должны учитывать тип элементов, располагающихся до или после контейнера с плавающими элементами (попытка решить задачу очистки плавающих контейнеров может превратиться в бесконечный бег в беличьем колесе).

Так что использованный метод подходит не для всех ситуаций. Впрочем, существуют и другие способы решения этой проблемы, которые не столь привязаны к конкретным условиям. Давайте рассмотрим и их.

Очистка плавающих элементов с помощью `overflow`

Использование свойства `overflow` позволяет очистить все плавающие элементы внутри контейнера (см. статью Алекса Уокера www.sitepoint.com/blogs/2005/02/26/simple-clearing-of-floats/). Данный подход просто реализовать, и, как это ни странно, он будет работать.

Вернемся к нашему примеру. Если бы мы отказались от очистки плавающих элементов в `<dl>` и использовали свойство `overflow: auto`, то достигли бы этой же цели.

Тогда начальное объявление

```
#sweden dl {
    float: left;
    width: 260px;
    margin: 10px 20px;
    padding: 0;
}
```

примет вид:

```
#sweden dl {
    overflow: auto;
    width: 260px;
    margin: 10px 20px;
    padding: 0;
}
```

В большинстве ситуаций свойство `overflow` будет выполнять нужную нам очистку. Однако в некоторых случаях оно само может стать проблемой. Мы указали значение `auto`, которое используется для создания полос прокрутки вокруг элементов, контент которых превышает заданную ширину (260px), — и это только один из возможных вариантов. Вместо `overflow: auto`; можно написать `overflow: hidden`;. Значение `hidden` скроет контент, если он превысит ширину контейнера.

Если вы уверены, что ни одна из перечисленных ситуаций точно не произойдет, тогда `overflow` — идеальное решение. Но я хочу быть уверен в том, что проблем в будущем не возникнет. И тут нам поможет следующий метод.

ПРОСТАЯ ОЧИСТКА С ПОМОЩЬЮ ГЕНЕРИРОВАННОГО КОНТЕНТА

Самое надежное решение я нашел на сайте *Position Is Everything* (<http://positioniseverything.net/easyclearing.html>). Идея гениальна в своей простоте: использовать псевдоэлемент `:after` (CSS) (www.w3c.org/TR/CSS21/selector.html#before-and-after) для добавления точки после контейнера с плавающими элементами. Правило `clear: both` очищает плавающие элементы, а точка становится невидимой. Код выглядит следующим образом:

```
#sweden dl {
    width: 260px;
    margin: 10px 20px;
    padding: 0;
}
```

```
#sweden dl:after {  
  content: ".";  
  display: block;  
  height: 0;  
  clear: both;  
  visibility: hidden;  
}
```

Теперь нам не нужно добавлять `float` или `overflow` в основное объявление `#sweden dl`, как мы это делали в предыдущих случаях. Очистку выполняет второе объявление. Давайте разберемся, как все происходит.

Правило дает браузеру инструкции: «Добавить точку после каждого списка определений (`content: ".";`), очистить все предшествующие плавающие элементы (`clear: both;`) и сделать точку невидимой (`height: 0;` и `visibility: hidden;`)». Весьма изобретательно.

Используя псевдоэлемент `:after`, мы очистили плавающие элементы, не делая контейнер плавающим. Кроме того, теперь мы можем быть уверены, что в будущем `overflow` не доставит нам хлопот.

Эх, если бы все это работало в Internet Explorer! Догадались? IE6 и 7 не поддерживают `:after` (как и `:before`). К счастью, есть другие правила, которые позволяют реализовать самоочистку элементов в этих версиях IE. Хорошая новость — IE8+ поддерживает `:after` и `:before`, поэтому в будущем не понадобится прибегать к дополнительным уловкам.

Самоочистка в IE6

Традиционно IE/Win выполняет самоочистку контейнеров, только если для них были указаны размеры. Конечно, это неправильно, но мы извлечем пользу из этой ошибки. Трюк получил название Holly Hack (по имени его создателя — Холи Берджвина). Благодаря ему вы можете задать свойство `height: 1%;`, которое работает только в IE. Свойство заставляет контейнер расширяться, охватывая контент (даже плавающие элементы). Если высота контента превышает 1% (а это происходит практически всегда), то все сработает как надо, игнорируя правило `height`. Хотя это и не совсем правильно с точки зрения спецификации, но ошибка помогает нам решить проблему с самоочисткой.

СОВЕТ

Забавно, но в IE определение `height: 1%` вызывает так называемую процедуру `hasLayout`. Почитайте хорошую (но длинную) статью *On Having Layout* (www.satzansatz.de/cssd/onhavinglayout.html), если вам интересно, как все работает. Только позаботьтесь о достаточном количестве кофе перед чтением.

Поставим перед объявлением селектор `* html`, который понимает только IE/Win версии 6 (IE7 мы займемся через минуту):

```
* html #sweden dl { height: 1%; }
```

Одно-единственное объявление очистит все плавающие элементы в списке определений. Используя `* html`, вы ограничиваете область действия объявления IE6. Другие браузеры проигнорируют этот текст, так как для них он не имеет смысла, ведь до `<html>` элементы определять невозможно.

Использование `:after` в современных браузерах, которые распознают его (Mozilla, Firefox, Safari и т. п.), и Holly Hack (для IE6) позволяют создать надежный способ самоочистки, работающий в большинстве браузеров. В идеале нужно разместить Holly Hack (и другие характерные только для IE CSS-коды) в отдельной таблице стилей (мы поговорим об этом далее, в главе 9 «Сводим воедино»).

Решаем проблему IE7

Браузер IE7 выпустили с расширенной поддержкой стандартов, многие ошибки были исправлены, за что мы благодарны разработчикам. Тем не менее он все еще не поддерживает функции, которые давно используются Firefox, Safari и Opera. И это заставляет нас прибегать к изощренным способам реализации дизайнерских замыслов. Описываемый здесь метод простой очистки — один из них.

В IE7 исправлена ошибка `height: 1%;`. IE6 расширял контейнер для «охвата» контента, даже когда контент выходил за пределы границ, определенных в CSS, а IE7 корректно обрабатывает эти значения. Мы не можем винить разработчиков IE7 за то, что браузер работает правильно. Кроме того, как и другие современные браузеры, IE7 игнорирует объявления селектора `* html`, что также вполне естественно.

Проблема в том, что IE7 почему-то не поддерживает псевдоэлемент `:after`. Исправив обработку `height: 1%` без поддержки `:after`, разработчики IE7 лишили нас возможности использовать предыдущий метод самоочистки плавающих элементов.

Для решения этой задачи используем фрагмент кода:

```
*:first-child+html #sweden dl { min-height: 1px; }
```

Свойство `min-height` в IE7 позволяет расширить контейнер так же, как свойство `height` в IE6. Столь странный селектор, использованный перед `#sweden dl` (`*:first-child+html`), — будет «понятен» только IE7, и больше никому.

Ну вот, теперь все на месте и можно очищать плавающие элементы в популярных браузерах. Посмотрим на фрагмент кода в окончательном виде:

```
#sweden dl:after { /* для браузеров, поддерживающих :after */
    content: " .";
```

```

display: block;
height: 0;
clear: both;
visibility: hidden;
}
* html #sweden dl { height: 1%; } /* для IE5+6 */
*:first-child+html #sweden dl { min-height: 1px; } /* для IE7 */

```

В идеале этот код, «понятный» только IE, необходимо размещать в отдельной таблице стилей, чтобы не «загрязнять» чистый код (более подробно об этом говорится в главе 9 «Сводим воедино»). В то же время эти три объявления позволяют нам надежно очищать плавающие элементы. На первый взгляд код выглядит довольно сложным. Однако если вы начнете пользоваться им постоянно, то увидите, что он незаменим для придания гибкости. Теперь можно не опасаться, что плавающие элементы станут причиной каких-либо проблем даже в отдаленной перспективе.

Объединим селекторы, чтобы обезопасить код

Давайте объединим селекторы, используемые для самоочистки, в одном объявлении, чтобы не повторять код для каждого контейнера. Допустим, что нам нужно очистить плавающие элементы, расположенные в заголовке документа:

```

#header:after,
#sweden dl:after { /* для браузеров, поддерживающих :after */
    content: ".";
    display: block;
    height: 0;
    clear: both;
    visibility: hidden;
}

* html #header,
* html #sweden dl { height: 1%; } /* для IE6 */
*:first-child+html #header,
*:first-child+html #sweden dl { min-height: 1px; } /* для IE7 */

```

По мере работы над макетом просто добавляйте новые элементы в эти фрагменты кода.

Выбираем лучшее

Мы исследовали три метода очистки. Не забывайте, что (как и в большинстве случаев) не существует единственного способа, правильного всегда и везде. Эксперименты с кодом позволяют понять, какой метод работает лучше, а какой хуже в конкретных ситуациях.

Я люблю простой метод очистки за его надежность и работоспособность в любой ситуации, но часто использую `overflow: hidden;` для контейнеров, изначально

внедренных в макет, что облегчает задачу разработки макета. На более поздних этапах мне будет достаточно найти соответствующее правило и добавить три объявления, выполняющих простой метод очистки.

Какой бы метод вы ни использовали, важно помнить, что самоочистка — мощный инструмент. Плавающие элементы делают макет максимально гибким, сохраняя независимость компонентов страницы.

ПРЕИМУЩЕСТВА ПУЛЕНЕПРОБИВАЕМОГО ПОДХОДА

После всех этих долгих экспериментов вы, возможно, уже забыли, зачем мы занялись блоком из изображения, заголовка и аннотации. Мы нашли оптимальную структуру разметки, вместо того чтобы использовать таблицы и GIF-разделители. Именно поэтому у нас получился компактный и достаточно гибкий результат — гибкий с точки зрения обновления, добавления или удаления контента. Кроме того, простой код упрощает работу браузеров, приложений и облегчает жизнь редакторам сайта.

Такой подход дает огромное преимущество для сайтов, работающих с базами данных, где единообразная и предсказуемая структура разметки может быть использована многократно. Вам не нужно думать о том, располагается ли изображение до названия и описания в разметке (как при работе с таблицами), ведь полученная нами структура станет стабильной и будет легко отображаться даже в динамических шаблонах.

Благодаря использованию CSS для создания структур, аналогичных таблицам, мы смогли отказаться от кода и графического отображения названия в информационном блоке. Это обеспечило нам достаточную гибкость при размещении изображений и изменении размера названия/описания, стиля и контента.

РЕЗЮМЕ

Надеюсь, что рассмотренный пример поможет вам научиться эффективно использовать плавающие элементы для упорядочивания данных. Если вы поймете, как корректно очищать плавающие элементы, то сможете использовать их дополнительные возможности, не добавляя лишней разметки. Конечно, в некоторых случаях использование `<table>` неизбежно (для табличных данных). Не забывайте, что от стремления к компактной и понятной разметке не должен страдать дизайн. Мы можем использовать разметку, чтобы сделать нашу работу «понятной» большому диапазону устройств и приложений и решить задачи дизайна с помощью CSS.

Запомните следующее:

- обдумывайте оптимальную разметку заранее. Вы всегда сможете изменить ее, но лучше придерживаться компактной и понятной структуры с самого начала;
- выравнивайте плавающие элементы по сторонам, независимо от того, где они находятся в структуре документа. В нашем примере название находится перед изображением, нам было нужно расположить элементы в соседних колонках, и наш метод помог решить эту задачу;
- помните об очистке плавающих элементов: проверяйте свою работу при разном объеме контента и размере текста, чтобы гарантировать целостность макета;
- используйте свойство `box-shadow` для создания теней. Избегайте излишней разметки, изображений и корректировок макета;
- экспериментируйте с разными методами самоочистки плавающих элементов, чтобы обеспечить гибкость макета и независимость компонентов.

ГЛАВА 5

НЕСОКРУШИМЫЕ ЭЛЕМЕНТЫ



Несмотря на то что основой CSS является блочная верстка (box-модель), это не означает, что дизайн должен быть ограничен прямоугольниками. Поскольку box-модель CSS помещает каждый структурный элемент документа в прямоугольник (www.w3.org/TR/REC-CSS2/box.html), именно эта геометрическая фигура стала стандартным методом упорядочивания элементов в CSS. Естественно, это привело к тому, что «прямоугольный» дизайн стал нормой.

Ничего нового, табличные формы предлагали такой же блочный дизайн еще до того, как CSS вошел в моду, поскольку каждая ячейка таблицы имеет форму прямоугольника.

В данной главе мы рассмотрим несколько методов создания прямоугольных элементов со скругленными углами, используя фоновые изображения и (или) CSS. Мы возьмем реальный пример оформления, взятого из Сети, и реализуем его по-новому — сделаем его гибким и способным работать с любым контентом. После этого мы займемся альтернативными методами создания истинно гибких (в любом смысле слова) блоков со скругленными углами. А в конце рассмотрим простые примеры того, как можно модифицировать обычные прямоугольные элементы. Методы, которые CSS3 предлагает нам для оформления, идеальны с точки зрения гибкости и обслуживания. Кроме того, они позволяют отказаться от растровых изображений. Тем не менее мы обсудим несколько вариантов использования «картинок», — ведь ваш начальник или клиент хочет, чтобы сайт имел тот же вид в новых и старых версиях браузеров.

ОБЩЕПРИНЯТЫЙ ПОДХОД

Как и в предыдущих примерах, давайте сначала проанализируем общепринятый подход к разработке блоков со скругленными углами и трансформируем его с помощью CSS. Для данного примера я решил использовать старый дизайн версии раздела **Your Account** (Учетная запись) популярного портала Netflix (рис. 5.1).

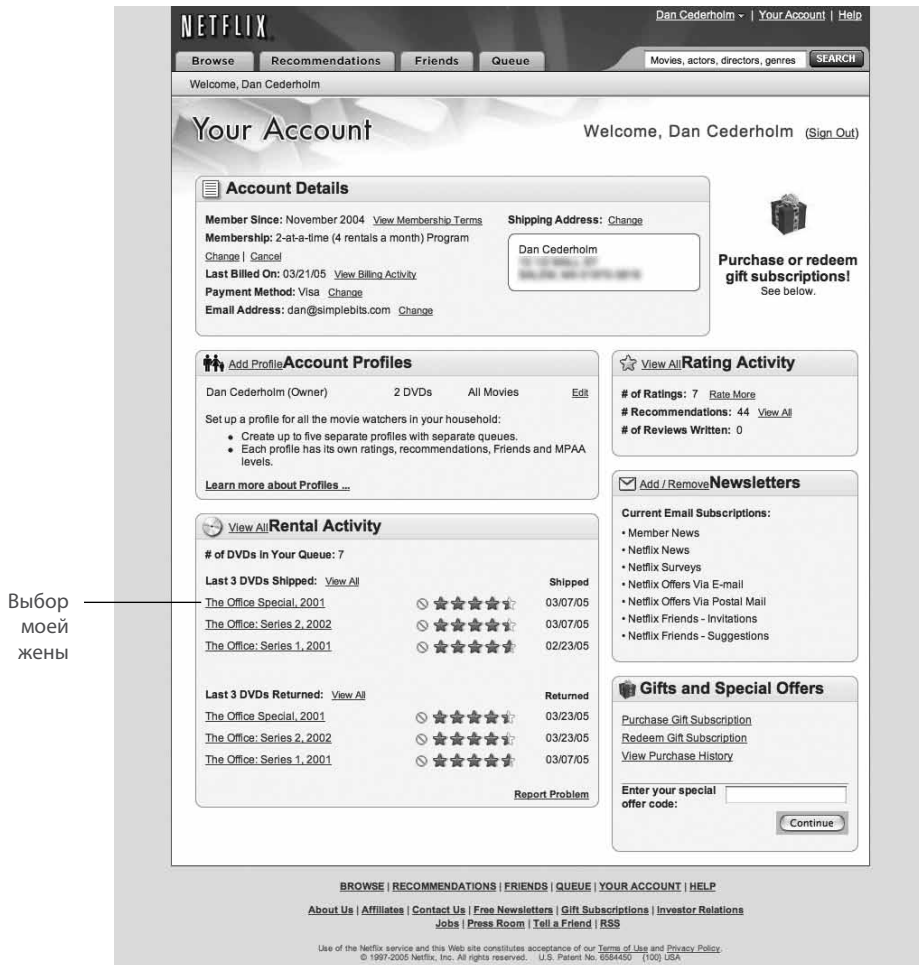


Рис. 5.1. Страница My Account сайта Netflix.com (апрель 2005 года)

Прямоугольные элементы с закругленными углами — стандартный элемент оформления на сайте Netflix и множестве других сайтов. Скругленные углы добавляют изюминку в дизайн, который без них выглядит несколько угловатым (рис. 5.2).

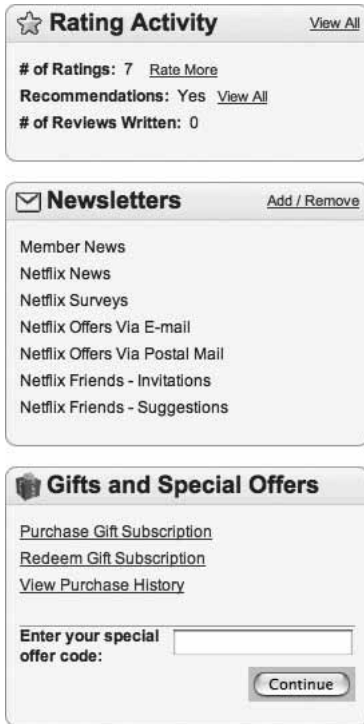


Рис. 5.2. Прямоугольные элементы со скругленными углами, которые используются на странице Netflix, довольно популярны

Возможно, вы уже знаете все, что я скажу дальше. На самом деле блоки Netflix созданы с помощью вложенных таблиц и GIF-изображений, на которые ссылается разметка. Такое оформление было создано много лет назад, поэтому это простительно.

Давайте переделаем эти блоки. Для этого увеличим и рассмотрим компоненты дизайна, чтобы сразу отметить, что нам потребуется. На рис. 5.3 я удалил контент, чтобы продемонстрировать, из каких графических компонентов состоит прямоугольный элемент.



Рис. 5.3. Давайте удалим контент, чтобы тщательнее проанализировать оформление

Посмотрите на верхнюю часть блока — углы скруглены. В качестве фона используется градиентная заливка, заполняющая всю верхнюю область до линии, отделяющей область заголовка (рис. 5.4).

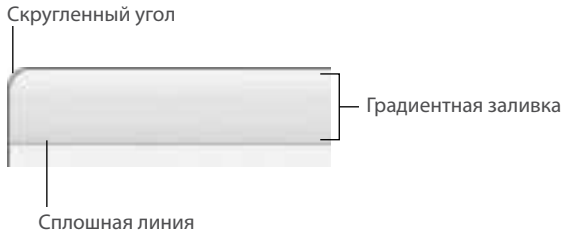


Рис. 5.4. Увеличив масштаб, мы заметим градиентную заливку, заполняющую всю область заголовка сверху вниз

Аналогично, внизу элемента углы скруглены с обеих сторон, но градиентная заливка не используется (рис. 5.5).

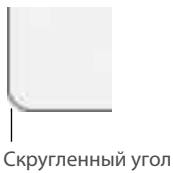


Рис. 5.5. Скругленные углы в нижней части элемента без градиентной заливки

Уязвимые места

По многим причинам, которые я уже приводил в предыдущих главах, такие прямоугольные элементы со скругленными углами — это всегда проблема. Во-первых, вложенные таблицы порождают огромное количество презентационной разметки. Мы уже знаем, как легко можно решить этот вопрос с помощью CSS, при этом останется только значимая разметка, структурирующая контент.

Во-вторых, дизайн не предусматривает использование объемного контента или заголовков, длина которых превышает фиксированное количество символов. На рис. 5.6 показано, что произойдет с прямоугольным блоком из нашего примера, если увеличить размер заголовка.

Хотя заголовочный элемент может увеличивать свои размеры по вертикали, фоновое изображение рассчитано на одну строку с определенным количеством символов и размером шрифта. Если строка окажется длиннее или пользователь увеличит размер шрифта, целостность дизайна нарушится.

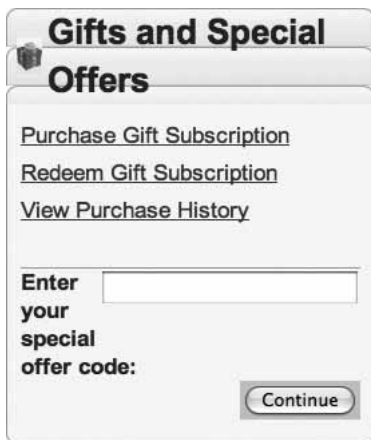


Рис. 5.6. При увеличении шрифта фоновое изображение дублируется, но такой фон рассчитан только на одну строку заголовка

Изображение, размещенное позади заголовка элемента, является одновременно фоновой иллюстрацией элемента `<td>`. В терминах разметки это записывается как:

```
<td background="/images/box_top.gif">
```

По умолчанию заданное таким образом фоновое изображение будет повторяться в ячейке. Дублирование изображения неизбежно, если высота увеличенного текста превысит высоту изображения.

Понятно, что это не криминал, но данную особенность необходимо учесть.

ПУЛЕНЕПРОБИВАЕМЫЙ ПОДХОД

Для того чтобы сделать прямоугольный блок гибким и убрать избыточный код, переделаем оформление так, чтобы сократить количество разметки и использовать уникальные возможности CSS для создания элементов дизайна.

Давайте сначала подумаем, как обеспечить гибкость всех элементов прямоугольного блока по вертикали. Градиентная заливка в заголовке элемента должна реагировать на изменение объема и кегля текста. Оставшаяся (нижняя) часть блока также должна менять свои вертикальные размеры в зависимости от содержимого. При этом нижние скругленные углы располагаются под всем контентом (рис. 5.7).

ПРИМЕЧАНИЕ

Для данного примера нам понадобится прямоугольный элемент с фиксированной шириной, но изменяющейся высотой. Но можно создать прямоугольный блок со скругленными углами, который будет менять ширину. Об этом способе мы поговорим позднее.

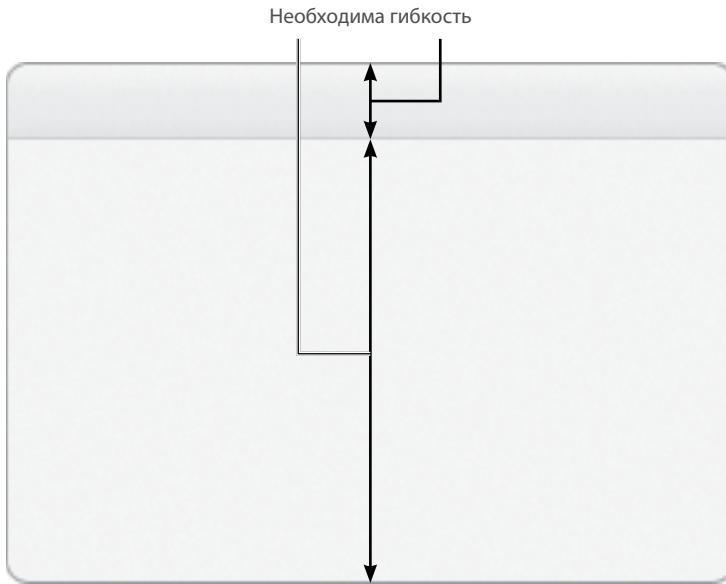


Рис. 5.7. Обе области прямоугольного блока должны быть подвижными по вертикали

Для реализации нашего плана понадобится хитрая работа с фоновым изображением и возможности CSS. Позднее мы займемся воспроизведением этого же дизайна с помощью CSS3, т. е. без изображений, но сейчас мы начнем с компактной структуры разметки.

СТРУКТУРА РАЗМЕТКИ

Перед тем как структурировать прямоугольный блок со скруглениями, вспомним наши технические условия: фиксированная ширина, заголовок расположен вверху, изменяемое количество контента в нижней части. Мне кажется, что такой блок стоит поместить в контейнер `<div>` (или элемент `<section>`), под которым будет располагаться заголовок блока, а под ним — любой контент. В данном примере я упрощу задачу — буду использовать контент в виде списка из трех ссылок (поскольку сейчас речь идет об оформлении скруглений в прямоугольных элементах).

Поскольку блоки встречаются на странице много раз, нам понадобится стиль, который можно будет применять к большому количеству элементов. Так что для разметки `<div>` мы возьмем `class` (для многократного использования), а не `id` (подходит для однократного использования), в котором будет задан стиль прямоугольных блоков:

```
<div class="box">
  <h3>Gifts and Special Offers</h3>
  <ul>
    <li><a href="/purchase/">Purchase Gift Subscription</a></li>
    <li><a href="/redeem/">Redeem Gift Subscription</a></li>
    <li><a href="/view/">View Purchase History</a></li>
  </ul>
</div>
```

Код довольно прост: используется только `<div>` с указанием класса `class="box"`, заголовок третьего уровня и нумерованный список из трех ссылок. Такая разметка будет оптимальной.

Внешний `<div>` позволяет задать фиксированную ширину для элемента и установить флаг, чтобы задать стили заголовка и контента.

РАБОТАЕМ С ИЗОБРАЖЕНИЯМИ

Самый сложный и важный момент в данном примере — разделение графических элементов и дизайна, чтобы гарантировать обработку контента любого объема и шрифта любого кегля. Давайте выберем высоту изображения, которая будет заведомо больше размера, занимаемого как заголовком, так и основным текстовым блоком.

СОВЕТ

Данный способ навеян методом «раздвижные двери» Дугласа Боумана (о котором шла речь в главе 2 «Навигация»). Можно использовать довольно большие изображения, на которых точно уместится контент максимального объема, а CSS будет управлять размером изображения в зависимости от контента. Данный метод имеет определенные ограничения, например размеры или объем контента могут превысить величину изображений. В идеале было бы найти золотую середину: изображения должны быть достаточно большими, чтобы вместить контент, но не настолько, чтобы пользователи с медленным каналом испытали проблемы при загрузке.

CSS3 работает с несколькими фоновыми изображениями, что сильно упрощает метод «раздвижных дверей». Мы можем отказаться от лишней разметки для вставки дополнительных фоновых изображений. Зайдите на страничку <http://people.opera.com/patrickl/experiments/css3/sliding-doors/>, чтобы познакомиться с этой возможностью.

Давайте начнем с заголовка, для него понадобится изображение, на котором можно разместить заголовок из нескольких строк, даже с увеличенным размером шрифта. Поскольку это изображение будет фоном заголовка `<h3>` (размеры

изображения по вертикали подстроится под размер заголовка), в его верхней части должны присутствовать скругленные углы.

На рис. 5.8 показано созданное мною изображение (`h3-bg.gif`). Его высота позволит вместить несколько строк заголовка. Обратите внимание, что я не стал оформлять нижнюю границу, которая отделяет заголовок от основного поля. Для создания этой границы мы воспользуемся возможностями CSS. Данный подход позволяет выровнять изображение по верхней границе блока, оформив скругление верхних углов и обеспечив поле переменного размера для заголовка. Под заголовком будет располагаться 1-пиксельная граница, созданная с помощью CSS.

Второе изображение будет создавать скругленные углы в нижней части блока и 1-пиксельную рамку по его границам. Как и в случае с фоном заголовка, нам понадобится дополнительное пространство по вертикали, чтобы разместить контент переменной длины, который потенциально может попасть в этот блок.

На рис. 5.9 показан окончательный вид этого изображения (`div-bottom.gif`). Его высота учитывает размер контента, который может быть помещен в этот блок.

После создания изображения пора перейти к CSS, чтобы объединить дизайн и контент воедино.



Рис. 5.8. Фон для заголовка элемента имеет достаточную высоту для размещения нескольких строк текста



Рис. 5.9. Фоновое изображение имеет достаточную высоту для объемного контента

Стили

После структурирования и создания расширяемых изображений пора приступить к сборке «мозаики» при помощи CSS.

В нашем примере используется базовый размер текста `small`, а в свойстве `font-family` по умолчанию указан шрифт `Arial`. Сначала зададим ширину для прямоугольного элемента (а значит, его контента):

```
.box {  
    width: 273px;  
}
```

Сначала мы вставим фоновое изображение `div-bottom.gif` в контейнер `<div>`. Это гарантирует нам, что скругленные углы окажутся ниже любого контента.

```
.box {  
    width: 273px;  
    background: url(img/div-bottom.gif) no-repeat bottom left;  
}
```

СОВЕТ

Я использую `class` для объявления стилей в прямоугольном блоке, поскольку мы будем использовать такое оформление для нескольких элементов на странице. Свойство `class` необходимо, когда на странице присутствует несколько однотипных элементов, в то время как `id` подходит только для однократного использования.

Выравнивая изображение по нижнему краю, мы располагаем скругленные углы в нужном месте, т. е. под контентом.

На рис. 5.10 показаны результаты нашей работы после задания ширины и вставки одного фонового изображения.

Gifts and Special Offers

- [Purchase Gift Subscription](#)
- [Redeem Gift Subscription](#)
- [View Purchase History](#)

Рис. 5.10. Так как фоновое изображение выровнено по нижнему краю, то верхняя часть изображения не будет отображаться до тех пор, пока размер контента не увеличится

Теперь добавим фоновое изображение для заголовка, выравнивая его по верхнему краю, чтобы верхние углы были скруглены.

```
.box {
  width: 273px;
  background: url(img/div-bottom.gif) no-repeat bottom left;
}
.box h3 {
  background: url(img/h3-bg.gif) no-repeat top left;
}
```

На рис. 5.11 показан результат после вставки фонового изображения заголовка <h3>. Создание скругленной рамки блока завершено. Теперь мы добавим 1-пиксельную границу, отделяющую заголовок, определим поля и отступы и изменим шрифт.

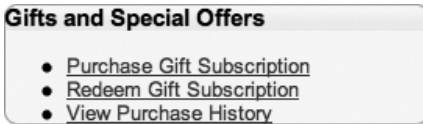


Рис. 5.11. Фоновое изображение заголовка завершает процесс скругления углов со всех сторон

```
.box {
  width: 273px;
  background: url(img/div-bottom.gif) no-repeat bottom left;
}
.box h3 {
  margin: 0;
  padding: 6px 8px 4px 10px;
  font-size: 130%;
  color: #333;
  border-bottom: 1px solid #E0CFAB;
  background: url(img/h3-bg.gif) no-repeat top left;
}
```

Задав отступы <h3>, размер и цвет шрифта заголовка, мы оставляем пространство для отображения фона, который выравнивается по верху и левому краю нашего бокса. Граница `border-bottom` будет располагаться под текстом заголовка, а сам фон будет виден лишь частично (рис. 5.12).

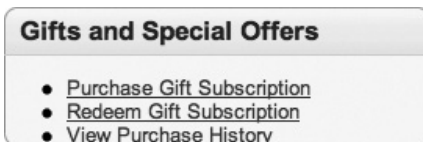


Рис. 5.12. Фон заголовка и граница позволяют заголовку изменяться, отображая необходимый фрагмент

Нам осталось задать отступы маркированного списка и удалить сами маркеры.

```
.box {
  width: 273px;
  background: url(img/div-bottom.gif) no-repeat bottom left;
}
.box h3 {
  margin: 0;
  padding: 6px 8px 4px 10px;
  font-size: 130%;
  color: #333;
  border-bottom: 1px solid #E0CFAB;
  background: url(img/h3-bg.gif) no-repeat top left;
}
.box ul {
  margin: 0;
  padding: 14px 10px 14px 10px;
  list-style: none;
}
.box ul li {
  margin: 0 0 6px;
  padding: 0;
}
```

После обнуления полей и отступов (с помощью свойства `list-style`) и задания отступов для маркированного списка мы завершили дизайн прямоугольного блока (рис. 5.13). Не так уж и сложно, не так ли?

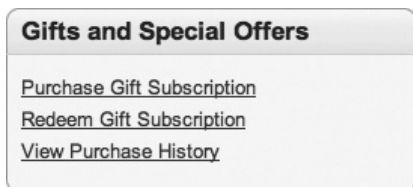


Рис. 5.13. Итоговый пуленепробиваемый прямоугольный блок

ПРЕИМУЩЕСТВА ПУЛЕНЕПРОБИВАЕМОГО ПОДХОДА

Если объединить весь код, необходимый для создания этого блока, то его объем будет существенно меньше кода, необходимого для реализации вложенных таблиц. Так как мы продумали структуру разметки, можно не сомневаться, что наши блоки будут корректно отображаться большинством браузеров и устройств. Меньший объем кода (точнее, использование более понятного кода) делает вашу реализацию более быстрой, эффективной, доступной и читаемой на большинстве устройств — от телефона до настольного компьютера.

Помимо сокращения объема кода, преимуществом нашего дизайна прямоугольного элемента является его «растяжимость» по вертикали. Так как мы используем эффективную стратегию размещения фоновых изображений, увеличение кегля шрифта или добавление нескольких строк текста никак не повлияет на целостность дизайна.

На рис. 5.14 показано, как меняется внешний вид блока при увеличении размера шрифта. Именно здесь проявляется его несокрушимый и пуленепробиваемый характер. Переменное количество контента? Нет проблем! По мере увеличения размера контента будет видна большая часть фонового изображения. Так как нижние углы выровнены по нижнему краю контейнера `<div>`, содержащего весь элемент, то фон всегда будет прижат к нижнему краю, а верхняя его часть будет отображаться при необходимости. Все сказанное справедливо и для фона заголовка, так как верхний край фона заголовка выровнен по верхнему краю блока, нижний край этого изображения будет проступать по мере увеличения размера или количества текста.

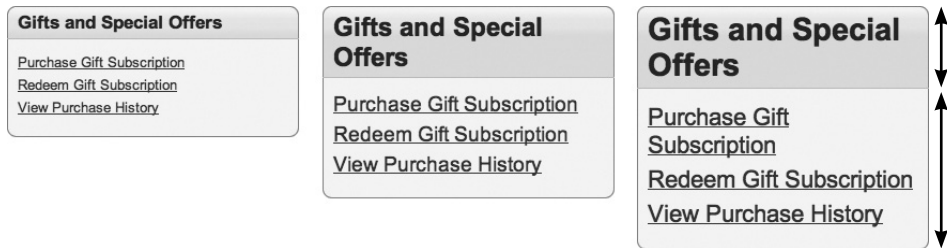


Рис. 5.14. Заголовок элемента и поле контента могут растягиваться по вертикали, подстраиваясь под размер текста

Используем CSS3

В качестве альтернативы фоновым изображениям давайте воссоздадим дизайн прямоугольного блока с сайта Netflix, используя возможности CSS3, т. е. без изображений. Благодаря CSS3 мы получаем следующие преимущества:

- прямоугольный элемент станет по-настоящему гибким — никаких ограничений по ширине или высоте;
- все цвета (границы, фоны, градиенты) можно редактировать в таблице стилей, при необходимости легко изменить оформление.

Использование `border-radius` и градиентов CSS3 для отказа от фоновых изображений, которые мы ранее использовали, потребует небольшого изменения кода. Для этого необходимо скорректировать объявления `.box` и `.box h3`.

Во-первых, удалим изображения, задав сплошную фоновую заливку и 1-пиксельную границу для элемента `.box`. Также удалим фоновое изображение, создающее скругленные углы вверху и градиент для заголовка.

```
.box {
  width: 273px;
  border: 1px solid #C6A765;
  background: #FAF7F1;
}
.box h3 {
  margin: 0;
  padding: 6px 8px 4px 10px;
  font-size: 130%;
  color: #333;
  border-bottom: 1px solid #E0CFAB;
}
```

На рис. 5.15 показаны результаты изменения. Теперь у нас есть лишь прямоугольный блок с рамкой.

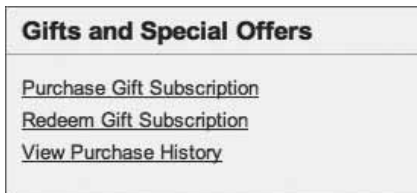


Рис. 5.15. В таком виде прямоугольный блок будет отображаться в браузерах, не поддерживающих CSS3

Затем добавим свойство `border-radius`, которое позволит скруглить углы прямоугольного блока.

```
.box {
  width: 273px;
  border: 1px solid #C6A765;
  background: #FAF7f1;
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  border-radius: 10px;
}
```

Мы задаем радиус скругления 10px, добавляя правило `border-radius` с префиксами разработчиков, и оставляем предыдущий код без префиксов, как делали это раньше. На рис. 5.16 показаны изменения в браузере Safari.

Осталось воссоздать градиентный фон заголовка. Ранее мы использовали изображение, однако сейчас снова можно полностью положиться на CSS3.

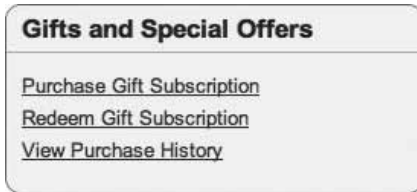


Рис. 5.16. Задание border-radius для скругления углов прямоугольного элемента

Помните, как в главе 2 мы использовали ColorZilla's CSS Gradient Generator (www.colorzilla.com/gradient-editor), чтобы сгенерировать правила для создания градиентов в CSS во всех типах браузеров? Сейчас мы сделаем то же самое, используя цвета, взятые из фонового изображения заголовка (рис. 5.17).



Рис. 5.17. Увеличенный фрагмент градиентной заливки заголовка

После указания правильных начального и конечного значений градиента в ColorZilla можно добавить правила в объявление `.box h3`.

```
.box h3 {
    margin: 0;
    padding: 6px 8px 4px 10px;
    font-size: 130%;
    color: #333;
    border-bottom: 1px solid #E0CFAB;
    background: rgb(250,247,241); /* Старые браузеры */
    background: -moz-linear-gradient(top, rgba(250,247,241,1) 0%,
    ▶ rgba(237,228,208,1) 100%); /* FF3.6+ */
    background: -webkit-gradient(linear, left top, left bottom,
    ▶ color-stop(0%,rgba(250,247,241,1)),
    ▶ color-stop(100%,rgba(237,228,208,1)));
    ▶ /* Chrome,Safari4+ */
    background: -webkit-linear-gradient(top, rgba(250,247,241,1)
    ▶ 0%,rgba(237,228,208,1) 100%); /* Chrome10+,Safari5.1+ */
    background: -o-linear-gradient(top, rgba(250,247,241,1)
    ▶ 0%,rgba(237,228,208,1)
    ▶ 100%); /* Opera11.10+ */
}
```



```

↖
background: -ms-linear-gradient(top, rgba(250,247,241,1)
  ▶ 0%,rgba(237,228,208,1) 100%); /* IE10+ */
filter: progid:DXImageTransform.Microsoft.gradient(startColorstr=
  ▶ '#FAF7F1',
  ▶ endColorstr='#EDE4D0', GradientType=0 ); /* IE6-9 */
background: linear-gradient(top, rgba(250,247,241,1)
  ▶ 0%,rgba(237,228,208,1)
  ▶ 100%); /* W3C */
}

```

На рис. 5.18 показаны результаты после добавления правил для заголовка в браузере Safari. Мы уже близки к завершению, но осталось подправить одну мелочь. Несмотря на то что сам контейнер прямоугольного элемента имеет скругленные углы, у фонового градиента заголовка они остаются прямоугольным, из-за чего верхний левый и правый углы «торчат» вверх. Эта ошибка проявляется не во всех браузерах, так как по стандарту скругленные углы должны обрезать все лишнее, включая градиентный фон.

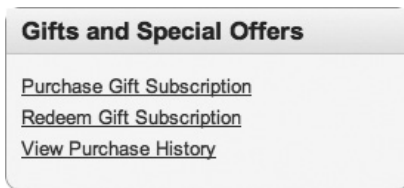


Рис. 5.18. Градиентный фон заголовка перекрывает скругленные углы .box

Добавим правила `border-radius` в объявление заголовка, что позволит скруглить верхние углы. Теперь мы должны получить полностью скругленную верхнюю часть модуля.

```

.box h3 {
  margin: 0;
  padding: 6px 8px 4px 10px;
  font-size: 130%;
  color: #333;
  border-bottom: 1px solid #E0CFAB;
  background: rgb(250,247,241); /* Старые браузеры */
  background: -moz-linear-gradient(top, rgba(250,247,241,1) 0%,
    ▶ rgba(237,228,208,1) 100%); /* FF3.6+ */
  background: -webkit-gradient(linear, left top, left bottom,
    ▶ color-stop(0%,rgba(250,247,241,1)), color-stop
    ▶ (100%,rgba(237,228,208,1))); /* Chrome,Safari4+ */
}

```

```

background: -webkit-linear-gradient(top, rgba(250,247,241,1)
  ▶ 0%,rgba(237,228,208,1) 100%); /* Chrome10+,Safari5.1+ */
background: -o-linear-gradient(top, rgba(250,247,241,1) 0%,
  ▶ rgba(237,228,208,1) 100%); /* Opera11.10+ */
background: -ms-linear-gradient(top, rgba(250,247,241,1) 0%,
  ▶ rgba(237,228,208,1) 100%); /* IE10+ */
filter: progid:DXImageTransform.Microsoft.gradient (
  ▶ startColorstr='#FAF7F1', endColorstr='#EDE4D0', GradientType=0 );
  ▶ /* IE6-9 */
background: linear-gradient(top, rgba(250,247,241,1)
  ▶ 0%,rgba(237,228,208,1)
  ▶ 100%); /* W3C */
-webkit-border-top-left-radius: 10px;
-webkit-border-top-right-radius: 10px;
-moz-border-radius-topleft: 10px;
-moz-border-radius-topright: 10px;
border-top-left-radius: 10px;
border-top-right-radius: 10px;
}

```

На рис. 5.19 показан итоговый вид нашего блока в Safari, со скругленными углами и градиентом, созданными с помощью CSS3.

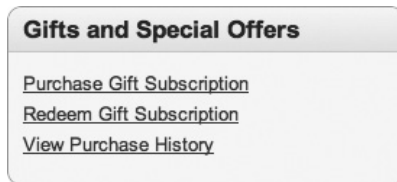


Рис. 5.19. Окончательный вид прямоугольного блока, созданный при помощи CSS

Гибкость, которую обеспечивает CSS (за счет отказа от использования изображений с фиксированной шириной), просто удивляет. Всего несколько строк в таблице стилей, и прямоугольный элемент преобразуется. Не говоря уже о том, что модуль становится очень мобильным. Его высота и ширина могут меняться, подстраиваясь под требования любого макета, а визуальное оформление не пострадало.

Используя возможности CSS и CSS3 вместо изображений, мы можем обеспечить единообразие дизайна. Более старые версии браузеров (до IE8) отобразят обычный угловатый блок без градиентного фона. Однако сам элемент и контент будут работать со всеми браузерами и устройствами. Просто дизайн будет несколько более привлекательным в действующих и последующих версиях.

ДРУГИЕ СПОСОБЫ СКРУГЛЕНИЯ УГЛОВ

Рассмотренный ранее метод хорошо проиллюстрировал процесс скругления углов прямоугольного блока с фиксированной шириной. Мы использовали фоновые изображения и возможности CSS. Но нередко случаи, когда CSS3 не может удовлетворить требованиям дизайна, тогда задача становится сложнее. Чтобы решать такую задачу, лучше всего использовать «резиновый» блок. Под «резиновым» блоком я подразумеваю элемент, который может растягиваться как по горизонтали, так и по вертикали.

Почему это сложнее? Потому что вам понадобятся четыре изображения — по одному на каждый угол. (А CSS3 использовать нельзя!) Каждому углу должно соответствовать отдельное изображение, чтобы блок мог растягиваться во всех направлениях. А четыре изображения потребуют четырех элементов разметки, чтобы связать их с фоном.

Некоторые методы используют сгенерированный контент в таблице стилей, задействуя псевдоэлементы `:before` и `:after` для вставки нескольких фоновых изображений в один блок. Проблема заключается в том, что IE8 и более старые версии не поддерживают такой контент. Поэтому многие пользователи не смогут увидеть скругленные углы вашего дизайна.

Так что же делать? Если фиксированная ширина известна заранее, используйте описанный выше метод с парой фоновых изображений: одно — для верхних скругленных углов, а второе — для нижних. Чтобы привязать эти изображения, хватит пары элементов разметки, и никаких проблем. Если же вам нужен «резиновый» элемент, тогда понадобится дополнительная разметка, чтобы пользователи IE смогли увидеть ваш прекрасный дизайн.

ПРИМЕЧАНИЕ

Следующий метод легче всего реализовать с помощью `border-radius`, как и в предыдущем примере. Однако если вы не можете использовать CSS3 для скругления углов, то читайте дальше и научитесь создавать гибкие скругленные элементы без помощи CSS3. Это довольно сложно и не очень красиво, поэтому я советую придерживаться принципа последовательного улучшения дизайна, используя CSS3 только для украшения.

СКРУГЛЕНИЕ С УЛЫБКОЙ

Автором этого метода является Итан Маркотт, разработчик стандартов и научный редактор первого издания этой книги. Несколько лет назад он занимался разработкой веб-сайта Browse Happy (рис. 5.20, <http://browsehappy.com>).



Рис. 5.20. Цель сайта Browse Happy — показать людям, что их возможности не ограничиваются Internet Explorer, существуют и другие альтернативы

На странице находится скругленный прямоугольный блок, позволяющий пользователю переключиться на соответствующий браузер. На элементе располагаются название браузера и его пиктограмма (рис. 5.21).



Рис. 5.21. Скругленный прямоугольный блок может расширяться в любом направлении

Итан не стал задавать элементу фиксированную ширину, а предусмотрел возможность изменения размеров по горизонтали и вертикали в зависимости от объема (или размера) контента с помощью метода «раздвижных дверей» (рис. 5.22).

Давайте воспользуемся опытом Итана и создадим гибкий элемент, который сможет вместить абсолютно любой контент.



Рис. 5.22. При увеличении шрифта блок увеличивается по горизонтали и вертикали, а его углы остаются скругленными

ПРИМЕЧАНИЕ

Я немного изменил исходную разметку, чтобы упростить пример. Изначально скругленный элемент был частью списка определений, содержащего название «переключателя» и другую информацию. Это хороший пример использования списков определений. Дополнительную информацию можно узнать на сайте <http://browsehappy.com>.

Структура разметки

Для создания в полной мере гибкого элемента нам понадобится дополнительная разметка. Она нужна для вставки четырех независимых углов таким образом, чтобы элемент мог расширяться во всех четырех направлениях.

Учитывая, что нам нужны как минимум четыре элемента для задания фоновых изображений, мы создали следующую структуру:

```
<div class="container">
  <p class="desc">This box is:</p>
  <p class="link"><em><a href="/browsers/firefox/">Indestructible!</a></em></p>
</div>
```

Контейнер `<div>` и следующая за ним строка дают нам два первых элемента. Для создания еще двух элементов добавим абзац с хвастливой ссылкой `Indestructible!` и (именно здесь появится дополнительная разметка) элемент ``. Получившаяся разметка кажется несколько избыточной. Я использую ``, чтобы выделить ссылку. Обычно я стараюсь избежать избыточности, но

в данном случае нам понадобится четвертый элемент, чтобы сформировать все четыре угла (рис. 5.23).

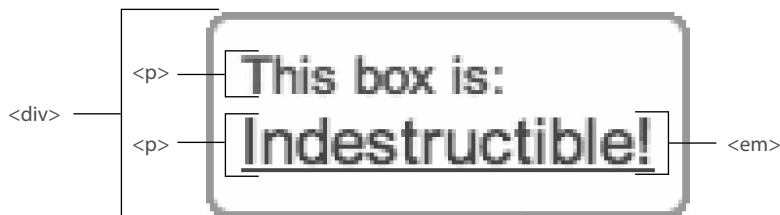


Рис. 5.23. Мы структурировали разметку элемента, доведя количество элементов до четырех, чтобы привязать к ним фоновые изображения

Стратегия размещения изображений

Мы обратимся к методу с сайта Browse Harry, в котором используются лишь два изображения, хотя будем ссылаться на фоновое изображение четыре раза (по разу на каждый угол). Вы сразу все поймете, как только посмотрите на изображения.

На рис. 5.24 показан рисунок `rounded-left.gif` — его ширина составляет 9 пикселей, — который содержит верхний и нижний левые углы. Высота этого изображения немного больше потенциальной высоты контента.

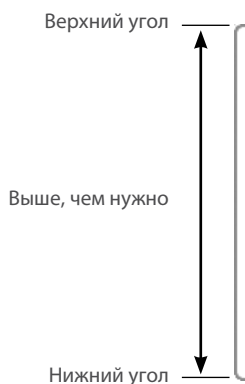


Рис. 5.24. Изображение `rounded-left.gif` содержит левые нижний и верхний углы

Рисунок `rounded-right.gif` (рис. 5.25) содержит правый верхний и нижний углы, а также верхнюю, правую и нижнюю границы. Высота этого изображения совпадает с предыдущим, но оно гораздо шире, чем нужно.

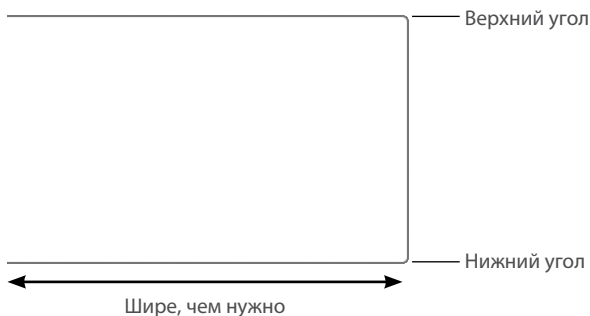


Рис. 5.25. На `rounded-right.gif` находятся правые верхний и нижний углы, а также границы элемента

Теперь мы будем располагать картинки как нам нужно. Выводим `rounded-left.gif` по верхнему краю блока для левого верхнего угла (рис. 5.26), после чего мы используем это же изображение, чтобы создать левый нижний угол, но выровняем его по нижнему краю.



Рис. 5.26. При выравнивании по верхнему краю нижний угол изображения будет виден, только если размер блока станет достаточно большим

Так как блок не превышает изображение по высоте, неиспользуемый скрытый угол никогда не отобразится (вот почему была выбрана такая высота изображения). Точно так же мы поступим с `rounded-right.gif` — используем его дважды — для правого верхнего и правого нижнего углов.

Заранее продумайте, какой высоты и ширины должны быть эти изображения, учитывая возможный контент. Добавьте чуть-чуть места про запас, чтобы учесть непредсказуемость размеров шрифта и объема контента.

Пора переходить к стилям и объединить все элементы воедино.

Применяем стили

Мы не будем использовать фиксированную ширину для блока, но нам нужно, чтобы закругленные углы «обхватывали» контент, поэтому сделаем контейнер плавающим. Это позволит избежать ситуации, когда ширина элемента становится равной ширине окна (или другого контейнера над ним). Вместо этого растяжение элемента будет определяться шириной внутреннего контента:

```
.container {
  float: left;
  color: #666;
}
```

Мы выровняли блок по левому краю и сделали цвет текста темно-серым.

Теперь займемся парой наших фоновых изображений, привязав их к четырем элементам разметки. Прежде всего сделаем правый верхний угол фоном для основного контейнера, используя верхнюю половину изображения `rounded-right.gif`, задав выравнивание по правому верхнему краю блока.

```
.container {
  float: left;
  color: #666;
  background: url(img/rounded-right.gif) top right no-repeat;
}
```

Обратите внимание на то, что мы выравниваем фоновое изображение по правой верхней части блока.

На рис. 5.27 можно увидеть результат наших действий.

На очереди следующий элемент — первый параграф в контейнере, который мы определили как `class="desc"`. Давайте настроим отображение левого верхнего угла, используя верхнюю часть `rounded-left.gif`, для этого нужно выровнять изображение по левому верхнему краю. Кроме того, мы обнулим поля и отступы элемента `<p>`. К настройкам отступов мы еще вернемся.

```
.container {
  float: left;
  color: #666;
  background: url(img/rounded-right.gif) top right no-repeat;
}
.desc {
  margin: 0;
  padding: 0;
  background: url(img/rounded-left.gif) top left no-repeat;
}
```

На рис. 5.28 показан внешний вид блока после выравнивания второго фонового изображения, которое формирует левый верхний угол.

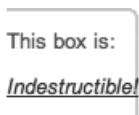


Рис. 5.27. Выравнивание изображения по правому верхнему краю делает угол видимым

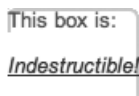


Рис. 5.28. Выравнивание изображения по левому верхнему краю делает угол видимым

Теперь займемся левым нижним углом, связав `rounded-left.gif` со вторым параграфом, который мы разметили как `class="link"`. Наши действия будут такими же, как при привязке верхней части изображения, но для формирования нижнего угла выравнивание делается по левому нижнему краю. Кроме того, мы добавим отступы (по 9px с трех сторон от первого параграфа), чтобы расширить свободное пространство вокруг элемента, и отступ слева от второго параграфа — 9px. Это значение совпадает с шириной изображения, что позволит увидеть угол позади ссылки «Несокрушимый!».

```
.container {
    float: left;
    color: #666;
    background: url(img/rounded-right.gif) top right no-repeat;
}
.desc {
    margin: 0;
    padding: 9px 9px 0 9px;
    background: url(img/rounded-left.gif) top left no-repeat;
}
.link {
    margin: 0;
    padding: 0 0 0 9px;
    background: url(img/rounded-left.gif) bottom left no-repeat;
}
```

На рис. 5.29 показаны результаты нашей работы. Теперь три из четырех углов находятся на нужных местах, нам остается только разобраться с последним.

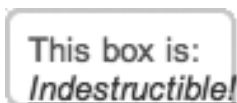


Рис. 5.29. С помощью изображения `rounded-left.gif` мы добавили третий угол, но на этот раз использовали выравнивание по нижнему краю

Последнее фоновое изображение привяжем к элементу `` (второй параграф). Мы выравниваем `rounded-right.gif` по правому нижнему краю. Кроме того,

зададим большой отступ, позволяющий разместить текст. Большинство браузеров воспринимают `` как команду сделать текст наклонным, так что нам придется изменить это правило.

```
.container {
  float: left;
  color: #666;
  background: url(img/rounded-right.gif) top right no-repeat;
}
.desc {
  margin: 0;
  padding: 9px 9px 0 9px;
  background: url(img/rounded-left.gif) top left no-repeat;
}
.link {
  margin: 0;
  padding: 0 0 0 9px;
  background: url(img/rounded-left.gif) bottom left no-repeat;
}
.link em {
  display: block;
  padding: 0 9px 9px 0;
  font-style: normal;
  background: url(img/rounded-right.gif) bottom right no-repeat;
}
.container a {
  font-size: 130%;
  color: #E70;
}
```

Считается, что `` — это строчный контейнер, который невозможно расширять так, как мы поступаем с родительским контейнером. Значит, в него невозможно вставлять фоновые изображения. Чтобы исправить ситуацию, мы применили правило `display: block;`, превращая `` в блочный элемент, заставляя его менять свои размеры, подстраиваясь под внутренний текст. Кроме того, мы добавили еще одно правило, увеличивающее размер текста ссылки и окрашивающее ее в оранжевый цвет. Мне кажется, что именно так и должна выглядеть несокрушимая ссылка.

На рис. 5.30 показан окончательный вид элемента с четырьмя закругленными углами, созданными с помощью двух фоновых изображений.

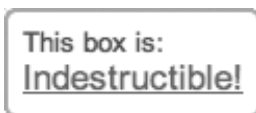


Рис. 5.30. Окончательный вид «несокрушимого» блока

Несокрушимый принцип

Обратите внимание на то, что закругленный элемент будет изменять свои размеры во всех направлениях, в зависимости от размера текста или количества контента, так же как и пример с сайта Browse Harry. Наше творение является абсолютно несокрушимым (рис. 5.31).

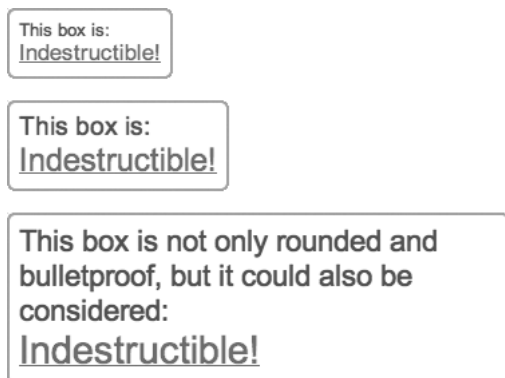


Рис. 5.31. Неважно, какой шрифт или объем контента мы будем использовать, наш блок может расширяться в любом направлении

Хитрости с боксами

Предыдущий пример показал, что без помощи CSS3 не так-то просто создать действительно гибкий элемент со скругленными углами. Поэтому вы должны познакомиться с разными способами обработки бокс-модулей. Так что сейчас мы поговорим о методах, использующих одно фоновое изображение и (или) цвет. Они относятся к чисто оформительским методам, но делают разметку красивой и обеспечивают гибкость при использовании различного контента.

Такие уловки могут стать секретным оружием разработчика пуленепробиваемого дизайна, но только в том случае, если методы будут использованы корректно. Правильно размещенное фоновое изображение превратит обычный прямоугольник в стильный элемент оформления. Рассмотрим несколько примеров.

Один скругленный угол

Представим, что у нас есть список из нескольких текстовых элементов. Простая разметка может иметь вид:

```
<ol>
  <li>Lorem ipsum dolor sit amet ...</li>
  <li>Lorem ipsum dolor sit amet ...</li>
```

```

    <li>Lorem ipsum dolor sit amet ...</li>
    <li>Lorem ipsum dolor sit amet ...</li>
</ol>

```

Под нечетные строки подложен цветной фон, позволяющий визуально разделить пункты списка (рис. 5.32), поэтому введем класс для выделения строк, чтобы управлять их стилем с помощью CSS:

```

<ol>
    <li class="alt">Lorem ipsum dolor sit amet ...</li>
    <li>Lorem ipsum dolor sit amet ...</li>
    <li class="alt">Lorem ipsum dolor sit amet ...</li>
    <li>Lorem ipsum dolor sit amet ...</li>
</ol>

```

CSS-код, работающий с нашим списком, будет выглядеть так:

```

ol {
    width: 22em;
    margin: 0;
    padding: 0;
    list-style: none;
}
ol li {
    margin: 0;
    padding: 1em;
}
ol li.alt {
    background: #E0EAC5;
}

```

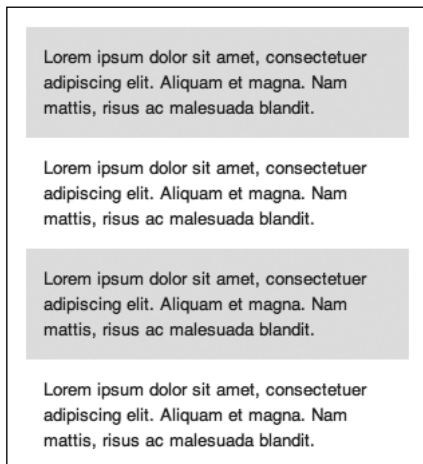


Рис. 5.32. Упорядоченный список, в котором под нечетные строки подложена цветная подложка

Впервые я встретился с этим трюком в Hicksdesign — блоге Джона Хикса (<http://hicksdesign.co.uk>). Джон использовал только один скругленный угол (правый нижний) на фоновом изображении (рис. 5.33), назначенным для разделения строк. Эта небольшая деталь добавляла изюминку и вполне соответствовала общему дизайну сайта.

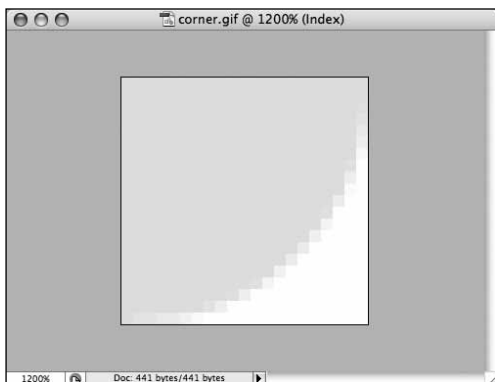


Рис. 5.33. Нижний правый угол GIF-изображения при увеличении

СОВЕТ

Можно было и не трогать разметку, то есть не добавлять класс `.alt` для элементов ``, а вместо этого использовать `nth-child` — псевдоселектор CSS3. С помощью `nth-child(odd)` можно задать цветной фон для каждого нечетного пункта списка следующим образом:

```
ol li:nth-child(odd) {  
    background: #E0EAC5;  
}
```

Если привязать это изображение к концу строки с цветным фоном (как это сделал Джон), то мы получим простой стиль, который не потребует дополнительной разметки и позволит обойтись элементарным CSS-кодом (рис. 5.34).

Стиль для нечетных пунктов списка `` задается следующим образом:

```
ol li.alt {  
    background: #E0EAC5 url(img/corner.gif) no-repeat bottom right;  
}
```

Выворачивая фоновое изображение по правому нижнему краю блока, мы сохраним стилевое оформление при любых размерах блока (рис. 5.35). Такой дизайн не зависит от ширины или высоты, и в этом его преимущество.

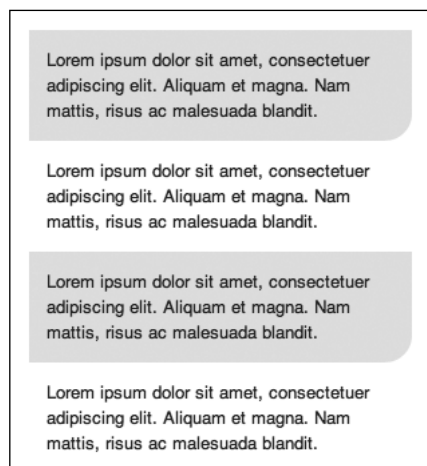


Рис. 5.34. Список с изменением фоновой заливки и скруглением правого нижнего угла

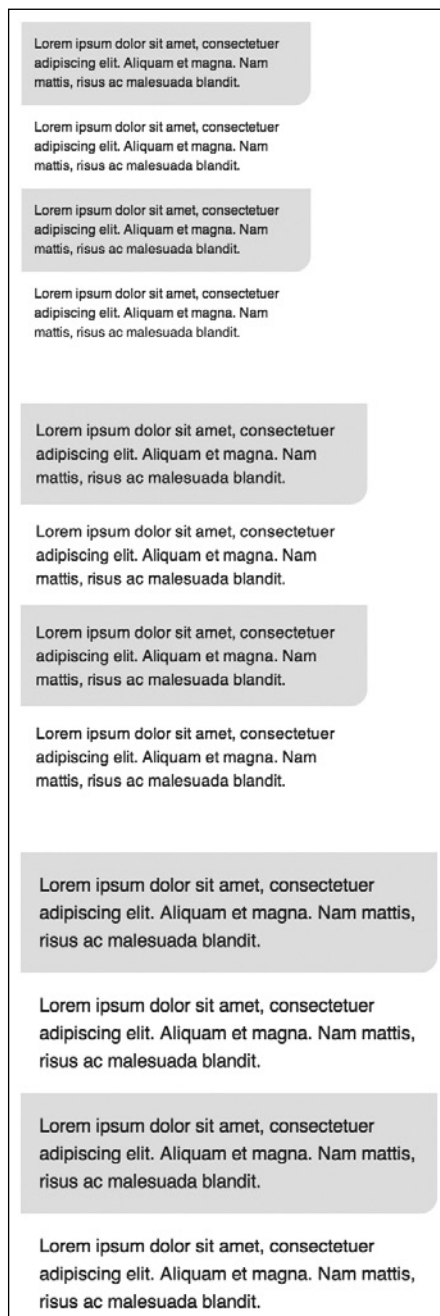


Рис. 5.35. При изменении размера текста скругленный правый нижний угол остается на месте, а размер цветной плашки меняется

ХИТРОСТИ С УГЛАМИ

Великолепный пример работы с углами можно найти в предыдущей версии блога бельгийского дизайнера Вирле Питерса (<http://veerle.duoh.com>).

На рис. 5.36 показана часть формы для комментариев пользователей, где слева вверху виден скругленный угол. Это классический пример дизайнерской хитрости, позволяющей использовать одно фоновое изображение, которое не зависит от размера заголовка. Размер текста, находящегося в верхней части изображения, может быть любым, он может увеличиваться, может состоять из нескольких строк, и это никак не повлияет на целостность дизайна. Это простой и пуленепробиваемый элемент дизайна.

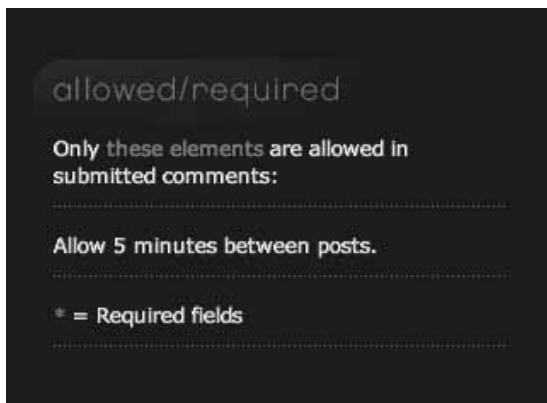


Рис. 5.36. Пример изменения элемента в блоге Верле Питерс

ПУЛЕНЕПРОБИВАЕМАЯ СТРЕЛКА

Конечно же, мы можем использовать не только скругленные углы. Посмотрите на рис. 5.37. На нем показан обычный квадратный элемент, а фоновое изображение делает из него однонаправленную стрелку. Используя фоновое изображение с высотой заведомо больше необходимой (рис. 5.38), мы можем просто центрировать его по вертикали, тогда видна будет большая (или меньшая) часть изображения, в зависимости от реального размера блока (рис. 5.39).



Рис. 5.37. Еще один зеленый квадратный элемент с фоновым изображением в виде стрелки

Разметка, создающая стрелку, будет простой:

```
<h2>This Way</h2>
```

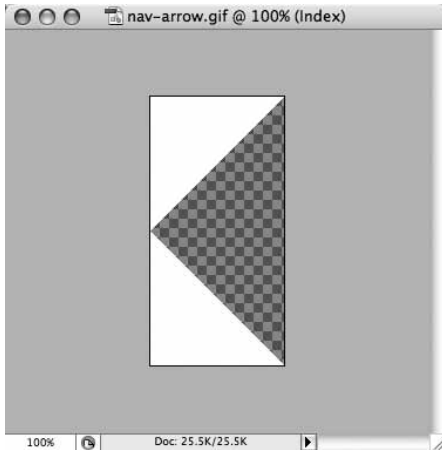


Рис. 5.38. Большой размер GIF-изображения с прозрачным участком (в этом месте будет просвечивать цвет блока) и белыми треугольниками, образующими стрелку



Рис. 5.39. В зависимости от размера блока (размера шрифта или количества контента) нам открывается большая или меньшая часть центрированного по вертикали фонового изображения

Следующие строки кода размещают изображение в точке `0 50%` (выравнивание по левому краю и центрование по вертикали), что позволяет создать эффект стрелки:

```
h2 {
  width: 4.5em;
  padding: .5em .8em .5em 1.5em;
  font-weight: normal;
  color: #FFF;
  background: #693 url(nav-arrow.gif) no-repeat 0 50%;
}
```

Обратите внимание на то, что я использую `em`, чтобы задать ширину и отступ. Таким образом удастся сохранить пропорции, отступы вокруг текста и длину сторон стрелки при изменении размеров надписи. Вы не обязаны использовать `em`, ведь это всего лишь совет.

ГОЛЬ НА ВЫДУМКУ ХИТРА

Чтобы разобраться с ограничениями `box`-модели, а также узнать, какие именно стандарты CSS поддерживают разные браузеры, понадобится много экспериментов с методами, делающими прямоугольный блок не таким угловатым. Четыре скругленных угла кажутся слишком сложной задачей? Попробуйте что-нибудь попроще (изобретательнее или гибче). Используя креативные методы, соче-

тающие в себе гибкость и стильный дизайн, вы создаете пуленепробиваемые элементы. Тем не менее иногда приходится чем-то жертвовать.

РЕЗЮМЕ

CSS позволяет создавать прямоугольные элементы со скругленными или непрямыми углами. Такие элементы достаточно гибки, так как не содержат избыточного кода и позволяют разделить дизайн и контент. В этой главе я рассказал о методах создания скругленных углов как для элементов с фиксированной шириной, так и для «резиновых» элементов. Надеюсь, это подтолкнет вас к экспериментам по созданию пуленепробиваемых элементов, нарушающих законы box-модели CSS.

При создании пуленепробиваемых элементов помните:

- как правило, прямоугольные элементы со скругленными элементами и фиксированной шириной можно создавать с помощью экономной разметки и пары фоновых изображений;
- «резиновые» прямоугольные элементы со скругленными углами требуют больше разметки, необходимой, чтобы связать фоновые изображения с четырьмя углами; заранее обдумайте, стоит ли использовать дополнительную разметку;
- комбинируйте фоновые изображения и используйте выравнивание, для того чтобы отобразить нужные фрагменты;
- используйте избыточную ширину и высоту фоновых изображений, но тщательно продумывайте максимальные размеры элемента; учитывайте контент как самой страницы, так и контент, который может оказаться в создаваемом блоке, а после этого прибавьте еще немного;
- если возможно, используйте свойства CSS3 для визуального оформления элементов, это обеспечит максимальную гибкость и позволит избежать дополнительной разметки и несущественных изображений; помните, что старые браузеры могут отображать ваш дизайн иначе;
- экспериментируйте с простыми методами, используйте уловки и хитрости, чтобы упростить дизайн и придать обычным прямоугольным элементам стильный вид.

ГЛАВА 6

**НЕТ КАРТИНОК? НЕТ
CSS? НЕТ ПРОБЛЕМ!**



Контент должен прочитываться всегда, даже если отключено отображение картинок и не поддерживается CSS

Первые несколько глав книги были посвящены гибкости с точки зрения дизайна. Мы говорили о том, как важно, чтобы компоненты могли подстраиваться под объем и размер контента. На многих этапах разработки сайта можно принять дополнительные меры, чтобы сделать дизайн пуленепробиваемым и обеспечить его гибкость. Будет ли доступна информация на странице, если CSS не поддерживается или отключен? Сможет ли пользователь видеть и читать контент, если изображения слишком медленно загружаются или вообще отключены? Это очень важные вопросы, которые необходимо решать при разработке веб-сайта.

Планирование с учетом наихудшего сценария сделает ваш веб-сайт работоспособным в любых ситуациях. В данной главе мы рассмотрим две стратегии пуленепробиваемого дизайна, которые помогут вам в ситуации, когда изображения и (или) CSS недоступны.

ОБЩЕПРИНЯТЫЙ ПОДХОД

Несколько лет назад я получил сообщение от одного из пользователей, читающих мой блог SimpleBits (www.simplebits.com). Я узнал, что несмотря на то что пользователю нравится мой сайт, ему часто приходится отключать отображение картинок, так как он использует медленное соединение с Интернетом и ему хочется видеть текст, не дожидаясь загрузки изображений. И он не одинок в этом желании — многие отключают изображения в браузере для ускорения работы, загружая только контент и фоновые заливки (рис. 6.1).

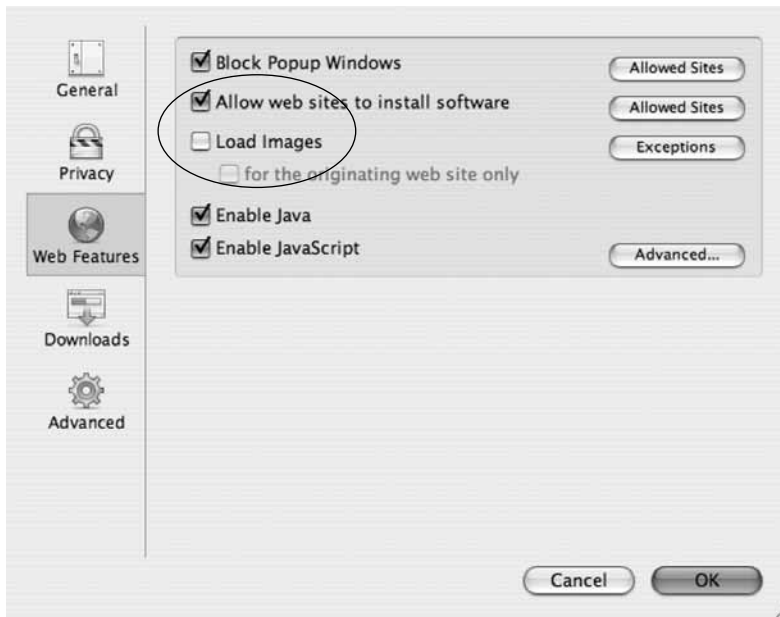


Рис. 6.1. Если вы уберете флажок у поля Load Images в браузере Firefox, все изображения будут отключены

Давайте займемся читателем, который отключает картинки. Но перед этим разберемся, как на сайте SimpleBits использовалось мозаичное заполнение фона.

В основе довольно распространенный подход — CSS и свойства `background` и (или) `background-image`. Изображение, находящееся под текстом, создает цветной фон, узоры и т. п. Столбцы и декоративная рамки на сайте SimpleBits так же были созданы с помощью мозаичного заполнения фона. Одно изображение отвечает за боковики, белое поле контента и светло-серую боковую панель (рис. 6.2).



Рис. 6.2. Изображение bg.gif повторялось по вертикали, располагаясь поверх темно-серого фона, для оформления многоколоночного сайта

Единственное изображение, повторяющееся по вертикали сверху вниз, формирует колонки и боковые полосы, которые располагаются поверх темно-серого фона, назначенного элементу `<body>` (рис. 6.3).

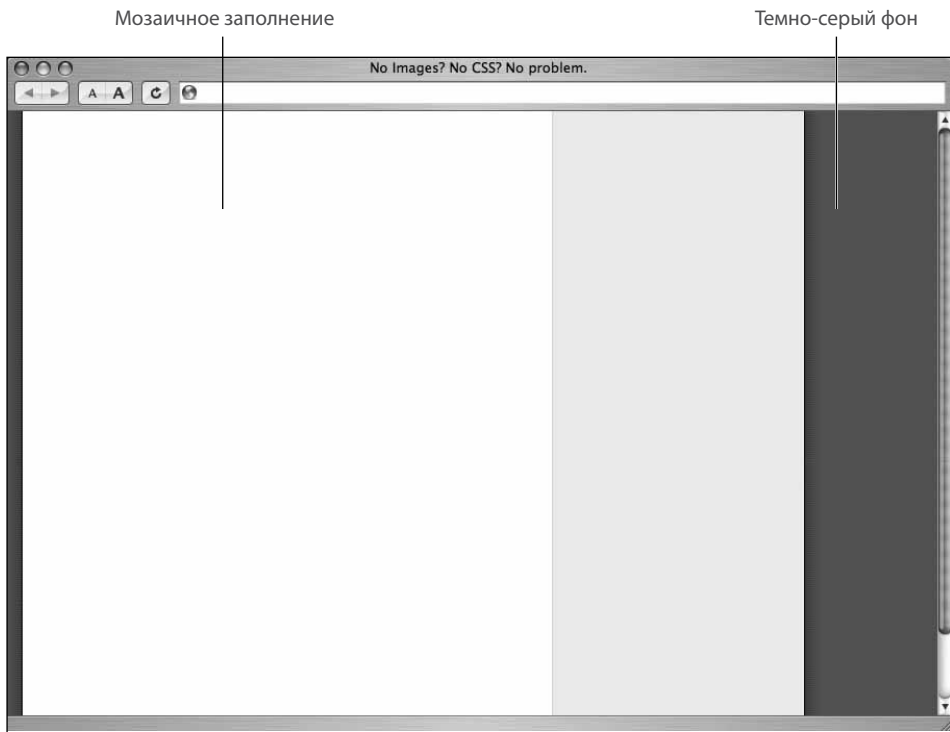


Рис. 6.3. Повторяясь, фоновое изображение bg.gif создает белые и серые колонки, расположенные за текстовым содержимым страницы

Задав темно-серый фон для элемента `<body>`, я определил цвет фона для всей страницы по умолчанию, который может быть отменен в пределах любого внутреннего элемента.

```
body {
  background: #666;
}
```

Мозаичное заполнение связано с внутренним контейнером `<div>`, в котором располагается весь контент страницы. Поэтому оно располагается поверх серого фона, создавая белые и серые колонки для текста. Вроде бы все работает хорошо. Изображение находится поверх фоновой цвета, создавая области для текста, который оказывается над повторяющимся изображением (рис. 6.4).

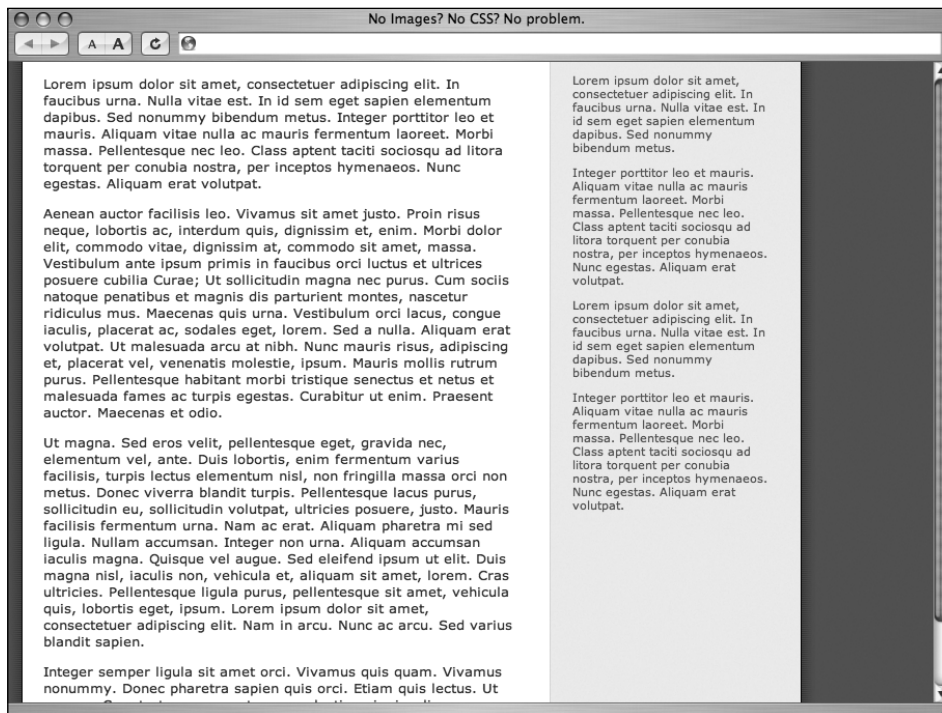


Рис. 6.4. Темно-серый текст расположен поверх повторяющегося изображения

Проблема возникает только когда изображения нет — вот уязвимое место нашего пулепробиваемого сайта.

Уязвимые места

Вернемся к читателю, который отключает изображения при использовании медленного соединения. Так он проинформирует меня, что контент моего сайта

становится нечитаемым либо очень тяжелым для восприятия. Должен признать, что я был несколько смущен. Давайте разберемся, почему так произошло.

Для контента использовались шрифты, окрашенные в два оттенка серого. Когда фоновое изображение не отображалось (или было отключено пользователем), текст оказывался поверх фоновой заливки, которая так же была темно-серой. В результате страница становилась практически нечитаемой. Пользователь с трудом мог различать текст в широкой колонке, а правая колонка вообще становилась невидимой! Цвет шрифта и цвет фоновой заливки здесь совпадали, поэтому текст просто исчезал.

На рис. 6.5 показано, что происходит, когда я отключаю мозаичное заполнение. Недостаточный контраст между цветом шрифта и фона в основной колонке затрудняет чтение. Однако, что еще хуже, текст в правой колонке вовсе исчез, так как его цвет совпал с цветом фона!

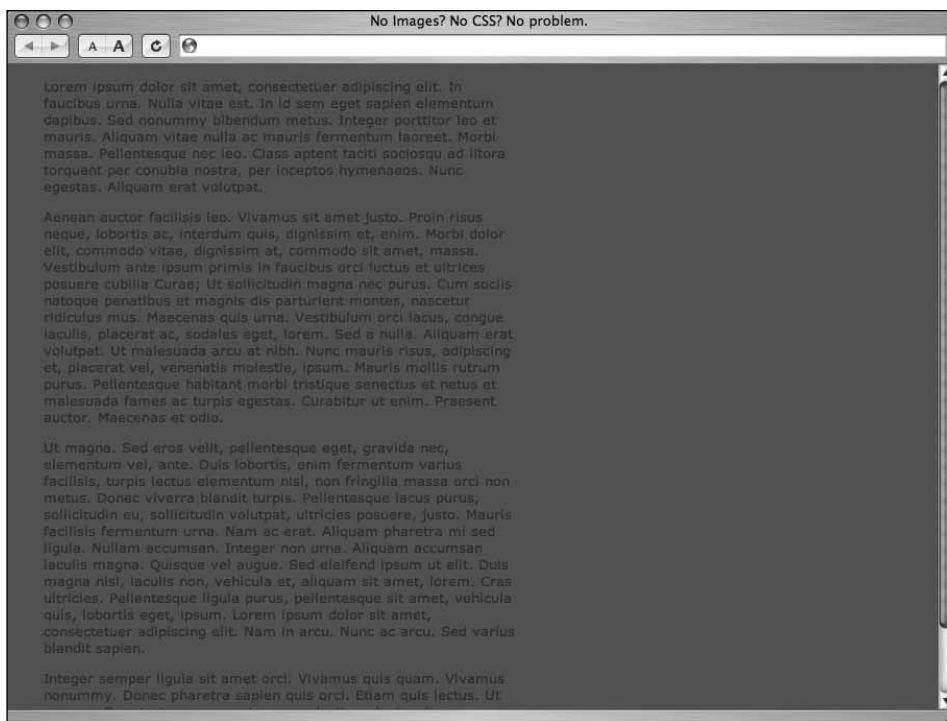


Рис. 6.5. Страница стала нечитаемой. А где же правая колонка?

Несомненно, такой сайт разочарует пользователей, отключающих изображения для экономии трафика или при использовании медленного соединения. При

некоторых сочетаниях цветов текст может просто исчезнуть. К счастью, для предотвращения такой проблемы достаточно предпринять несколько простых шагов.

ПУЛЕНЕПРОБИВАЕМЫЙ ПОДХОД

Давайте сделаем текст читаемым, даже в отсутствие изображений. Не забывайте, что цвет фона всегда должен соответствовать всем фоновым изображениям, которые вы используете.

Вернемся к нашему нечитаемому примеру. Достаточно было добавить полю контента фоновый цвет, чтобы мой читатель прочитал текст без проблем.

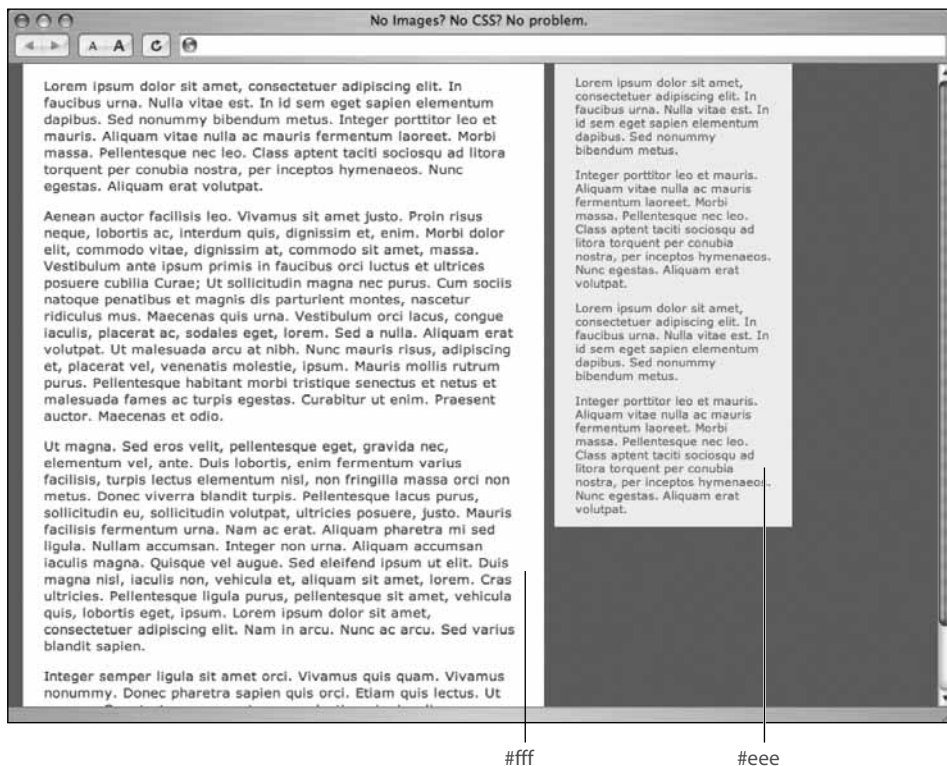


Рис. 6.6. При добавлении фоновых цветов текст становится читаемым

Так как часть контента страницы располагается поверх белого фона, мне нужно добавить этот цвет в контейнер `<div id="content">`, содержащий эту колонку:

```
#content {
    background: #FFF;
}
```

Таким же образом поступим с боковой колонкой, которая находится поверх светло-серого фона повторяющегося изображения, — зададим цвет соответствующему контейнеру `<div id="sidebar">`:

```
#sidebar {
    background: #EEE;
}
```

На рис. 6.6 показаны результаты после добавления двух правил к таблице стилей. Изображения все так же отключены. Обратите внимание на то, что при отключенном мозаичного заполнения каждая колонка текста имеет собственную фоновую заливку, перекрывающую темно-серый цвет, заданный по умолчанию для всей страницы.

ПРЕИМУЩЕСТВА ПУЛЕНЕПРОБИВАЕМОГО ПОДХОДА

Если вы потратите немного времени (буквально секунды) на подбор адекватных фоновых цветов, эквивалентных фоновым изображениям, то ваш сайт приблизится к пуленепробиваемому идеалу. Теперь пользователи, которые зайдут на ваш сайт, отключив изображения, чтобы сэкономить трафик, смогут прочесть текстовый контент. Если соединение будет медленным, то пользователь сразу увидит текст и фоновый цвет, а изображения загрузятся постепенно.

Пуленепробиваемость означает готовность работать с любым контентом. Это лишь еще один простой шаг к тому, чтобы обеспечить целостность дизайна при просмотре сайта без изображений.

В качестве иллюстрации давайте рассмотрим заголовки боковой колонки в старой версии сайта SimpleBits (рис. 6.7).

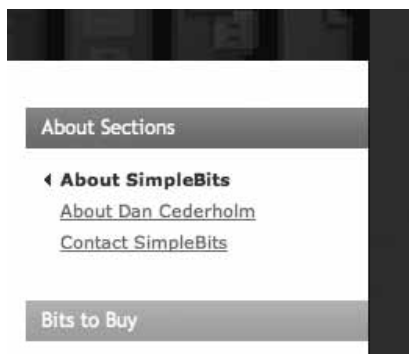


Рис. 6.7. На сайте simplebits.com заголовки About Sections и Bits to Buy были размечены как `<h3>`

Заголовки **About Sections** и **Bits to Buy** были размечены как `<h3>` (например, `<h3>About Sections</h3>`), а небольшое фоновое изображение было помещено за белым текстом.

CSS-код верхнего заголовка будет следующим:

```
#sidebar h3 {
  margin: 30px 0 12px 0;
  padding: 5px 10px;
  color: #FFF;
  font-size: 120%;
  background: url(..img/sub-h-bg.gif) repeat-x top left;
}
```

Там, где `sub-h-bg.gif` создавал фон зеленого цвета, текст был окрашен в белый цвет. Цвет фона, установленный на странице по умолчанию, также был белым. Думаю, вы понимаете, к чему я клоню.

Если отключить изображения, то заголовки пропадут (рис. 6.8).

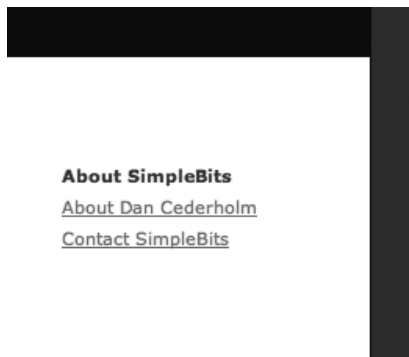


Рис. 6.8. Текст заголовка чудесным образом исчез

К счастью, мы можем воспользоваться предыдущим советом. Выбрав эквивалентный фоновый цвет и связав его с изображением, мы обеспечим пуленепробиваемость:

```
#sidebar h3 {
  margin: 30px 0 12px 0;
  padding: 5px 10px;
  color: #FFF;
  font-size: 120%;
  background: #538620 url(..img/sub-h-bg.gif) repeat-x top left;
}
```

В CSS-коде мы назначили фоновую заливку, совпадающую с цветом фонового изображения. Так как фоновые изображения всегда располагаются поверх фоновых цветов, то мы обезопасили себя на случай, если браузер не сможет

загрузить изображения, когда они были отключены пользователем или из-за низкой скорости соединения.

Теперь, если пользователь увеличивает размер шрифта для текста, лежащего поверх изображения, фоновая картинка будет выравниваться по верхней части прямоугольного блока, а фоновый цвет при необходимости будет растягиваться (рис. 6.9). Мы гарантируем, что белый текст заголовка останется читаемым при любых условиях, поскольку будет находиться поверх зеленого фона.

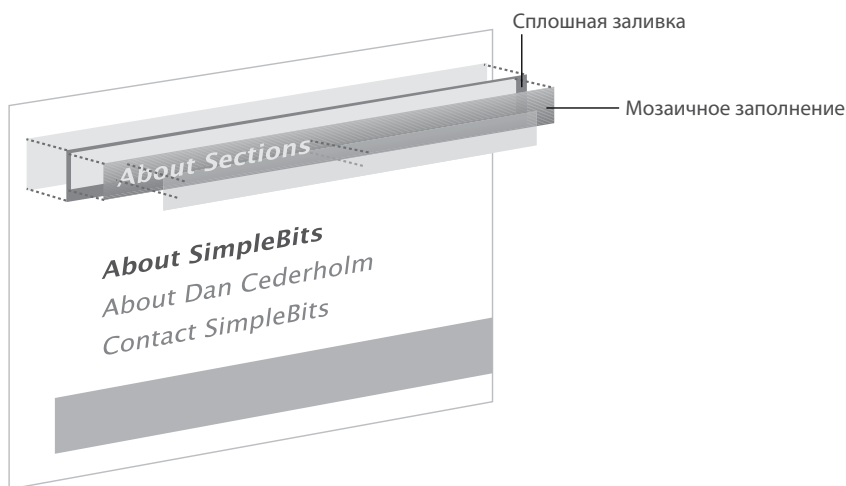


Рис. 6.9. Трехмерное представление, демонстрирующее порядок наложения слоев. Обратите внимание, что белый текст и мозаичное заполнение фона располагаются поверх сплошной фоновой заливки

На рис. 6.10 показаны заголовки с отключенными изображениями, однако в этот раз эквивалентный фоновый цвет делает текст читаемым. Мы задали цвета, а теперь посмотрим, что произойдет, если CSS недоступен.

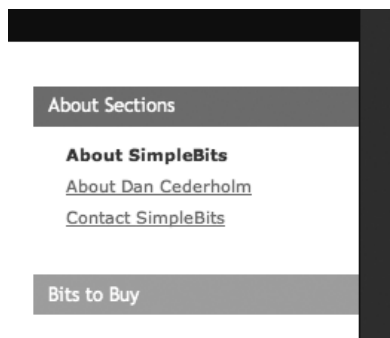


Рис. 6.10. Если предусмотреть фоновые заливки контрастных цветов, заголовки будут читаться даже при отсутствии изображений

Со стилем или без стиля

До этого момента мы в основном использовали возможности CSS, чтобы создавать пуленепробиваемый дизайн, учитывая всевозможные сценарии и защищая контент. Но что случится, если CSS не работает? Как будут выглядеть и работать страницы, на которых остался только голый контент без дизайна?

Этот очень важный, но простой шаг я рекомендую выполнять всем веб-дизайнерам при разработке сайтов с использованием экономичной разметки и CSS. Если вы посмотрите на структуру страницы с отключенным дизайном, вам станет понятно, как будет отображаться контент в браузерах и устройствах, не поддерживающих CSS. Несмотря на то что при создании привлекательного дизайна вы опираетесь на возможности CSS, важно убедиться, что страница останется читаемой и функциональной, если CSS не поддерживается или недоступен.

Так что не забудьте проверить пуленепробиваемость — полностью отключите дизайн и оцените удобочитаемость страницы. Она все еще хороша?

10-СЕКУНДНАЯ ПРОВЕРКА ЮЗАБИЛИТИ

Я называю эту задачу «10-секундной проверкой юзабилити». Конечно, это не научный метод определения юзабилити сайта, который предполагает фокус-группы, исследования, время и множество весьма спорных оценок. Однако если вы просто отключите оформление, то сможете управлять тем, как мир видит вашу страницу. Большинство пользователей используют современные браузеры и получают удовольствие от привлекательного дизайна. Но некоторые браузеры и устройства отображают лишь контент. Помните об этом, разрабатывая дизайн.

В старые времена дизайнеры опирались на таблицы и разметку (теги ``, изображения-разделители и т. п.), чтобы создать страницу, которая будет одинаково выглядеть во всех браузерах. В большинстве случаев так и происходило, но сами сайты были негибкими и недоступными ни для каких приложений и устройств, кроме стандартного браузера. Так как количество разнообразных устройств и приложений постоянно растет, разработчики веб-сайтов считают, что их труд должен быть доступен всем категориям пользователей, а значит, необходимо проверять дизайн на работоспособность в различных ситуациях. Отключение CSS — лишь одна из ситуаций. Посмотрим на возможные результаты.

Общепринятый подход

Наиболее опасный сценарий — на сайте использована взрывоопасная смесь из CSS, использованного для конкретных элементов дизайна, презентационная

разметка и фоновые изображения, вставленные непосредственно в разметку страницы (или даже вложенные таблицы).

На рис. 6.11 можно увидеть сайт BaseballDog.com (выдуманный пример, основанный на реальном сайте, который отказался фигурировать в книге) с отключенным CSS. Страница становится практически нечитаемой: вы видите черный текст (цвет по умолчанию) и синие ссылки поверх темно-синего фонового изображения. В нормальных условиях, то есть с работающим CSS, сайт выглядит отлично, и вся информация читается.



Рис. 6.11. Внешний вид сайта BaseballDog.com, созданного на базе реального примера, при отключении CSS. Обратите внимание, как тяжело читаются текст и ссылки при использовании настроек по умолчанию

Незначущая графика (повторяющееся фоновое изображение) оказалась тесно связана со структурой документа и разметкой. Чтобы разместить повторяющееся фоновое изображение, дизайнеры сайта BaseballDog.com добавили свойство фона для элемента `<body>`:

```
<body background="/img/tile.gif">
```

Казалось бы, ничего страшного, но прямое добавление графики в разметку имеет последствия. Мы видим их, когда отключаем CSS.

CSS используется только для управления текстом и позиционирования. Как только CSS перестает работать, информацию на странице невозможно прочесть. Очевидно, что дизайнеры BaseballDog.com еще не перешли на CSS и значимую разметку, чтобы разделить контент и дизайн, сделав сайт доступным и удобным в любых ситуациях.

При разработке сайта можно использовать и гибридный (промежуточный) подход, используя CSS для отдельных элементов, а таблицы — для создания

структуры и сетки страницы. Если все выполнено хорошо, даже при отключении стилей CSS сайт будет функционировать. Хочу еще раз подчеркнуть — узнайте, что произойдет с сайтом при отключении стилей (как в случае с BaseballDog.com). Ваш сайт может перестать читаться или, в идеале, остаться без изменений. Рассмотрим еще один пример.

ПУЛЕНЕПРОБИВАЕМЫЙ ПОДХОД

При просмотре без CSS удачная структура страницы видна сразу. Давайте взглянем на старую версию сайта разработчика антивирусного программного обеспечения McAfee (www.mcafee.com) (рис. 6.12). При просмотре с отключенными стилями в глаза бросаются продуманная организация и читаемый контент, а акценты расставлены с помощью разметки, которая хорошо видна в любом браузере или устройстве (рис. 6.13).



Рис. 6.12. Чистый дизайн домашней страницы McAfee, по состоянию на февраль 2005 года, является результатом продуманной структуры разметки и CSS



Рис. 6.13. С отключенным CSS становится видна основа сайта McAfee

По мере того как вы просматриваете страницу сверху вниз, становится очевидно, что для структурирования названий ссылок, описаний и списков ссылок была использована значимая разметка. Как будто вы сделали рентгеновский снимок сайта и теперь изучаете его кости (рис. 6.14).

Если браузер или устройство не поддерживают CSS, оформление сайта исчезает, однако контент и функциональность остаются. Задумайтесь, телефоны, планшеты, устройства с маленькими экранами, читалки и текстовые браузеры — большая часть этих устройств поддерживает таблицы стилей в минимальном объеме, а самые старые — вообще не поддерживают. Сделав страницу читаемой и удобной при любом отображении, вы делаете еще один шаг к пуленепробиваемости.

Отключите CSS, чтобы быстро и легко разобраться с логикой разметки, которую использовал разработчик, при этом нет нужды разбираться с исходным кодом.



Рис. 6.14. При прокрутке страницы становится видна большая часть структуры сайта McAfee

DIG DUG-ТЕСТ

Помните классическую старую аркаду Dig Dug? Вы управляли героем, который надувал и взрывал плохих парней с помощью воздушного насоса (рис. 6.15). Это была одна из моих любимых игр. Подобно этой игре, быстрый тест на целостность и пуленепробиваемость дизайна можно выполнить просто увеличивая или уменьшая размера текста. Используйте браузер, чтобы посмотреть на поведение страницы (мы так уже поступали в предыдущих примерах). Дизайн трещит по швам или остается незыблемым?

Увеличение текста — это то, к чему я часто прибегаю при разработке нового дизайна, и одна из первых вещей, которые я проверяю в готовых разработках. Как ведут себя элементы страницы, когда текст увеличивается или уменьшается?

Нужно убедиться, что текст не пропадет и не станет нечитаемым при увеличении размера шрифта.

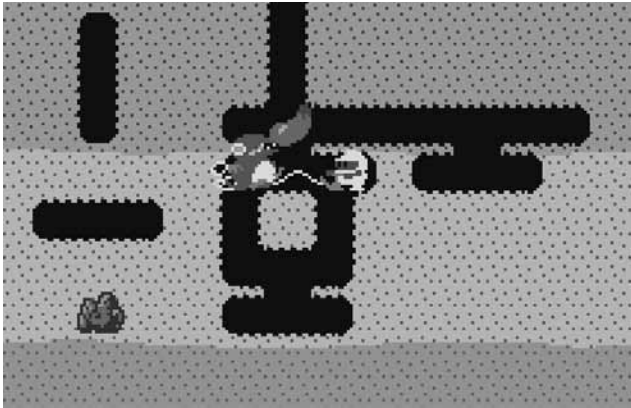


Рис. 6.15. Классическая аркада 80-х Dig Dug

Обычно удается увеличить размер текста на пару уровней выше базового размера `font-size`, пока дизайн не начнет разваливаться. Как и монстры, которых Dig Dug надувает, любой дизайн, несомненно, имеет свой предел прочности. Просто важно сохранять пространство для маневра и гибкость. Я думаю, что для большинства разработок прочность при увеличении на пару уровней — хорошая база, а не волонтаристское требование.

В предыдущих главах мы изучали методы, гарантирующие масштабируемость дизайна вне зависимости от контента. Просто не забывайте проверять прочность оформления, изменяя размер текста с помощью браузера (на Mac это выполняется клавишами `Command++` или `Command+-`, а на PC — `Ctrl++` или `Ctrl+-`). Добавьте эту проверку к тесту пуленепробиваемости.

ПРИМЕЧАНИЕ

В некоторых браузерах эти клавиатурные комбинации могут масштабировать страницу, а не размер шрифта, но это легко поправить в настройках.

ИНСТРУМЕНТЫ ПУЛЕНЕПРОБИВАЕМОСТИ

К счастью, существует несколько инструментов, которые делают процесс тестирования страницы простым и быстрым. Хороший инструмент должен быть доступен и удобен в использовании. Кроме того, он призван помогать вам, а не

мешать. Если вы собираетесь проверить читаемость текста при отключенных изображениях и (или) CSS, можно использовать букмарклеты, плагины Web Developer Extension (для Firefox и Mozilla) или Web Accessibility Toolbar (для IE/Win).

БУКМАРКЛЕТЫ

Букмарклеты — это небольшие приложения JavaScript, которые динамически иницииируют события на веб-странице. JavaScript используется для закладок или любимых ссылок, которые сохраняет браузер, предлагая доступ по щелчку. JavaScript — это всего лишь одна строка кода, помещенная в гиперссылку. Букмарклеты могут выполнять множество удобных операций, например, проверять HTML-код страницы, включать или отключать таблицы стилей, переназначать размеры для окна браузера для симуляции разных разрешений экрана и т. д.

Одно из хранилищ букмарклетов — Accessify.com — находится по адресу www.accessify.com/tools-and-wizards/accessibility-tools/favelets/ (рис. 6.16). Здесь вы найдете список полезных ссылок. Сохранив закладки на эти скрипты в вашем любимом браузере, вы сможете получить быстрый доступ к различным режимам тестирования.

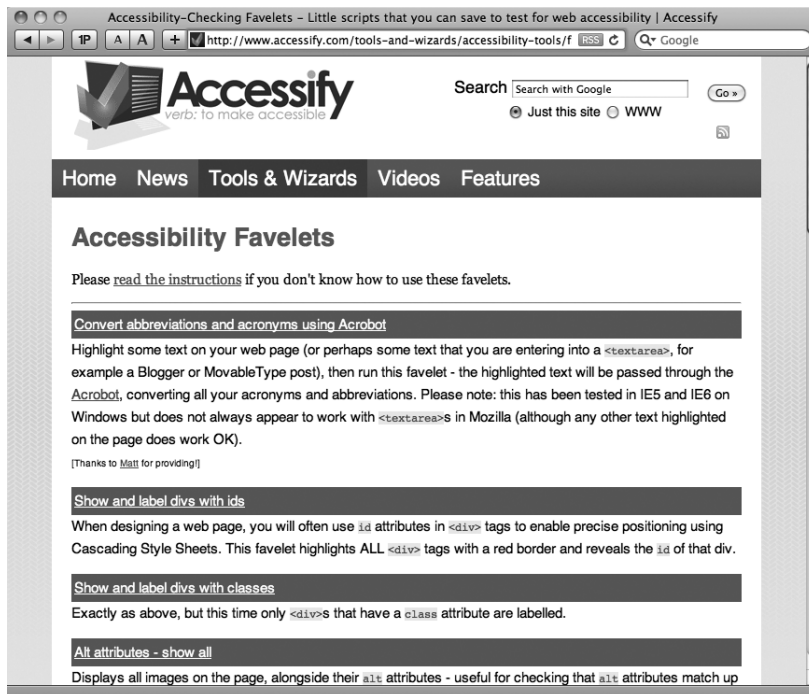


Рис. 6.16. Accessify.com — полезная коллекция ресурсов и ссылок, помогающих протестировать доступность веб-сайта

На рис. 6.17 видно, что я добавил букмарклет **Disable stylesheets** (Отключение таблицы стилей) в закладки браузера. Теперь, чтобы отключить CSS на любом сайте, мне достаточно нажать на соответствующий значок. Десятисекундный тест юзабилити, который я уже упоминал, стал еще проще.

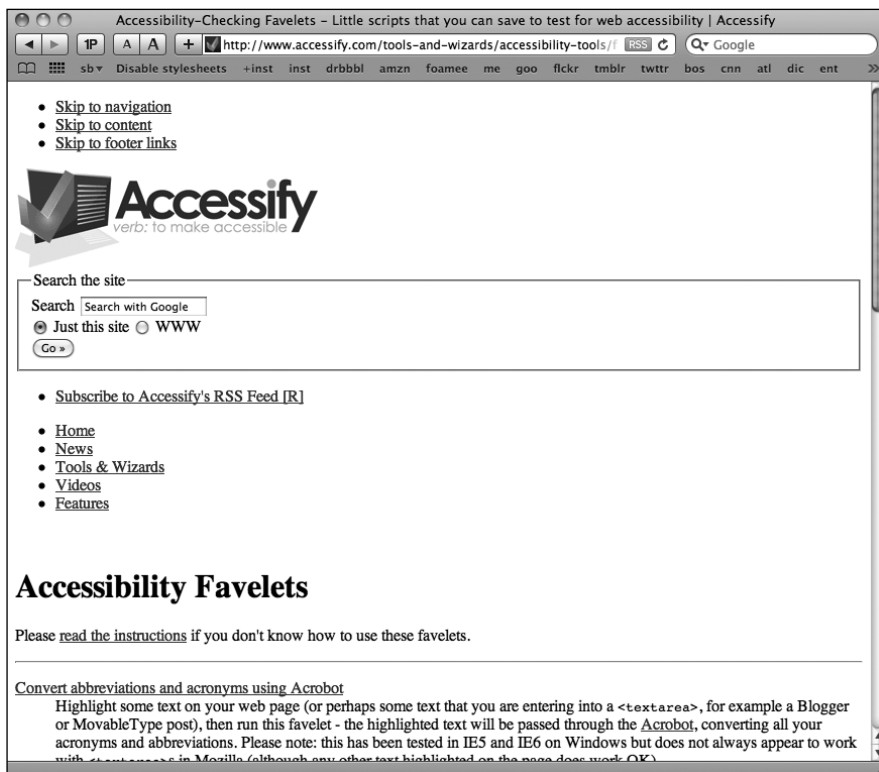


Рис. 6.17. После того как вы добавите букмарклет в закладки браузера, он будет ждать, пока вы нажмете на его значок, чтобы отключить таблицы стилей на любом сайте

Аналогично другие букмарклеты могут изменять страницу по-разному. Например, букмарклет **Show all DIVs** (Показать все контейнеры DIV) выделяет все элементы `<div>`, позволяя вам мгновенно увидеть структуру страницы (рис. 6.18).

COBET

Букмарклеты можно найти на сайтах: <http://tantek.com/favelets/> и www.andybudd.com/bookmarklets.

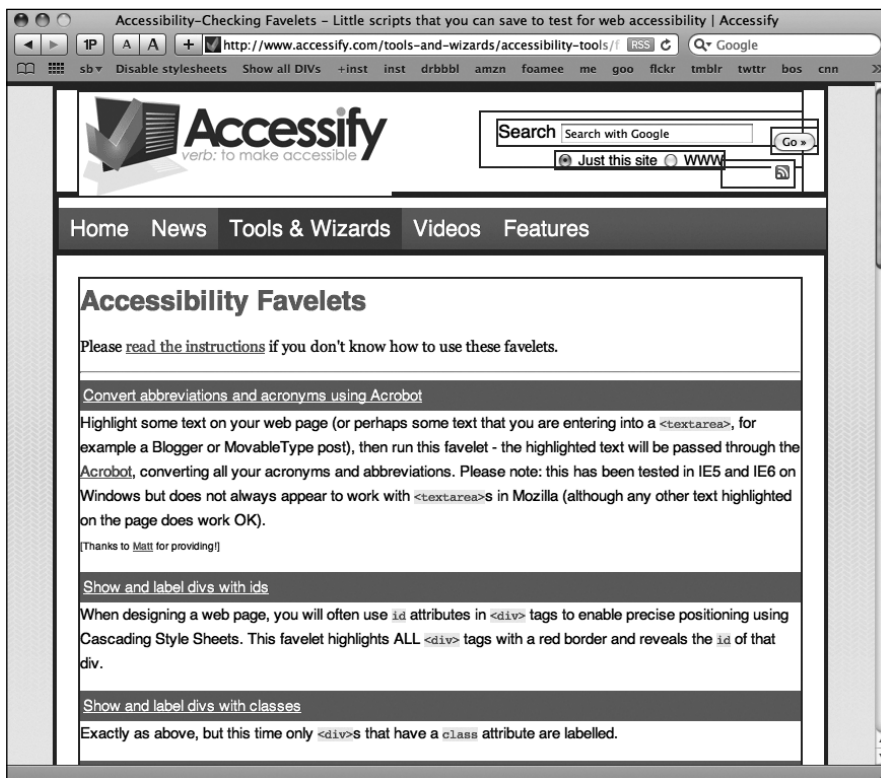


Рис. 6.18. Accessify.com выглядит так после того, как вы запустите букмарклет «Показать все контейнеры DIV», который выделяет все элементы `<div>` красной рамкой

Благодаря ненавязчивости, букмарклеты могут взять на себя рутинную дизайнерскую работу и позволить вам проверять пуленепробиваемость в любое время.

WEB DEVELOPER EXTENSION

Если вы используете Firefox или Chrome, я настоятельно рекомендую плагин Web Developer Extension, разработанный Крисом Педериком (<http://chrispederick.com/work/web-developer/>). Панель инструментов плагина представляет собой набор действий, аналогичных букмарклетам, которые можно применить к текущей странице. Эти действия расположены на панели инструментов в верхней части браузера (рис. 6.19).

Здесь есть любой тест, какой вы только можете вообразить. Например, тесты **Images** (Изображения) быстро проверяют страницу при отключенных изображениях, чтобы удостовериться, что фоновые цвета для всех фоновых изображений были подобраны правильно, как было описано в этой главе.

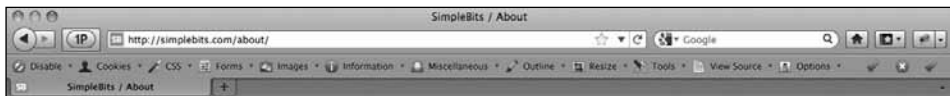


Рис. 6.19. Web Developer Extension содержит десятки тестов, доступных через выпадающие меню на панели инструментов

На рис. 6.20 показана команда **Hide Images** (Отключить изображения) для страницы SimpleBits. Вы мгновенно отключаете изображения, чтобы проверить читаемость страницы. Другие команды объединяют лучшие из букмарклетов (изменения размера окна, проверка разметки и таблицы стилей и т. п.) с другими возможностями, например редактированием CSS-кода страницы непосредственно в боковом поле. Это позволяет быстро проверить небольшие изменения.

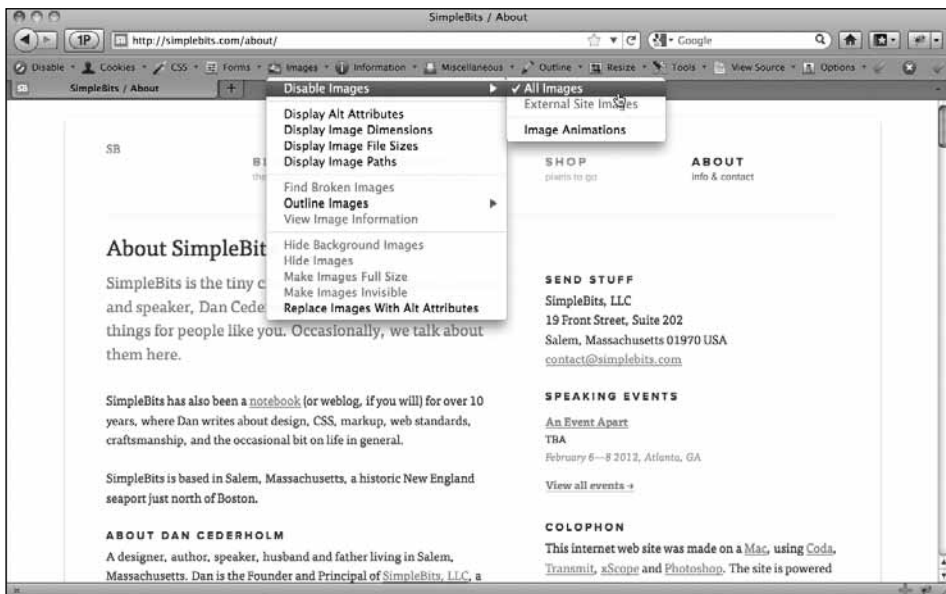


Рис. 6.20. Проверка отображения страницы в отсутствие изображений заключается в простом нажатии на ссылку, чтобы мгновенно проверить пуленепробиваемость страницы

Web Developer Extension создает «панель инструментов» пуленепробиваемости. Она необходима всем, кто разрабатывает страницы для веб-сайтов. Проверка разнообразных сценариев — отсутствие изображений, проблема CSS и др. — теперь не является трудностью.

WEB ACCESSIBILITY TOOLBAR

Как и предыдущий плагин, Web Accessibility Toolbar (www.visionaustralia.org.au/ais/toolbar/) предлагает «панель» полезных инструментов для пользователей IE/Win (рис. 6.21).

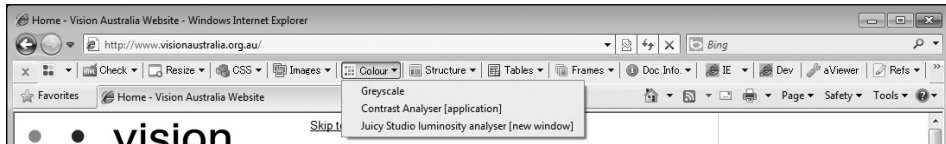


Рис. 6.21. Проверка страниц на соответствующем уровне доступности с помощью Web Accessibility Toolbar для IE/Win

Кроме проверки разметки, изменения размеров окна и опций CSS, панель инструментов предлагает дополнительные средства диагностики доступности, с помощью которых можно быстро проверить любую веб-страницу. Это отличный выбор для пользователей Internet Explorer для Windows.

FIREBUG

Firebug (<http://getfirebug.com>) — это аналогичное предыдущему пакету расширение для браузера Firefox, которое предлагает удивительно удобные инструменты проверки любой страницы:

- навигация по HTML-элементам;
- подсветка конкретного элемента;
- просмотр информации о параметрах элемента, стилях CSS или информацию о DOM (Document Object Model) и т. п.

Firebug позволяет изменять стили CSS и HTML-код текущей страницы в реальном времени — это полезно для отладки, внесения быстрых корректировок или решения возникших проблем (рис. 6.22).

Кроме того, Safari и Chrome предлагают отличные инструменты проверки и разработки, встроенные прямо в браузеры. Оценка пуленепробиваемости и отладка никогда не были такими простыми.

СОВЕТ

Если вы используете Opera, попробуйте Dragonfly — пакет средств разработчика для браузера Opera: <http://www.opera.com/dragonfly/>.



Рис. 6.22. Firebug — расширение для Firefox — дает разработчику интерфейс «лупы» для исследования HTML, CSS и JavaScript любой страницы в Сети

ПРОВЕРКА КАК ИНСТРУМЕНТ

При описании букмарклетов и плагина Web Developer Extension я упоминал об автоматической проверке кода страницы, однако проверка разметки и CSS сама по себе является отличным инструментом.

Соответствие разметки страницы и CSS спецификациями World Wide Web Consortium (W3C) гарантирует, что страница будет отображаться лучше и быстрее.

Полная проверка больших сайтов, над которыми работают большие команды разработчиков, может быть сложной или даже неосуществимой. Осложняет жизнь и программное обеспечение, управляющее контентом, которое ненамеренно выполняет неправильную разметку. Нужно стремиться к идеалу, но добиться совершенства не всегда удастся.

Я пытаюсь обратить ваше внимание на важность проверки на начальном этапе разработки веб-сайта — это еще один способ сделать сайт пуленепробиваемым. При создании гибкого доступного дизайна не забудьте уделять внимание проверке разметки и таблицы стилей, так неизвестно откуда появляющиеся ошибки визуализации принесут вам меньше головной боли.

Один незакрытый `<div>` — и рабочий шаблон на базе CSS превращается в раздражающий и нервующий кошмар. Однажды я потратил много часов на ис-

правление такой ошибки, поскольку не проверил страницу — простой пропуск закрывающего элемента породил огромные проблемы. Регулярно проверяя ваши файлы, вы избежите досадных ошибок, которые способны уничтожить весь дизайн. Такая небрежность может привести к разным результатам на разных браузерах. Проверенная страница имеет более высокие шансы выжить в разных браузерах.

Как проводится проверка

Чтобы проверить HTML-код, достаточно поставить **DOCTYPE** в начале страницы, а затем запустить программу проверки. DOCTYPE сообщает программе (и любому другому приложению) правила, которые действуют на вашей странице. В основе пуленепробиваемых примеров, использованных в этой книге, лежит HTML5, поэтому в начале страницы есть строка:

```
<!DOCTYPE html>
```

Конечно, вы можете использовать другой DOCTYPE. Просто важно указать тип и понимать, к чему это приведет, чтобы проверка была корректной.

СОВЕТ

За более подробной информацией о DOCTYPE обратитесь к сайту www.htmlhelp.com/tools/validator/doctype.html.

СОВЕТ

Синтаксис HTML5 менее жесткий, чем в XHTML (не нужен закрывающий элемент, атрибуты необязательно заключать в кавычки и т. п.). Однако я предпочитаю использовать XHTML-правила даже в HTML5. Я закрываю все элементы, пишу имена тегов в нижнем регистре и заключаю атрибуты в кавычки. Это помогает обеспечить чистоту разметки и быстрее отладить страницу при возникновении проблем с отображением CSS. У каждого есть собственный стиль написания HTML-разметки, который превращает ее в великолепный инструмент. При разработке пуленепробиваемой разметки необходим творческий подход.

Программа проверки — это приложение, которое проверяет ошибки документа в соответствии со спецификацией W3C. Объявление **DOCTYPE** информирует программу проверки, какую спецификацию использовать.

W3C предлагает бесплатную программу на базе веб-интерфейса, проверяющую разметку (<http://validator.w3.org/>). Чтобы использовать ее, достаточно ввести адрес в строку (рис. 6.23). Упомянутые ранее букмарклеты и Web Developer Extension

позволяют перейти к этому инструменту с помощью одного щелчка, просто передавая текущую страницу в программу проверки W3C.

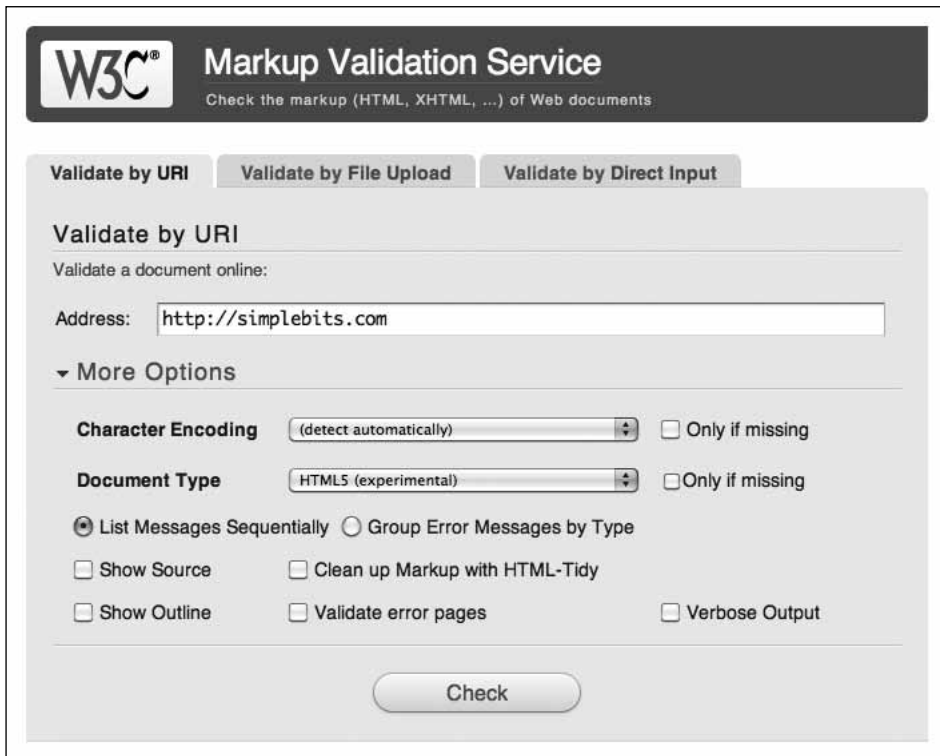
The image shows the W3C Markup Validation Service web interface. At the top, there's a dark header with the W3C logo and the text "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below the header, there are three tabs: "Validate by URI", "Validate by File Upload", and "Validate by Direct Input". The "Validate by URI" tab is selected. Under this tab, it says "Validate a document online:" followed by a text input field containing "http://simplebits.com". Below the input field is a section titled "More Options" with a dropdown arrow. Inside this section, there are several settings: "Character Encoding" set to "(detect automatically)" with a checkbox "Only if missing"; "Document Type" set to "HTML5 (experimental)" with a checkbox "Only if missing"; radio buttons for "List Messages Sequentially" (selected) and "Group Error Messages by Type"; checkboxes for "Show Source", "Clean up Markup with HTML-Tidy", "Show Outline", "Validate error pages", and "Verbose Output". At the bottom of the form is a large "Check" button.

Рис. 6.23. Бесплатная программа проверки W3C находится по адресу <http://validator.w3.org>

Я часто использую Validate by File Upload, чтобы проверить локальные файлы, над которыми еще идет работа. Программа тестирует локальные файлы так, как будто они уже размещены в Сети. Еще проще запустить Validate Local HTML с панели инструментов Web Developer Extension, о которой я уже писал. После выбора этой команды сразу загружается локальный HTML-файл, открытый в браузере. Еще раз повторяю, что проверка ошибок наиболее важна на ранней стадии.

Кроме разметки важно (и просто) проверять и CSS-код. W3C предлагает собственную программу (<http://jigsaw.w3.org/css-validator/>), которая очень похожа на программу проверки разметки. Эти функции проверки также доступны с помощью букмарклета или по одному «клику» панели инструментов плагина.

РЕЗЮМЕ

Целостность страницы в разных ситуациях не менее важна, чем гибкость оформления страницы. Проверьте страницу, отключив изображения или CSS, чтобы убедиться, что контент остается читаемым.

Инструменты обеспечения пуленепробиваемости — букмарклеты, панели инструментов браузера и программы проверки — помогают упростить тестирование и подготовить ваш дизайн к разнообразным ситуациям.

Вот простые правила:

- используйте 10-секундный тест юзабилити для проверки читаемости страницы при отсутствии изображений и (или) CSS; разберитесь, как будет вести себя ваша страница в различных ситуациях;
- не забывайте задавать фоновый цвет, эквивалентный цвету фонового изображения, использованного в оформлении страницы; тогда пользователи, которые отключают изображения или пользуются медленным соединением, смогут прочитать текст;
- используйте букмарклеты и (или) панели инструментов в браузере, чтобы проверка происходила легко и быстро; добавьте этот этап в график работ;
- выполняйте Dig Dug-тест для проверки целостности и масштабируемости дизайна;
- используйте проверку кода на этапе разработки как способ превентивного решения последующих проблем.

ГЛАВА 7

ПРЕОБРАЗОВАНИЕ ТАБЛИЦ



В большинстве примеров этой книги мы избавлялись от вложенных таблиц с помощью минимальной разметки и CSS. Но бывают случаи, когда таблицы идеально подходят для представления данных. Для представления финансовых, статистических или сравнительных данных таблица — единственно правильный выбор.

В этой главе мы поговорим о таблицах. Отказавшись от совмещения визуальных эффектов с данными, мы перенесем оформление (собственно дизайн) из разметки в таблицу стилей. Результатами станут сокращение кода, большая доступность таблиц и повышение гибкости дизайна, который будет легкоизменяемым или легкообновляемым.

Общепринятый подход

Как и другие компоненты, которые мы уже рассматривали, стильные таблицы можно создавать с помощью GIF-разделителей и дополнительных ячеек, содержащих элементы оформления. Дополнительные данные, отвечающие за внешний вид таблицы, никак не связаны со смысловым контентом, хотя они размещаются в одной и той же таблице. В современных браузерах такие таблицы выглядят изумительно, но доставляют множество проблем с точки зрения гибкости и доступности.

Давайте рассмотрим форум официального сайта одного из популярных научно-фантастических сериалов. Чтобы сохранить конфиденциальность оригинального сайта, назовем его Lance Spacerunner (рис. 7.1).

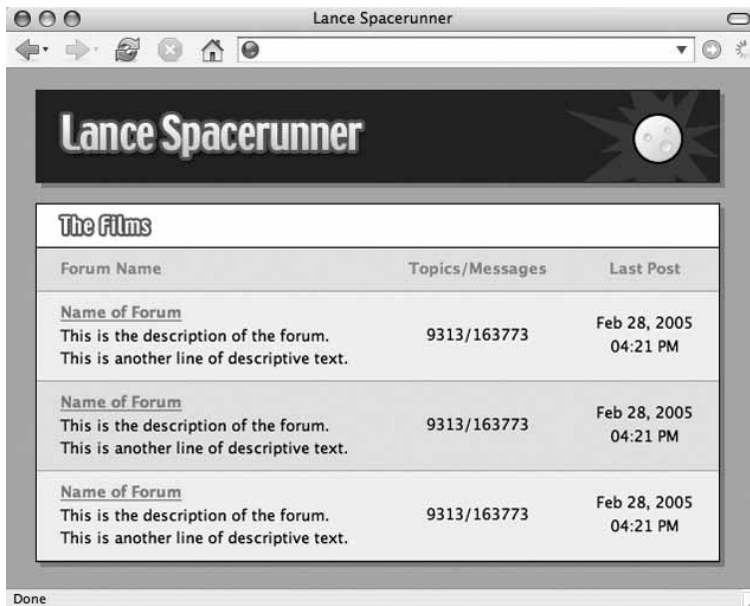


Рис. 7.1. Займемся доской объявлений выдуманного сайта Lance Spacerunner, у которого есть реальный сайт-прообраз

Доска объявлений — это простая таблица, в которой отображаются следующие данные — название и описание каждого раздела, количество тем и сообщений, а также время и дата последнего сообщения. В верхней части располагается название форума — The Films.

Давайте приглядимся к элементам оформления, которые являются причиной негибкости таблицы.

На рис. 7.2 показаны элементы дизайна таблицы, которые придется учитывать при изменении стиля ее оформления. Над таблицей расположено ее название, под названием таблицы — названия столбцов, далее идут строки с чередованием заливки, разделенные 1-пиксельной серой линией. Кроме того, таблица отбрасывает тень шириной 4 пиксела, которая смещена вправо и вниз, чтобы создавать эффект объема.

The diagram shows a table titled 'The Films' with three columns: 'Forum Name', 'Topics/Messages', and 'Last Post'. The table has three data rows, each with a light gray background. Annotations point to various design elements: 'Название таблицы' (Table title) points to 'The Films'; 'Заголовки столбцов таблицы' (Table column headers) points to the header row; 'Чередование цветов заливки строк' (Row background color alternation) points to the alternating light gray rows; 'Смещение тени' (Shadow offset) points to the bottom-left corner of the table; and 'Ширина тени 4 пиксела' (Shadow width 4 pixels) points to the bottom-right corner of the table.

The Films		
Forum Name	Topics/Messages	Last Post
<u>Name of Forum</u> This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM
<u>Name of Forum</u> This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM
<u>Name of Forum</u> This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM

Рис. 7.2. Элементы дизайна таблицы, представляющей раздел форума

В целом таблица выглядит довольно привлекательно. Такое оформление соответствует «межгалактическому» стилю всего сайта Lance Spacerunner.

Уязвимые места

Для создания стилевого оформления данные были размещены во вложенных таблицах с множеством дополнительных ячеек, в которых заданы интервалы и границы таблицы.

На рис. 7.3 показана таблица, с которой нам предстоит поработать. Обратите внимание на прографку (она сделана с помощью пакета Web Developer Extension, о котором я рассказывал в главе 6 «Нет картинок? Нет CSS? Нет проблем!»). Сразу бросается в глаза, что дополнительные ячейки используются для задания

отступов между ячейками с данными, определения отступов, прографки и добавления тени. Я называю такие ячейки избыточными, потому что они никак не связаны информацией в таблице и отвечают только за ее оформление.

The Films				
Forum Name		Topics/Messages		Last Post
<u>Name of Forum</u> This is the description of the forum. This is another line of descriptive text.		9313/163773		Feb 28, 2005 04:21 PM
<u>Name of Forum</u> This is the description of the forum. This is another line of descriptive text.		9313/163773		Feb 28, 2005 04:21 PM
<u>Name of Forum</u> This is the description of the forum. This is another line of descriptive text.		9313/163773		Feb 28, 2005 04:21 PM

Рис. 7.3. Когда мы включили прографку, стало понятно, что такая таблица требует невероятного количества кода

Одним из недостатков общепринятого метода является смешение оформления и контента. Такой гибрид позднее доставит вам множество проблем с дизайном. Кроме того, для реализации этой концепции вам понадобится большое количество кода, дополнительные ячейки и графика. Но если правильно использовать возможности CSS, вам понадобится написать всего несколько строк.

Кроме сокращения избыточного кода и разделения оформления и контента, можно внести несколько усовершенствований, которые сделают таблицу более доступной: использовать разметку, позволяющую просматривать страницу с помощью ридеров и текстовых браузеров, которые понимают только табличные структуры и данные. С этой точки зрения таблица пока не слишком доступна, так как приложениям приходится искать нужные данные среди избыточного кода. Этот процесс сродни маневрированию космического корабля, укорачивающегося от залпов неприятеля. Мы хотим сделать наши данные доступными самой широкой аудитории, а значит, будем использовать гибкий дизайн и CSS.

ПУЛЕНЕПРОБИВАЕМЫЙ ПОДХОД

Для того чтобы сделать дизайн пуленепробиваемым, нужно убрать излишний оформительский код из разметки, повысить доступность страницы и исполь-

зовать возможности CSS для наведения красоты. Давайте сохраним интересное оформление, но сделаем код простым и гибким.

Начнем со структурирования данных таблицы, используя правильную разметку, и ничего другого кроме нее. На данном этапе мы сосредоточимся на правильной структуре представления данных, а дизайном займемся позднее.

СТРУКТУРА РАЗМЕТКИ

Начиная с базовой сетки, создадим разметку так, чтобы таблица содержала название форума, темы/сообщения и дату последнего сообщения. Добавим заголовки для столбцов, помещая их в соответствующие колонки.

```
<table>
  <tr>
    <td>Forum Name</td>
    <td>Topics/Messages</td>
    <td>Last Post</td>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is the description
      ▶ of the forum. This is another line of descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
      ▶ the description of the forum. This is another line of
      ▶ descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
      ▶ the description of the forum. This is another line of
      ▶ descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
      ▶ the description of the forum. This is another line of
      ▶ descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
</table>
```

В этом варианте разметки (с упрощенным контентом) строки и столбцы создаются в правильном порядке. На рис. 7.4 показаны результаты.

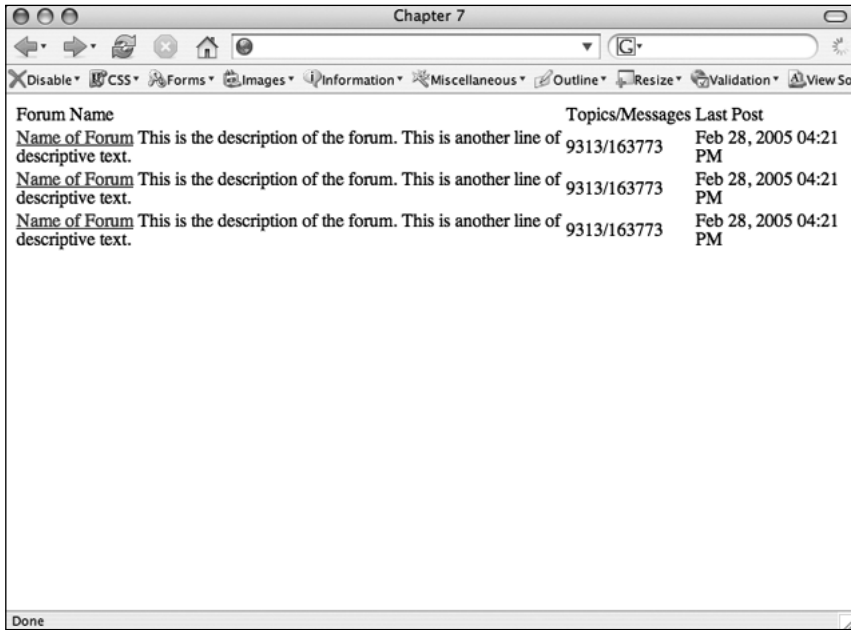


Рис. 7.4. Такой вид наша таблица принимает в браузере Firefox

Когда все элементы находятся на своих местах, можно выделить заголовки столбцов **Forum Name**, **Topics/Messages**, **Last Post** с помощью `<th>`. Элемент `<th>` позволяет назначить заголовки. Дополнительный уровень детализации поможет браузеру или устройству разобраться с организацией таблицы. В качестве бонуса этот уникальный элемент позволяет использовать стили в оформлении заголовков, чтобы назначить им другое оформление с помощью CSS. Нам даже не понадобится дополнительная разметка (например, `class`).

```
<table>
  <tr>
    <th>Forum Name</th>
    <th>Topics/Messages</th>
    <th>Last Post</th>  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
      ▶ the description of the forum. This is another line of
      ▶ descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
      ▶ the description of the forum. This is another line of
```





```

    ▶descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
    ▶the description of the forum. This is another line of
    ▶descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
</table>

```

Теперь браузеры оформят элемент `<th>` по-другому, выделяя полужирным шрифтом и центрируя (рис. 7.5). Внешний вид можно будет изменить позднее с помощью CSS. Сейчас нам важна только структура. Элементы `<th>` идеально подходят для шапки таблицы с точки зрения семантики.

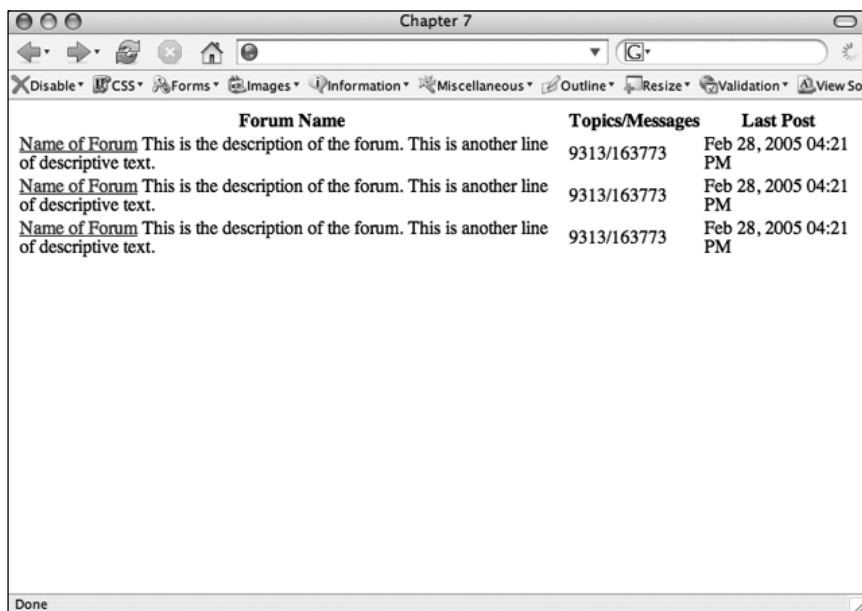


Рис. 7.5. Мы добавили элементы `<th>` для заголовка каждого столбца

ДОБАВЛЯЕМ SCORE

Задавая заголовки для столбцов таблицы, мы поможем пользователям, использующим ридеры, если добавим атрибут `score`. Значение `score` связывает ячейки

с заголовком. В нашем примере мы используем `col`, которое указывает на связь заголовка с ячейками находящегося под ним столбца.

```
<table>
  <tr>
    <th scope="col">Forum Name</th>
    <th scope="col">Topics/Messages</th>
    <th scope="col">Last Post</th>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
      ▶ the description of the forum. This is another line of
      ▶ descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
      ▶ the description of the forum. This is another line of
      ▶ descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
  <tr>
    <td><a href="/forum/">Name of Forum</a> This is
      ▶ the description of the forum. This is another line of
      ▶ descriptive text.</td>
    <td>9313/163773</td>
    <td>Feb 28, 2005 04:21 PM</td>
  </tr>
</table>
```

Для горизонтальной таблицы с заголовками строк, для `scope` мы использовали бы значение `row`.

ДОБАВЛЯЕМ НАЗВАНИЕ

Вернемся к рис. 7.2. Обратите внимание на название, расположенное над таблицей. На сайте Lance Spacerunner название The Films является не текстом, а графическим файлом. Мы также можем использовать изображение, но на моем компьютере нет большой библиотеки «футуристических» шрифтов, поэтому используем текстовый способ создания пуленепробиваемого элемента.

Для разметки названия таблицы подойдет дочерний элемент `<caption>`, предназначенный как раз для этих целей, — `<caption>`. Название таблицы располагается непосредственно под элементом `<table>`. Как правило, браузеры выравнивают название по центру (рис. 7.6).

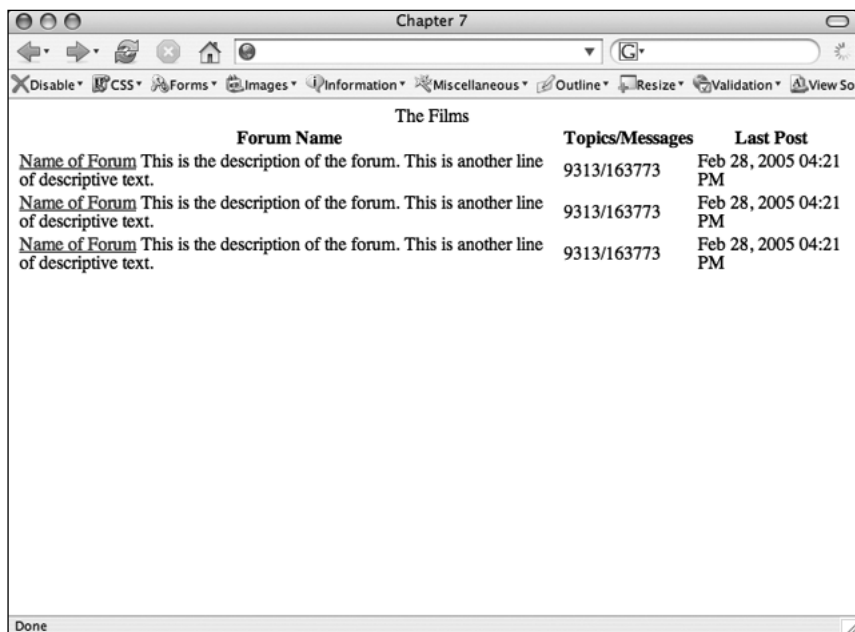


Рис. 7.6. Элемент `<caption>` часто располагается по центру над контентом таблицы

СОВЕТ

Существует множество элементов, которые можно использовать в таблицах, `<thead>`, `<tfoot>`, `<tbody>` позволяют группировать строки и работать с контекстом в длинных разделах таблицы. `<col>` и `<colgroup>` подходят для группировки столбцов, позволяя отображать таблицу не дожидаясь полной загрузки.

Ознакомьтесь со спецификацией W3C HTML5 (www.w3.org/TR/html5/tabular-data.html#tabular-data) или (возможно, так будет проще) прочитайте статью Роджера Йоханссона «Bring on the tables» (www.456bereastreet.com/archive/200410/bring_on_the_tables/), чтобы разобраться с возможностями структурирования таблиц.

Так как наши таблицы относительно просты и невелики, нам не понадобятся дополнительные свойства.

```
<table>
  <caption>The Films</caption>
  <tr>
    <th scope="col">Forum Name</th>
    <th scope="col">Topics/Messages</th>
```

```

        <th scope="col">Last Post</th>
    </tr>
    <tr>
        <td><a href="/forum/">Name of Forum</a> This is
        ▶ the description of the forum. This is another line of
        ▶ descriptive text.</td>
        <td>9313/163773</td>
        <td>Feb 28, 2005 04:21 PM</td>
    </tr>
    <tr>
        <td><a href="/forum/">Name of Forum</a> This is
        ▶ the description of the forum. This is another line of
        ▶ descriptive text.</td>
        <td>9313/163773</td>
        <td>Feb 28, 2005 04:21 PM</td>
    </tr>
    <tr>
        <td><a href="/forum/">Name of Forum</a> This is
        ▶ the description of the forum. This is another line of
        ▶ descriptive text.</td>
        <td>9313/163773</td>
        <td>Feb 28, 2005 04:21 PM</td>
    </tr>
</table>

```

РАССТОЯНИЕ МЕЖДУ ВНЕШНИМИ ГРАНИЦАМИ ЯЧЕЕК

Прежде чем перейти к CSS, давайте еще чуть-чуть позанимаемся разметкой: зададим `cellspacing` для элемента `<table>`. Если обнулить значение этого атрибута (задать значение 0), мы уберем интервалы между ячейками, чтобы задать их позднее с помощью CSS.

```
<table cellspacing="0">
```

Атрибут `cellspacing` — самый надежный способ сбросить стилевое оформление таблицы, но не единственный. Свойство `border-collapse` (CSS) дает аналогичный результат. Если ввести код

```
table {
    border-collapse: collapse;
}
```

элемент `<table>` также будет очищен. Но при этом остается проблема поддержки этого свойства браузерами. Большинство браузеров поддерживают `border-collapse` (IE7, Firefox 3.5 и их более старые версии обрабатывают это свойство не совсем корректно, <http://reference.sitepoint.com/css/border-collapse#compatibilitysection>). Думаю, что для нашего примера `cellspacing` вполне подойдет, поэтому пора перейти к стилям.

ПРИМЕНЯЕМ СТИЛЬ

После создания экономичной разметки таблицы можно перейти к CSS, чтобы заняться оформлением. Нам удалось сократить разметку вдвое, отказавшись от избыточных ячеек и оставив только то, что необходимо для структурирования данных.

Для создания страницы зададим фоновый цвет, определим шрифт и цвет ссылок (по умолчанию):

```
body {
    margin: 0;
    padding: 30px;
    font-family: "Lucida Grande", Arial, sans-serif;
    font-size: small;
    background: #B5B5B5;
}
a { color: #77985C; }
```

ГРАНИЦЫ И ФОН

Создадим 1-пиксельную черную рамку вокруг таблицы, а фоновый цвет сделаем белым.

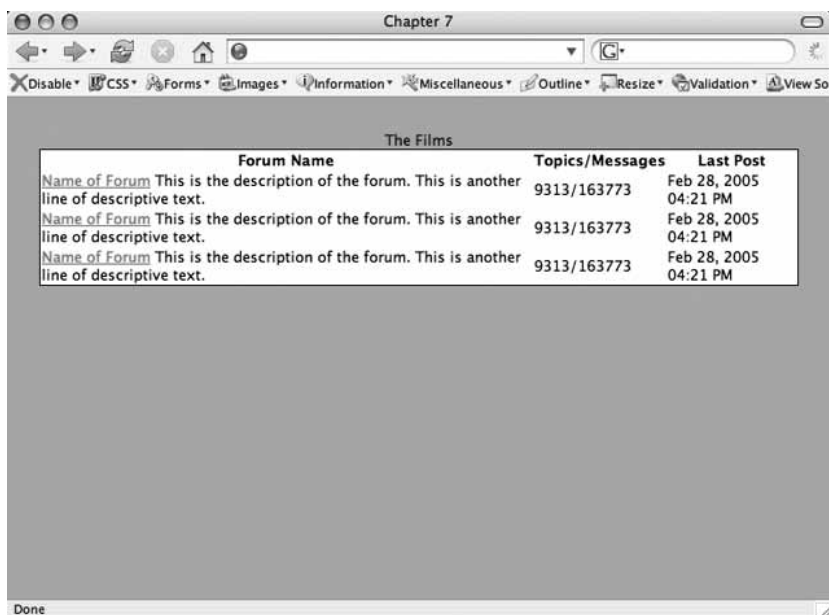


Рис. 7.7. Обратите внимание, что <caption> находится за пределами таблицы, хотя сам элемент задан внутри нее

Пусть ширина таблицы соответствует 100%, чтобы таблица занимала все доступное пространство. В нашем примере ширина таблицы будет равна ширине окна браузера (минус 30 пикселей отступа, который мы указали вокруг элемента `<body>`). На сайте Lance Spacerunner ширина таблицы будет совпадать с шириной контейнера, в котором она располагается (основная колонка контента страницы).

```
table {
  width: 100%;
  border: 1px solid #000;
  background: #FFF;
}
```

На рис. 7.7 показаны результаты.

ОТСТУПЫ И ПРОГРАФКА

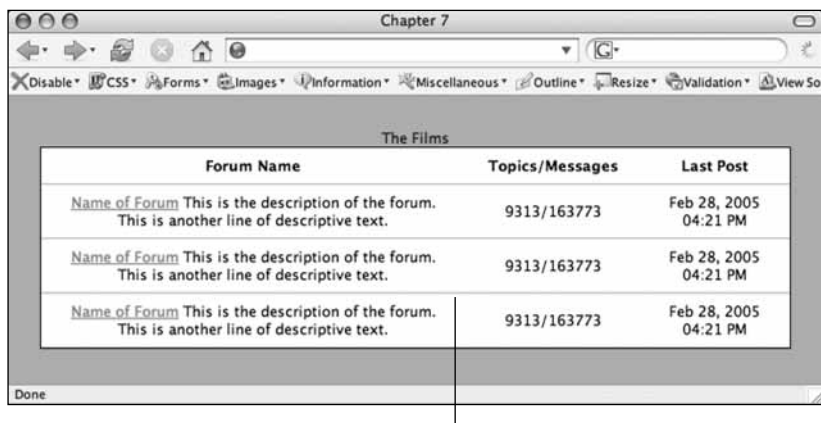
Теперь добавим отступы для элементов `<th>` и `<td>` и нарисуем 1-пиксельную серую линию, разделяющую строки:

```
table {
  width: 100%;
  border: 1px solid #000;
  background: #FFF;
}
table th, table td {
  margin: 0;
  padding: 8px 20px;
  text-align: center;
  border-bottom: 1px solid #B5B5B5;
}
```

На рис. 7.8 показаны результаты добавления 8-пиксельного отступа сверху и снизу, 20-пиксельного отступа слева и справа, а также серые линии на всю ширину таблицы, разделяющие строки. Свойство `border-bottom` позволяет нарисовать непрерывную прямую линию. Этот прием работает только если `cellspacing` равен нулю (мы задали это значение ранее) или если для элемента `<table>` было задано свойство `border-collapse: collapse;`. Мы выровняли текст во всех ячейках по центру, но для столбца **Forum Name** выравнивание по центру потребует отменить.

ПРИМЕЧАНИЕ

Давайте будем использовать только `<th>` и `<td>`, так как эти элементы всегда находятся внутри `<table>`. Однако для упорядочивания CSS-разметки я часто использую формальные селекторы. В нашем случае не нужно задавать ячейки внутри табличных элементов, но это простой способ визуальной группировки кода в таблице стилей.



border-bottom: 1px solid #b5b5b5;

Рис. 7.8. Так как мы обнулили cellpadding, непрерывную линию можно нарисовать с помощью границы элемента

ЗАКАЗНОЕ ВЫРАВНИВАНИЕ

Чтобы выровнять заголовок Forum Name и соответствующий столбец по левому краю, используем псевдоэлемент `:first-child` для отмены настроек `text-align: center;`, которые использовались для элементов `<th>` и `<td>`.

```
table {
    width: 100%;
    border: 1px solid #000;
    background: #FFF;
}
table th, table td {
    margin: 0;
    padding: 8px 20px;
    text-align: center;
    border-bottom: 1px solid #B5B5B5;
}
table th:first-child,
table td:first-child {
    text-align: left;
}
```

ПРИМЕЧАНИЕ

Псевдоэлемент `:first-child` поддерживается IE8+ (но не поддерживается предыдущими версиями). Учитывайте это, когда выбираете стилевое оформление. Если не требуется точное соблюдение дизайна в старых версиях IE, то используйте это свойство не раздумывая. В противном случае вам придется добавлять классы для элементов `<th>` и `<td>`, чтобы настроить стиль.

Давайте применим стиль к первым элементам `<th>` и `<td>` в строках, используя `:first-child`. Это довольно удобный способ назначения стиля конкретным элементам без привлечения классов.

На рис. 7.9 показаны результаты нашей работы. Текст и заголовки столбца `Forum Name` выравниваются по левому краю, а два других столбца — по центру.

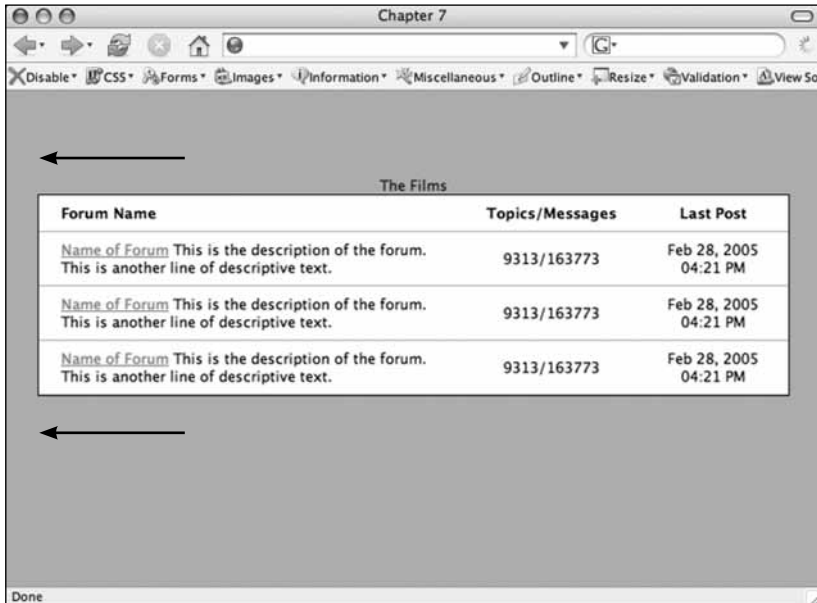


Рис. 7.9. Так как столбец `Forum Name` является исключением, мы выровняли эту ячейку по левому краю с помощью псевдоэлемента `:first-child`

ПРИМЕЧАНИЕ

Safari, Firefox, Chrome, Opera и IE9+ поддерживают псевдоэлемент `:nth-child`. Если стили должны отображаться в IE8 и более старых версиях, вам понадобится альтернативное решение с использованием классов в разметке элементов, которые вы хотите стилизовать.

ЧЕРЕДОВАНИЕ ЦВЕТОВ СТРОК («ЗЕБРА»)

Чтобы создать строки с чередованием цветов заливки («зебры»), использованные в исходном дизайне сайта Lance Spasegunner, мы опять обратимся к нашему старому знакомому `:nth-child`.

Прежде всего для всех элементов `<tr>` зададим серую заливку по умолчанию. Затем для четных строк сделаем фон более светлым, в этом нам поможет псевдо-элемент `:nth-child(even)`:

```
table {
    width: 100%;
    border: 1px solid #000;
    background: #FFF;
}
table th, table td {
    margin: 0;
    padding: 8px 20px;
    text-align: center;
    border-bottom: 1px solid #B5B5B5;
}
table th:first-child,
table td:first-child {
    text-align: left;
}
table tr {
    background: #E6E6E6;
}
table tr:nth-child(even) {
    background: #F1F1F1;
}
```

На рис. 7.10 показаны строки с чередующимися цветами. Теперь страница выглядит гораздо привлекательнее.

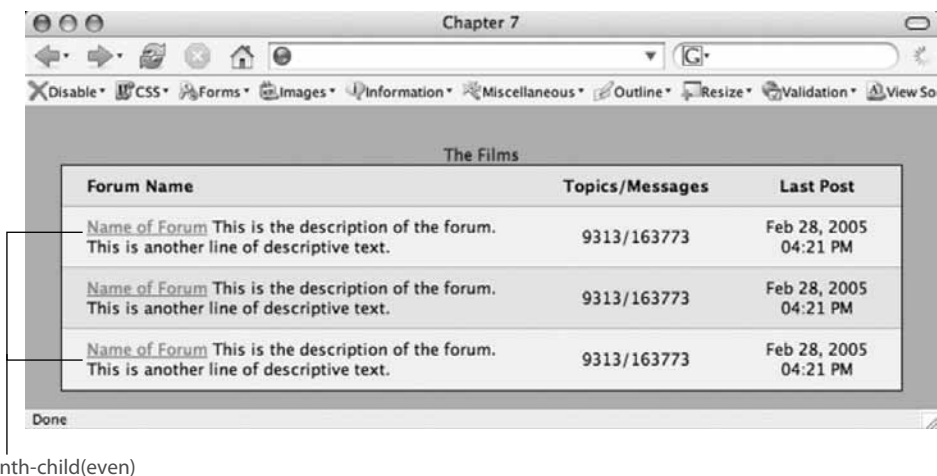


Рис. 7.10. С помощью `:nth-child(even)` мы изменим цвет заливки для четных строк

НОВАЯ СТРОКА БЕЗ ИСПОЛЬЗОВАНИЯ

Для того чтобы разместить название и описание каждого форума на отдельных строках, вместо
 можно указать связи в элементах <td>, а затем — свойство `display: block;`. Таким образом, описание, идущее после названия, будет располагаться на следующей строке, но дополнительной разметки не понадобится. Кроме того, выделим ссылки полужирным шрифтом:

```
table {
    width: 100%;
    border: 1px solid #000;
    background: #FFF;
}
table th, table td {
    margin: 0;
    padding: 8px 20px;
    text-align: center;
    border-bottom: 1px solid #B5B5B5;
}
table th:first-child,
table td:first-child {
    text-align: left;
}
table tr {
    background: #E6E6E6;
}
table tr:nth-child(even) {
    background: #F1F1F1;
}
table td a {
    display: block;
    font-weight: bold;
}
```

ПРИМЕЧАНИЕ

Указывая для ссылки `display: block;`, как сделано в этом фрагменте кода, мы позволяем полю ссылки расширяться по горизонтали до размеров родительского контейнера. Иными словами, активной становится вся строка (независимо от того, как был выровнен текст), а не только текст ссылки. Рассматривайте этот вариант как улучшение юзабилити за счет увеличения поля ссылки (почитайте <http://1976design.com/blog/archive/2004/09/07/link-presentation-fitts-law/>, чтобы получить представление о том, как размер поля ссылки влияет на юзабилити гиперссылок). Можно долго спорить о размере поля гиперссылки, но учитывайте и этот фактор, когда выбираете метод разделения ячейки на строки.

Как видно на рис. 7.11, каждая ссылка теперь имеет полужирное начертание и находится на отдельной строке. Команда `display: block;` позволяет отделить элементы списка линиями — замечательная идея, если нельзя использовать CSS. Я хочу позаботиться об устройствах, которые не поддерживают CSS, даже в этом случае ссылки и описания будут корректно отображаться.

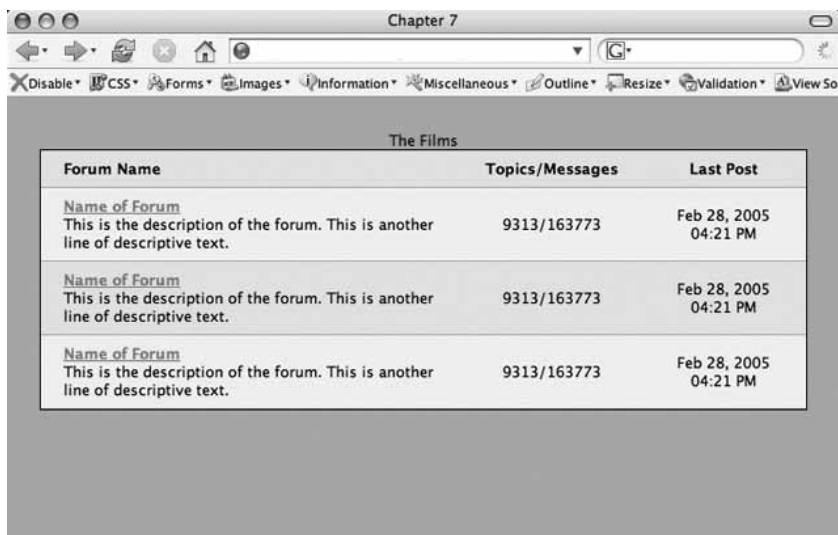


Рис. 7.11. В нашем случае можно использовать свойство `display: block;`, поскольку в каждой ячейке находится не более одной ссылки. Если ссылок было много, каждая оказалась бы в отдельной строке

ЦВЕТ ЗАГОЛОВКА

Как я говорил ранее, один из побочных эффектов использования заголовков таблицы (`<th>`) — это дополнительный контроль над стилевым оформлением этих элементов, который не требует дополнительной разметки, например, классов.

Если рассматривать исходный сайт Lance Spacerunner, то видно, что текст в шапке таблицы более светлого оттенка, чем остальной текст таблицы. Мы можем легко подправить наше оформление, добавив правило, изменяющее цвет всех элементов `<th>`:

```
table {
    width: 100%;
    border: 1px solid #000;
    background: #FFF;
}
table th, table td {
```

```

margin: 0;
padding: 8px 20px;
text-align: center;
border-bottom: 1px solid #B5B5B5;
}
table th {
color: #999;
}
table th:first-child,
table td:first-child {
text-align: left;
}
table tr {
background: #E6E6E6;
}
table tr:nth-child(even) {
background: #F1F1F1;
}
table td a {
display: block;
font-weight: bold;
}

```

На рис. 7.12 показаны результаты работы. Теперь заголовки в шапке таблицы окрашены в серый цвет вместо выбранного по умолчанию черного.

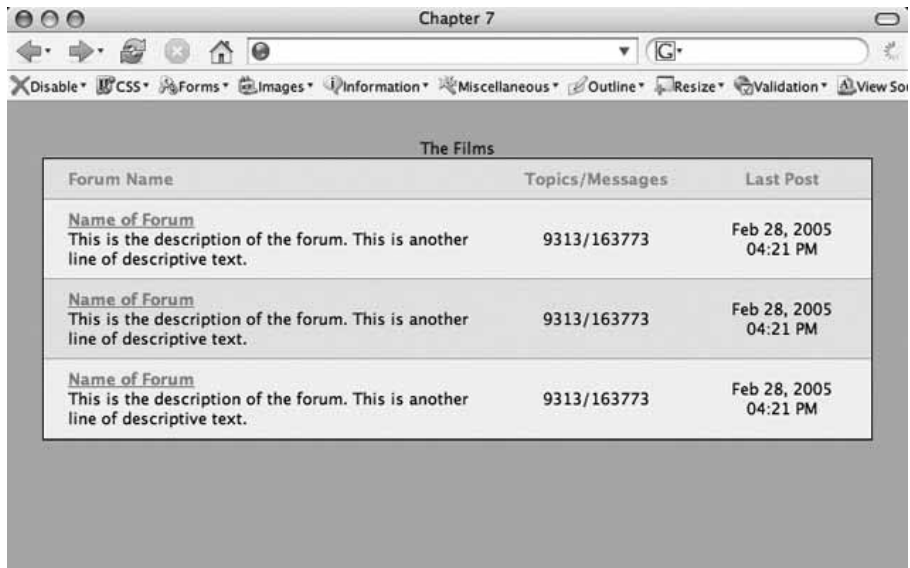


Рис. 7.12. Нам нужно было изменить цвет заголовков столбцов. Это делалось очень просто, так как мы использовали элементы `<th>` для структурирования таблицы

ОФОРМЛЕНИЕ <caption>

Теперь настала очередь элемента `<caption>`. Зададим белый фон и отступы с трех сторон (кроме нижнего, так как он уже является частью `<table>`).

Изначально мы решили использовать `<caption>` для заголовка таблицы, как того требует семантика. В данном случае это был наиболее разумный выбор, но браузеры WebKit (Safari и Chrome) обрабатывают элемент `<caption>` не так, как другие браузеры. В некоторых случаях результаты могут быть непредсказуемыми. В нашем примере `<caption>` окажется более узким, чем `<table>`, несмотря на то что `<caption>` находится внутри `<table>`. Это является следствием разных интерпретаций спецификаций со стороны браузера. Так что сначала оцените, какой элемент окажется более предпочтительным в конкретной ситуации: `<caption>` (оптимальный выбор) или другой элемент заголовка.

```
table {
    width: 100%;
    border: 1px solid #000;
    background: #FFF;
}
table caption {
    margin: 0;
    padding: 8px 20px;
    text-align: left;
    border: 1px solid #000;
    border-bottom: none;
    background: #FFF;
}
table th, table td {
    margin: 0;
    padding: 8px 20px;
    text-align: center;
    border-bottom: 1px solid #B5B5B5;
}
table th {
    color: #999;
}
table th:first-child,
table td:first-child {
    text-align: left;
}
table tr {
    background: #E6E6E6;
}
table tr:nth-child(even) {
    background: #F1F1F1;
}
table td a {
```

```
display: block;
font-weight: bold;
}
```

По умолчанию название таблицы располагается по центру. Поэтому помимо свойств `padding`, `background` и отступов `border` со всех сторон, кроме нижней, мы указали выравнивание текста по левому краю (рис. 7.13). И снова вернемся к оригинальному виду сайта Lance Spacerunner. Заголовок сайта представлял собой изображение, которое находилось в элементе `<caption>`. А в нашем случае мы использовали текст вместо изображения.

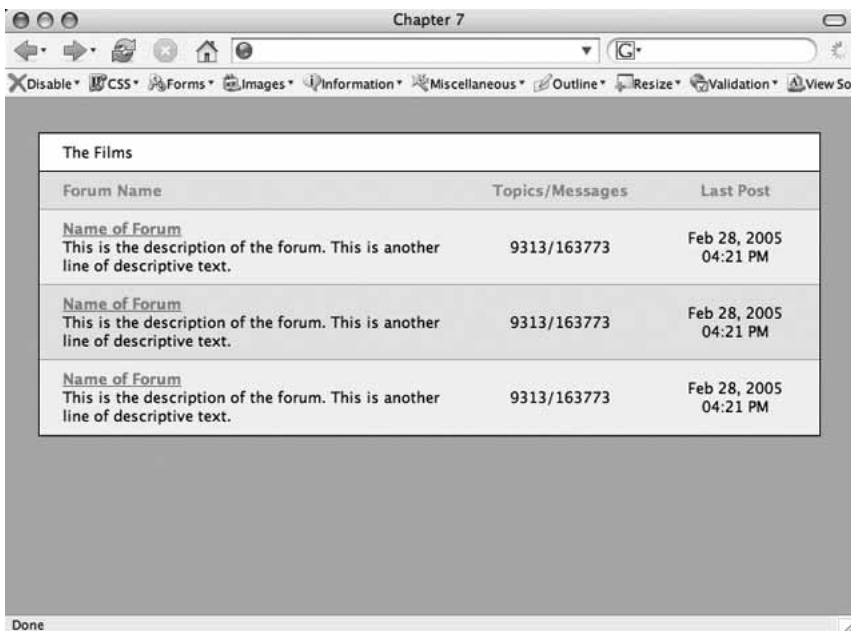


Рис. 7.13. В Firefox название смотрится симпатично

ТЕНЬ СО СМЕЩЕНИЕМ

Для создания тени со смещением на сайте Lance Spacerunner использовались дополнительные столбец и строка, в которые помещены изображения, формирующие эффект тени (рис. 7.14). Мы хотим добиться такого же результата с помощью свойства `box-shadow` (CSS3). При этом не понадобятся лишние ячейки и изображения.

Чтобы тень находилась позади `<caption>` и остальной части таблицы, нам понадобится еще один контейнер, в котором будет размещена таблица.

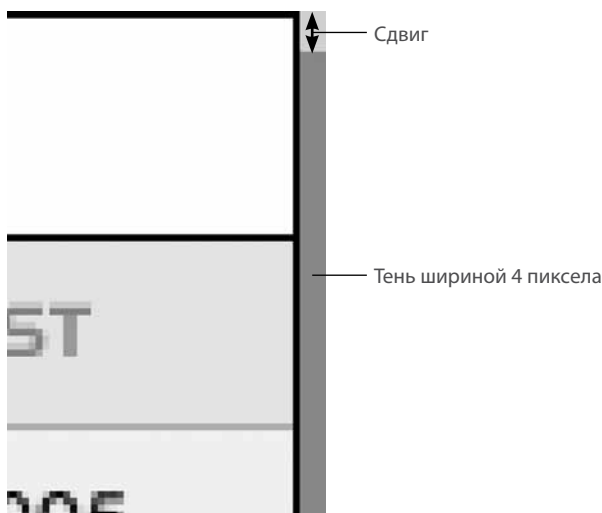


Рис. 7.14. Тень можно создать с помощью дополнительного внешнего элемента вокруг таблицы

Таким образом, наша разметка немного изменится, мы добавим контейнер `<section>`, в него и поместим таблицу. Начнем с определения класса контейнера, который можно будет использовать и для других таблиц на странице:

```
<section class="forums">
  <table cellspacing="0">
    ...
  </table>
</section>
```

Теперь определим для нового элемента `<section>` свойство `box-shadow` и его же версию с префиксами разработчиков. На рис. 7.15 видно, что таблица сдвинута относительно тени на 4 пиксела вверх и влево. Мы можем воссоздать этот эффект, указав соответствующие параметры `box-shadow` (по 4 пиксела сверху и слева и нулевое размытие, чтобы тень была сплошной):

```
section.forums {
  -webkit-box-shadow: 4px 4px 0 #919191;
  -moz-box-shadow: 4px 4px 0 #919191;
  box-shadow: 4px 4px 0 #919191;
}
```

На рис. 7.16 показан завершенный дизайн (в Firefox) со сдвигом тени относительно таблицы. Браузеры, не поддерживающие `box-shadow` (IE8 и более старые версии), не будут отображать эту тень.

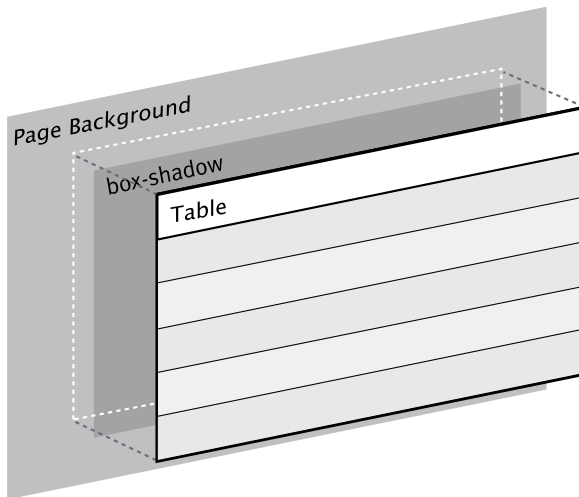


Рис. 7.15. Сдвиг тени мы создадим с помощью box-shadow

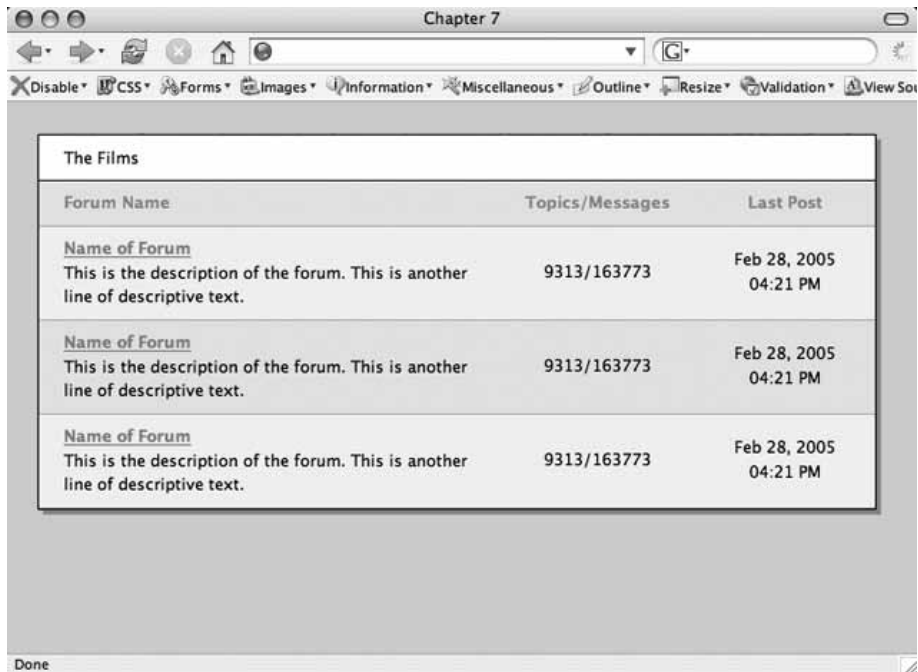


Рис. 7.16. Завершенный дизайн тени, созданный без дополнительных изображений

ПРЕИМУЩЕСТВА ПУЛЕНЕПРОБИВАЕМОГО ПОДХОДА

Мы начали с нуля — с семантической структуры — и перестроили таблицу, используя только необходимые элементы. Мы сделали таблицу более доступной, сократив количество строк кода. Кроме того, мы смогли полностью воссоздать оформление таблицы без использования изображений, разделить разметку и дизайн, перенести оформление в CSS. Теперь можно легко обновить дизайн и (или) данные, так как нам не понадобится просматривать сотни лишних ячеек таблицы, которые используются исключительно в оформительских целях.

The image displays three variations of a forum table, each with a different container width. The table structure is consistent across all three, with three columns: Forum Name, Topics/Messages, and Last Post. The data rows are identical, but the visual presentation changes based on the container size.

The Films		
Forum Name	Topics/Messages	Last Post
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM

The Films		
Forum Name	Topics/Messages	Last Post
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM

The Films		
Forum Name	Topics/Messages	Last Post
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM
Name of Forum This is the description of the forum. This is another line of descriptive text.	9313/163773	Feb 28, 2005 04:21 PM

Рис. 7.17. Изменение размера контейнера таблицы не влияет на ее дизайн

Сама таблица стала резиновой, как по высоте, так и по ширине. Мы отказались от использования пикселей при задании размеров, а так как изображения больше не нужны, наш дизайн можно разместить в любом контейнере и как угодно смасштабировать (рис. 7.17).

РЕЗЮМЕ

При создании таблиц с данными, для которых определены стили, нужно отказаться от использования избыточной разметки. Взяв за основу значимую структуру таблицы (только необходимые элементы), вы сократите количество кода и сможете оформить таблицу с помощью CSS.

Рамки, фоны и цвета можно перенести в таблицу стилей. Это сделает разметку чистой и экономичной, а результат станет также доступен для любых устройств.

При работе с таблицами:

- «сожмите» таблицу с помощью `<table cellpadding="0">` (или свойства `border-collapse`); задайте все границы, фоны и отступы в таблице стилей;
- используйте элемент `<caption>` для заголовка таблицы; помните, что браузеры WebKit по-своему интерпретируют заголовки `<caption>`;
- элементы `<th>` шапки таблицы позволят создать более эффективную структуру и использовать стилевое оформление ячеек с помощью CSS;
- создавайте прографку с помощью границ элементов `<th>` и (или) `<td>`;
- добавляйте фоновые цвета заливки строк, используя стилевое оформление элементов `<tr>`; реализуйте чередование цвета заливки строк («зебру») с помощью псевдоэлемента CSS3 `:nth-child`, не прибегая к классам в разметке.

ГЛАВА 8

РЕЗИНОВЫЕ И ЭЛАСТИЧНЫЕ МАКЕТЫ



Экспериментируйте с макетом страницы, чтобы он мог растягиваться и сжиматься

Вполне естественно заботиться о гибкости страницы и ее компонентов. В предыдущих главах книги речь шла об отдельных компонентах, а теперь я предлагаю проанализировать макет в целом. Мы научимся использовать CSS для создания многоколоночных страниц, которые могут расширяться или сжиматься, подстраиваясь под размер экрана, окна или контента.

Прежде всего хочу сказать, что не собираюсь полемизировать по поводу того, что лучше: эластичные или фиксированные макеты? Думаю, что каждому варианту найдется свое применение. Выбор зависит от множества факторов, которые являются уникальными для каждого проекта и зависят от конкретных условий. Макет не выбирают раз и навсегда, выбор определяется сайтом.

В этой главе я предложу вам стратегии разработки резиновой и эластичной верстки, которая станет прекрасным дополнением к вашему арсеналу пуленепробиваемых методов. Если вы научитесь эффективно использовать ее, то это только добавит вашим продуктам гибкости.

Резиновые (иногда их еще называют плавающими) макеты можно создать не только с помощью CSS. Вы можете создавать резиновые макеты и с помощью таблиц. Однако таблица стилей предоставляет вам дополнительные преимущества, о чем мы и поговорим в этой главе.

Ширина колонок в эластичных макетах задается в em. Как и некоторые из уже рассмотренных примеров, такие макеты расширяются или сжимаются, подстраиваясь под размер шрифта, обеспечивая «масштабируемость» дизайна.

ОБЩЕПРИНЯТЫЙ ПОДХОД

Резиновая многоколоночная верстка не является исключительной прерогативой CSS. Макеты, которые расширяются или сжимаются, подстраиваясь под размер окна браузера, можно создавать с помощью таблиц. Как правило, дизайнер думает о колонках, как о наборе ячеек таблицы.

На рис. 8.1 показана распространенная сетка страницы, в которой заголовок находится в верхней части и занимает всю ширину страницы, под ним расположены две колонки с контентом, под которыми находится нижний колонтитул, занимающий всю ширину страницы.



Рис. 8.1. Вам часто встречаются двухколоночные страницы с подобной структурой

Раньше считалось, что таблица является идеальным решением для подобной структуры, а атрибут `colspan` позволяет объединить ячейки, чтобы заголовок и нижний колонтитул использовали всю ширину колонок с контентом. Разметка в общих чертах имеет вид:

```
<table>
  <tr>
    <td colspan="2">header</td>
  </tr>
  <tr>
```





```

        <td>content</td>
        <td>sidebar</td>
    </tr>
    <tr>
        <td colspan="2">footer</td>
    </tr>
</table>

```

Раньше большинства дизайнеров и разработчиков испытывали неудобства с самого начала. После задания базовой структуры нужно было вложить в каждую ячейку дополнительные таблицы, управляющие границами и отступами. Для создания структуры и оформления страницы использовалось огромное количество разметки.

Чтобы сделать макет эластичным, можно задать ширину ячеек таблицы в процентах, тогда макет будет расширяться в зависимости от размера окна браузера:

```

<table width="100%">
    <tr>
        <td colspan="2">header</td>
    </tr>
    <tr>
        <td width="70%">content</td>
        <td width="30%">sidebar</td>
    </tr>
    <tr>
        <td colspan="2">footer</td>
    </tr>
</table>

```

Ширина таблицы соответствует 100%, при этом ширина каждой колонки меняется в зависимости от фактического размера таблицы. В данном случае слева расположена более широкая колонка. Изменение размера окна браузера будет менять размер макета, а соотношение ширины колонок останется неизменным.

Таблицы CSS действительно можно использовать для разработки макетов. Более того, таблицы CSS можно использовать для разработки резиновых макетов, которые меняют размеры в соответствии с размерами окна браузера. Однако существует способ проще и эффективнее. Давайте поговорим о том, почему табличный метод не является пуленепробиваемым.

Уязвимые места

Одна из основных проблем табличного макета — это смешение в единое целое контента и оформления. Рамки, GIF-разделители и изображения встраиваются непосредственно в разметку, то есть прячутся между фрагментами значимого

контента. Это приводит к тому, что не все браузеры могут отображать такие страницы корректно. У пользователей приложений для чтения с экрана, текстовых браузеров и устройств с небольшими экранами возникнут проблемы.

ИЗБЫТОЧНОСТЬ КОДА

Такое смешение предполагает большой объем кода. Количество разметки, необходимой для создания привлекательного макета с помощью вложенных таблиц, просто потрясает. Лишние ячейки таблицы используются для задания средников, рамок и других элементов оформления страницы. CSS позволяет сделать разметку минимальной, а правила отображения контента переместить в таблицы стилей. Такая страница сразу становится доступной для нетрадиционных устройств и приложений, а сам сайт станет более дружелюбным по отношению к поисковым системам (и это еще один плюс CSS).

КОШМАРНЫЙ СОН ТЕХПОДДЕРЖКИ

Весь этот сложный код, о котором я говорил, только добавляет проблем для службы поддержки. Изменение дизайна страницы, построенной на базе таблиц, потребует много дополнительной работы, связанной с поиском нужного кода среди огромного количества ячеек таблицы и избыточной разметки. Это раздражает или и вовсе делает работу бессмысленной, поскольку иногда бывает проще создать страницу заново.

НЕОПТИМАЛЬНЫЙ ПОРЯДОК РАЗМЕЩЕНИЯ КОНТЕНТА

Другим недостатком макетов на базе таблиц является порядок, в котором контент отображается в текстовых браузерах и приложениях для чтения с экрана. Давайте рассмотрим трехколоночный макет. Мы знаем, что порядок разметки всегда будет следующим: левая колонка, средняя колонка, а затем правая колонка (рис. 8.2). Так работают таблицы.

Пользователи текстовых браузеров или приложений для чтения с экрана увидят контент только в таком порядке. Однако важная информация может находиться не в первой (левой) колонке, а, например, в средней, которая окажется где-то далеко внизу. При использовании макета на базе таблиц невозможно расставить приоритеты отображения контента, которые позволят сразу увидеть самый важный контент в альтернативных браузерах и приложениях. Существует только один порядок отображения для всех устройств, что заставляет пользователей текстовых браузеров и приложений для чтения с экрана добираться до сути сквозь дебри менее важной информации.

К счастью, верстка на базе CSS позволяет создавать оптимальную структуру контента, размещая в исходном коде документа информацию в одном порядке,

а отображая в другом. Кроме того, разделив контент и визуализацию, мы используем гораздо меньше кода. Так что давайте перейдем к резиновой многоколоночной верстке на базе CSS.

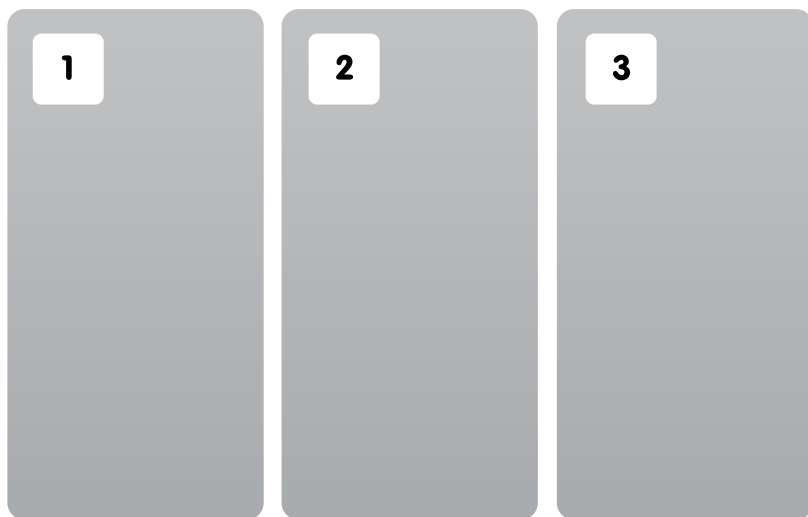


Рис. 8.2. Неоптимальный порядок размещения контента: левая (1), средняя (2), правая (3) колонки

ПУЛЕНЕПРОБИВАЕМЫЙ ПОДХОД

Хотя в качестве пуленепробиваемого примера мы займемся гибкой резиновой версткой, не могу не отметить, что фиксированные макеты на базе CSS имеют не меньше достоинств, чем их нефиксированные «коллеги». Основное различие в том, что резиновые макеты могут расширяться или сжиматься в соответствии с размерами окна браузера, подчиняясь желаниям пользователя. Это особенно удобно, если размеры экранов значительно различаются (например, в мобильных устройствах).

Давайте проделаем все необходимые шаги, чтобы создать резиновый макет на базе CSS и разобраться, почему мы поступаем именно так, а не иначе. И начнем с разработки простого двухколоночного дизайна с заголовком и нижним колонтитулом.

СТРУКТУРА РАЗМЕТКИ

Вместо использования таблицы для разработки макета многоколоночной страницы, мы используем контейнеры `<div>` и другие семантические элементы HTML5,

позволяющие разделить контент на фрагменты. Давайте при разработке учтем оптимальный порядок отображения контента (это будет еще важнее, когда мы приступим к трехколоночному дизайну).

В основе простого макета из двух колонок может лежать следующая разметка:

```
<div id="wrap">
  <header role="banner">
    <h1>Header Goes Here</h1>
  </header>
  <div id="content" role="main">
    ... content goes here ...
  </div>
  <div id="sidebar" role="complementary">
    ... sidebar goes here ...
  </div>
  <footer role="contentinfo">
    ... footer goes here ...
  </footer>
</div> <!-- end #wrap -->
```

СОВЕТ

Обратите внимание, что я снова использую WAI-ARIA для добавления дополнительной семантики к основным разделам сайта. Позднее свойство `role` также будет использоваться как стилевая зацепка.

Вот и все. Вряд ли вы сможете сделать проще. Порядок имеет смысл: заголовок, контент, боковая панель и нижний колонтитул. Мы мыслим в понятиях страницы без учета CSS. Пока все идет нормально.

Я добавил окружение `<div>`, в будущем оно нам еще пригодится. Каждый раз, когда я верстаю макет на основе CSS, я начинаются с контейнера `<div>`.

Колонки: ПЛАВАЮЩИЕ ПРОТИВ ФИКСИРОВАННЫХ

Одним из способов создания колонок с помощью CSS является абсолютное позиционирование, то есть использование координат `x` и `y` для размещения элементов в конкретных местах экрана. Главный недостаток абсолютного позиционирования — невозможность очистки нижнего колонтитула от нижней части всех колонок.

Две колонки (основная колонка контента и боковая панель) можно создать с указанием точных размеров, задав контейнеру контента `<div>` отступы слева и справа, чтобы оставить достаточно места для боковой панели. При абсолютном

позиционировании можно организовать боковую панель, задав ширину, приблизительно равную отступу колонки контента и разместив ее вверху справа (рис. 8.3).

Окно браузера

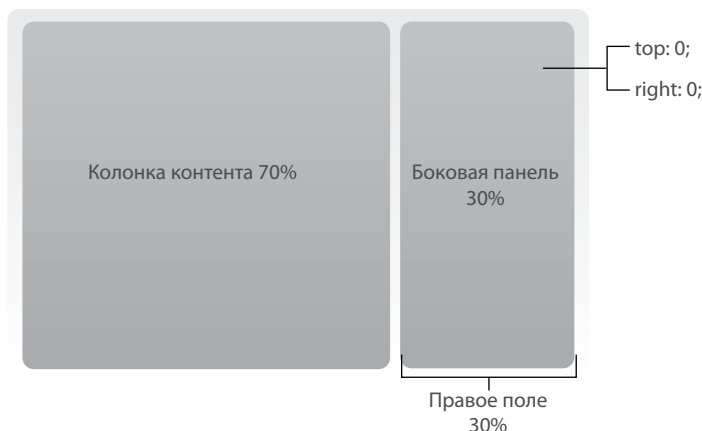


Рис. 8.3. При позиционировании правая боковая панель заходит на поле колонки контента

Например, для этого мы можем задать следующие правила CSS:

```
#content {
    width: 70%;
    margin-right: 30%;
}
#sidebar {
    position: absolute;
    top: 0;
    right: 0;
    width: 30%;
}
```

Код задает внешнее поле, равное ширине боковой колонки, и позиционирует ее с помощью свойства `position: absolute;`. Данный подход достаточно логичен и понятен, особенно для тех, кто только учится верстать с помощью CSS. Однако при использовании этого метода возникает одна проблема: невозможно очистить элементы, размещенные с использованием абсолютного позиционирования.

Если боковая панель окажется длиннее колонки контента, она перекроет все расположенные ниже элементы страницы (в нашем случае, нижний колонтитул) (рис. 8.4).

Это связано с тем, что свойство `position: absolute;` извлекает элемент из системы выравнивания и обтекания. Расположение и размеры данного элемента

перестают оказывать влияние на соседние элементы, что мешает корректной очистке нижнего колонтитула, расположенного под нашими колонками.



Рис. 8.4. Длинная колонка, размещенная с использованием абсолютного позиционирования, может перекрывать другие объекты

Такое отсутствие гибкости при создании многоколоночных макетов заставляет хитрых дизайнеров использовать свойство `float`. Так как плавающие элементы можно очистить (как мы узнали в главе 4 «Креативное перетекание»), они становятся лучшим инструментом, позволяющим управлять макетом с колонками. Стандарт CSS3 предлагает и другие способы создания модульных макетов с помощью нового синтаксиса, но, к несчастью, они пока не поддерживаются большинством браузеров в полном объеме. А такая поддержка критически важна для макета. Поэтому мы и будем использовать `float`.

ПРИМЕНЯЕМ СТИЛИ

Так как для создания колонок мы решили использовать плавающие элементы, нам нужно разделить страницу на две части: колонку контента и боковую панель, указав процентное соотношение размеров. В нашем случае 70% займет колонка контента и 30% — боковая панель.

Давайте на начальном этапе добавим фоновую заливку для заголовка, боковой колонки и нижнего колонтитула. Это немного упростит визуальное разделение компонентов при дальнейшей работе.

```
header[role="banner"] {
    background: #666;
}
#sidebar {
    background: #999;
}
```





```
footer[role="contentinfo"] {
    background: #DDD;
}
```

На рис. 8.5 показаны результаты. (Фоновые заливки заголовка, боковой колонки и нижнего колонтитула окрашены в разные цвета.)

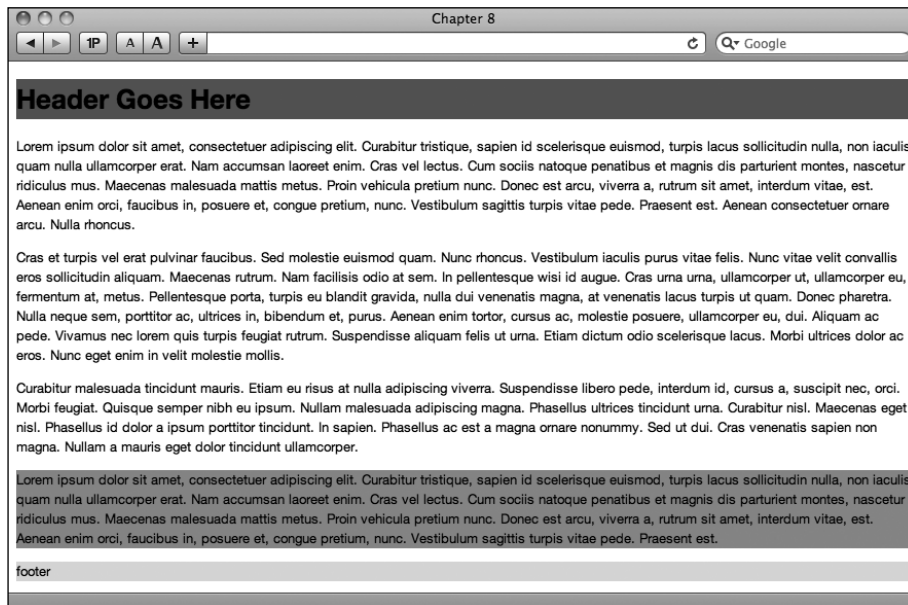


Рис. 8.5. Так макет выглядит до того, как мы сделаем каждую колонку плавающей

Теперь зададим ширину 70% для колонки контента и 30% — для боковой колонки. Затем прижмем колонку контента к левому краю, а боковую колонку — к правому. Таким образом, колонки окажутся друг напротив друга.

```
header[role="banner"] {
    background: #666;
}
#content {
    float: left;
    width: 70%;
}
#sidebar {
    float: right;
    width: 30%;
    background: #999;
}
```

```
footer[role="contentinfo"] {
    background: #DDD;
}
```

На рис. 8.6 показано, что происходит, когда колонка контента и боковая колонка занимают свои места. Обратите внимание на то, что нижний колонтитул «застревает» в плавающих контейнерах, расположенных над ним. Это происходит, так как нижний колонтитул идет в разметке после боковой колонки. Кроме того, нижний колонтитул не имеет конкретной ширины, поэтому его фоновая заливка пытается занять всю ширину контейнера.

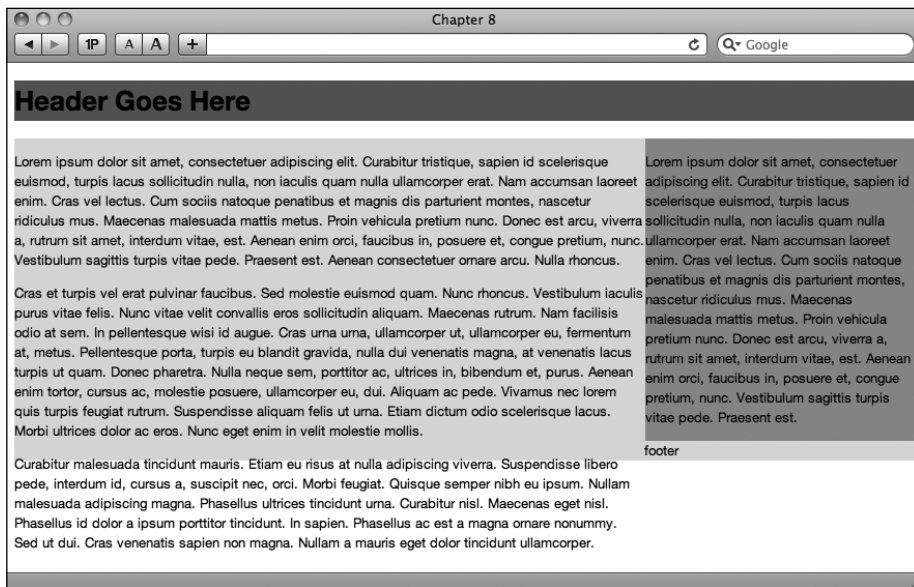


Рис. 8.6. Нижний колонтитул пересекается с плавающими контейнерами, определенными до него

Для исправления ситуации нам придется очистить нижний колонтитул от влияния плавающих элементов, которые находятся перед ним. Это гарантирует, что нижний колонтитул будет находиться под двумя колонками, независимо от размеров обеих колонок. Такое решение будет пуленепробиваемым.

```
header[role="banner"] {
    background: #666;
}
#content {
    float: left;
    width: 70%;
}
```





```
#sidebar {
    float: right;
    width: 30%;
    background: #999;
}
footer[role="contentinfo"] {
    clear: both;
    background: #DDD;
}
```

На рис. 8.7 видно, что с дополнительной очисткой макет принимает нужный вид.

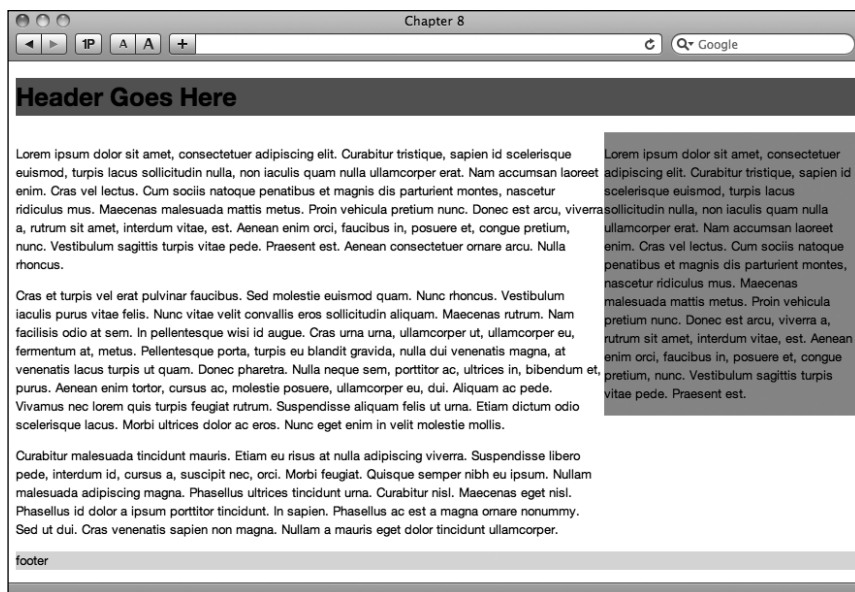


Рис. 8.7. После того как мы очистили плавающие элементы, нижний колонтитул всегда будет находиться под двумя колонками

Вот мы и получили гибкий макет на базе CSS. Возможно, он выглядит не особо привлекательно, но это только основа. Если мы изменим размер окна, ширина колонок также изменится, но их пропорции сохранятся — 70/30 (рис. 8.8).

Обе колонки будут менять ширину в зависимости от ширины окна браузера. Так и должен работать макет на базе CSS — пользователь получает контроль над размерами.

Мы построили прочный фундамент. Теперь пора поговорить о более сложных нюансах резиновой верстки.

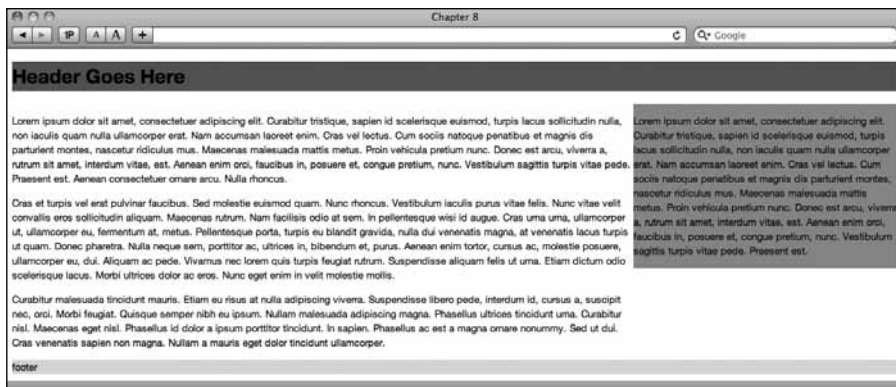
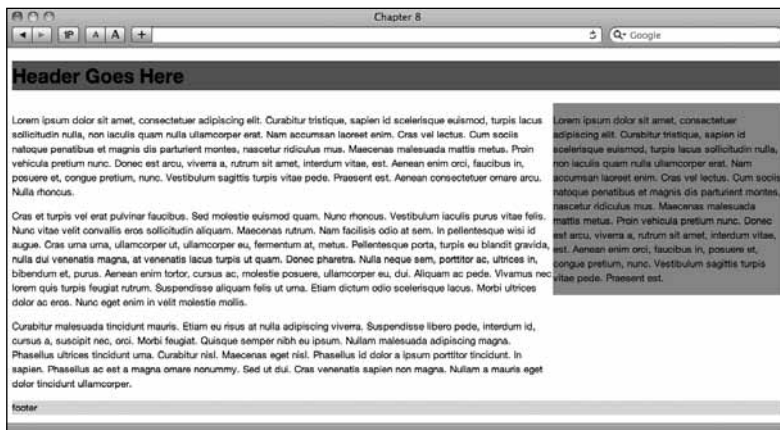
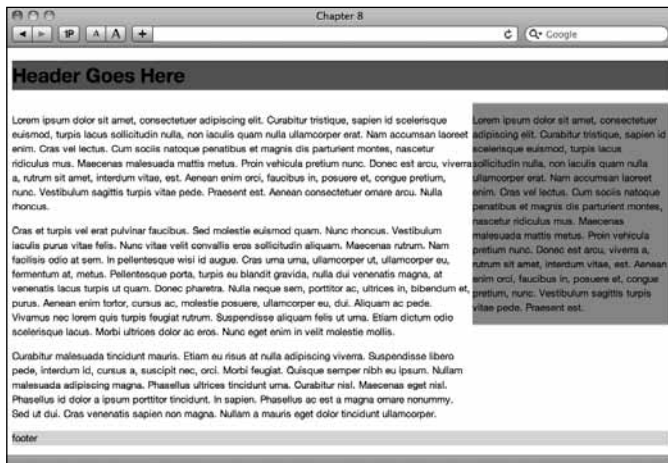


Рис. 8.8. Так поведет себя плавающий макет при трех значениях ширины окна, пропорция 70/30 останется неизменной

СРЕДНИКИ

Термин «средник» существует довольно давно и в полиграфии означает пространство между колонками текста (рис. 8.9). При работе с резиновыми колонками управлять этим расстоянием значительно сложнее. В фиксированном дизайне мы имеем дело с конкретной шириной в пикселах, что упрощает расчет ширины средника.



Рис. 8.9. Средник — это просвет между колонками текста

В резиновых колонках можно пойти двумя путями: задать ширину средника в процентах (так же как и ширину колонок) или использовать дополнительные контейнеры `<div>` для управления наружными и внутренними полями, не привязывая их к ширине колонок. Несомненно, второй подход не выглядит оптимальным с точки зрения экономичной разметки, хотя и предоставляет дополнительный уровень контроля, который может пригодиться при создании сложного оформления. Мы еще вернемся к этой теме позднее.

Процентное значение размера средника можно, например, задать следующим образом: назначить правое поле для колонки контента, а затем вычесть это значение из общей ширины колонки. Результирующая ширина должна составить 100% (рис. 8.10).

```
header[role="banner"] {  
    background: #666;  
}  
#content {  
    float: left;  
    width: 65%;
```

```

    margin-right: 5%;
  }
#sidebar {
  float: right;
  width: 30%;
  background: #999;
}
footer[role="contentinfo"] {
  clear: both;
  background: #DDD;
}

```

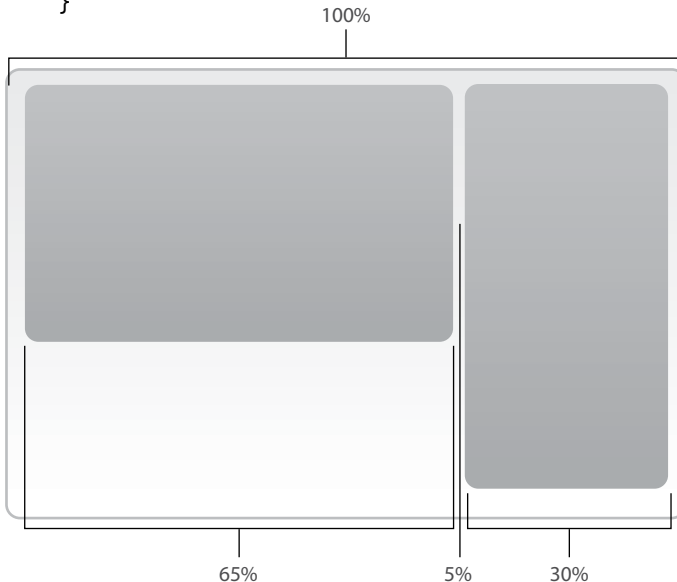


Рис. 8.10. Распределение в процентном соотношении размеров колонок и отступов для двухколоночного макета

В модифицированных правилах мы уменьшили ширину контента на 5% и добавили это же значение как правое поле, чтобы создать свободное пространство между колонками. На рис. 8.11 можно увидеть, как повлияло на оформление добавление средника.

При задании ширины средника в процентах не забудьте, что она будет изменяться в зависимости от ширины окна: будет становиться меньше при сужении экрана и больше — при его увеличении. В некоторых случаях это может стать проблемой. Например, если оформление рамок или фона колонок требует фиксированного расстояния между ними. В этих случаях лучше использовать контейнер второго уровня `<div>`, об этом мы узнаем позднее в этой главе. Однако если вам требуется простота, то наиболее приемлемым способом будет задание ширины в процентах.

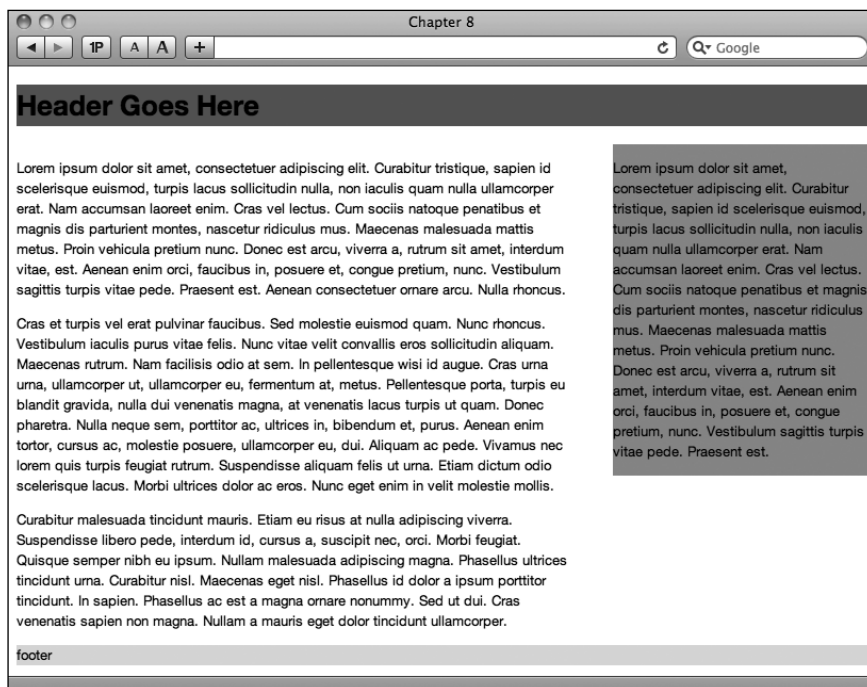


Рис. 8.11. При изменении ширины окна изменится и просвет между колонками

Отступы в колонках

Аналогичная ситуация возникает, когда вы задаете отступы в колонках, ширина которых указана в виде процентного значения. Если вы зададите отступы не в процентах, то общая ширина может легко стать больше или меньше, чем нужно. Такие проблемы вынудят дизайнера отказаться от резиновой верстки.

Давайте сделаем внутреннее поле боковой колонки равным 20 пикселям, чтобы добавить воздуха в наш дизайн:

```
header[role="banner"] {
    background: #666;
}
#content {
    float: left;
    width: 65%;
    margin-right: 5%;
}
#sidebar {
    float: right;
```

```
width: 30%;
padding: 20px;
background: #999;
}
footer[role="contentinfo"] {
clear: both;
background: #DDD;
}
```

В макете с фиксированной шириной мы можем просто уменьшить фиксированную ширину колонки с обеих сторон на 20 пикселей. Однако в резиновой верстке мы используем проценты, и нет способа задать ширину «30% минус 40 пикселей». Ширина колонки становится больше 30%, т. е. колонка перестает помещаться рядом с колонкой контента `<div>`, она опускается на ближайшее свободное поле — под колонку контента (рис. 8.12). Неудачное решение.

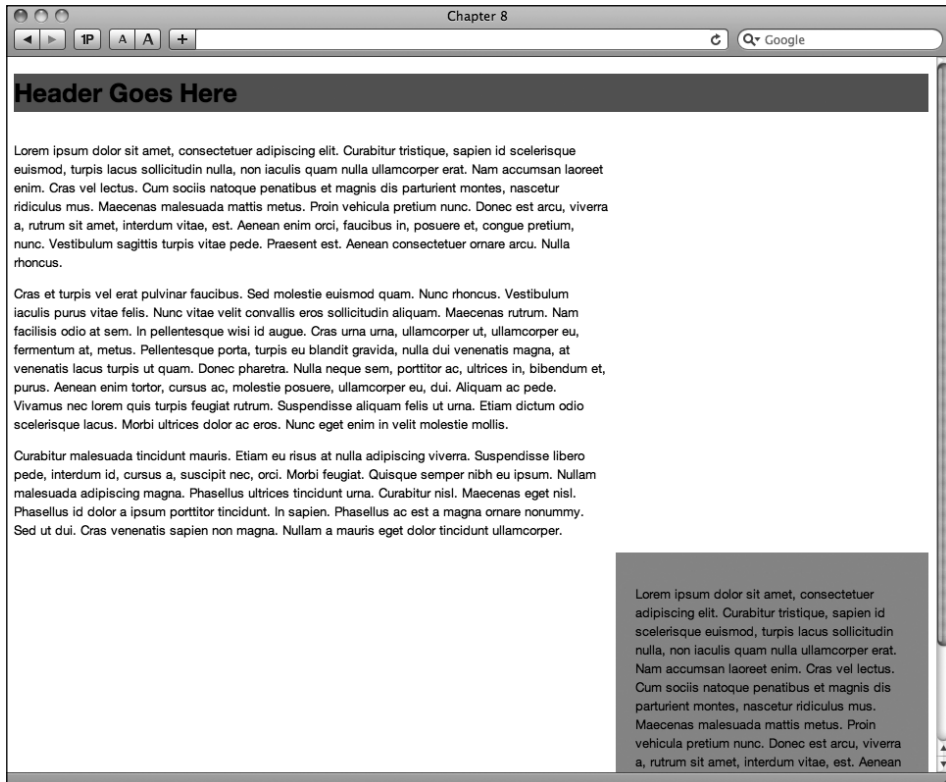


Рис. 8.12. Добавление пиксельного значения отступа к ширине, заданной в процентах, нарушает целостность верстки

Что делать в подобных случаях?

- Используйте процентное значение для задания полей и вычтите это значение из ширины колонки (как мы делали при задании 5%-го средника ранее);
- задавайте отступы от элементов только внутри боковой панели;
- добавьте еще один `<div>`, чтобы задавать любые отступы;
- используйте новое невероятно удобное свойство `box-sizing`, позволяющее изменить способ, которым браузер вычисляет размеры элемента (например, учет отступа внутри элемента).

ДОБАВЛЕНИЕ ОТСТУПА ВНУТРИ ЭЛЕМЕНТА

Один из способов решения проблемы — добавление отступа на уровне блока только тем элементам, которые вы предполагаете использовать в боковой панели. Если вы знаете, что там будут текст и списки, задайте левый и правый отступы для соответствующих элементов:

```
#sidebar p, #sidebar ul, #sidebar ol, #sidebar dl {
    padding-left: 20px;
    padding-right: 20px;
}
```

Такой отступ не повлияет на ширину колонки, однако не забывайте, что в боковую панель в любой момент могут быть добавлены непредусмотренные элементы. Вы как дизайнер должны убедиться, что такое объявление будет создавать корректные отступы в любой ситуации.

Этот метод хорошо подходит для простого предопределенного контента. Однако существует более пуленепробиваемый способ, требующий, правда, более длинного кода разметки.

ДОПОЛНИТЕЛЬНЫЙ КОНТЕЙНЕР <DIV>

Учитывая, что все мы используем в работе принцип «с глаз долой — из сердца вон», дополнительный контейнер `<div>` гарантирует, что весь контент боковой колонки будет иметь корректный отступ, что не повлияет на общую ширину колонки.

Если в разметку включить еще один `<div>`, располагающийся внутри боковой колонки,

```
<div id="sidebar">
  <div>
    ... здесь будет боковая колонка ...
  </div>
</div>
```

то мы сможем задать `padding` для внутреннего `<div>`, сохранив ширину внешнего контейнера 30%:

```
header[role="banner"] {
    background: #666;
}
#content {
    float: left;
    width: 65%;
    margin-right: 5%;
}
#sidebar {
    float: right;
    width: 30%;
    background: #999;
}
#sidebar div {
    padding: 20px;
}
footer[role="contentinfo"] {
    clear: both;
    background: #DDD;
}
```

СОВЕТ

С помощью селектора `#sidebar div` мы можем добавить отступ для любого `<div>`, размещенного под `#sidebar`. Для нескольких уровней вложения `<div>` все они будут иметь отступ 20 пикселей.

Отступ `<div>` не повлияет на ширину родительского контейнера. Теперь мы можем указать любое значение отступа, и в данном случае он не будет зависеть от ширины окна.

Аналогичным образом, если мы хотим задать фиксированный средник, для контента можно добавить внутренний контейнер `<div>` и задать правый отступ нужного размера в пикселах:

```
<div id="content">
    <div>
        ... здесь будет контент ...
    </div>
</div>
```

Используя внутренний `<div>`, справа от колонки можно добавить 40 пикселей отступа, чтобы создать средник фиксированной ширины:

```
header[role="banner"] {
    background: #666;
}
#content {
    float: left;
    width: 70%;
}
#content div {
    padding-right: 40px;
}
#sidebar {
    float: right;
    width: 30%;
    background: #999;
}
#sidebar div {
    padding: 20px;
}
footer[role="contentinfo"] {
    clear: both;
    background: #DDD;
}
```

На рис. 8.13 показаны результаты. При изменении ширины окна отступ боковой панели и 40-пиксельный средник фиксируются, а ширина колонок при этом изменяется.

Недостатком фиксированного пространства между колонками и отступа является добавление в разметку дополнительного незначимого `<div>`. Чтобы принять такое решение, нужно определиться, нужны ли вам фиксированные отступы в макете. Если нужны, тогда смиритесь с избыточной разметкой. Если ваша цель — простота, тогда экспериментируйте с процентными значениями.

ИСПОЛЬЗУЕМ СВОЙСТВО BOX-SIZING ИЗ CSS3

В стандарте CSS3 есть новое и очень удобное свойство `box-sizing`, которое позволяет менять способ расчета ширины и отступов. Как уже говорилось ранее, браузеры рассчитывают размеры элемента, принимая за основу заданную ширину и (или) высоту, а затем добавляя отступы или границы. Поэтому задание фиксированных средников с шириной, указанной в процентах, представляет большую проблему.

Вы не можете использовать `width: 100%` и `padding: 20px`; для одного и того же элемента, так как его ширина окажется $100\% + 20\text{px}$. Такого допускать нельзя! Вы можете вызвать сбой в работе браузера, провоцируя возникновение горизонтальных полос прокрутки для компенсации избыточной ширины.

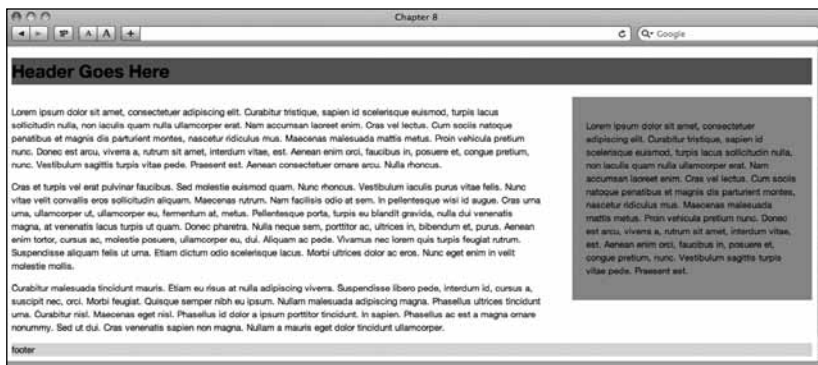
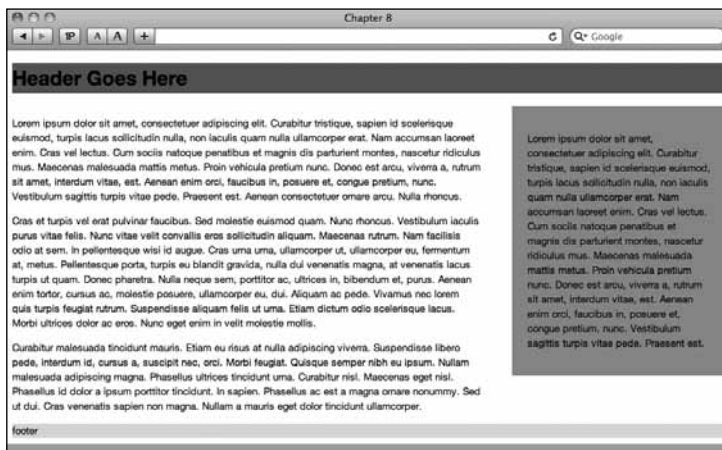
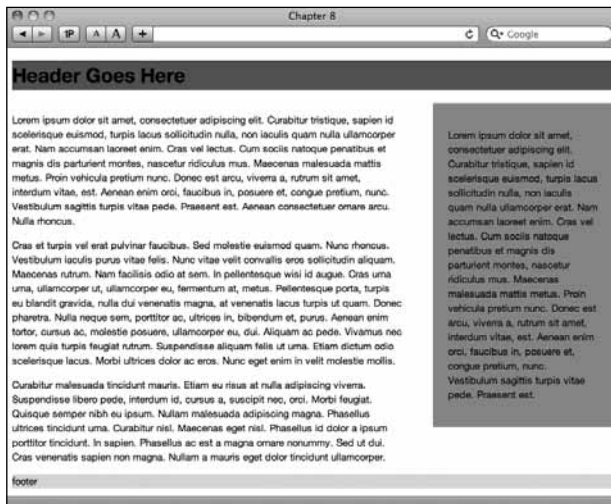


Рис. 8.13. Мы задали отступ, независимый от ширины колонки

Свойство `box-sizing` позволяет довольно элегантно решить проблему, используя значение `border-box`. Способ расчета изменится — ширина будет рассчитываться с учетом отступа и (или) границ. Вы сможете задавать ширину в процентах, а отступы и границы — в пикселах, а в разметку не придется помещать дополнительные контейнеры.

Давайте разберемся на нашем примере, как работает `box-sizing`, добавив колонкам макета отступ 20 пикселей. Обратите внимание, что мы снова используем префиксы разработчика для браузеров WebKit, Mozilla и др., которые пока не поддерживают это свойство.

```
header[role="banner"] {
    background: #666;
}
#content {
    float: left;
    width: 70%;
    padding: 20px;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}
#sidebar {
    float: right;
    width: 30%;
    padding: 20px;
    background: #999;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}
footer[role="contentinfo"] {
    clear: both;
    background: #DDD;
}
```

ПРИМЕЧАНИЕ

В оставшейся части этой главы я буду использовать метод дополнительного контейнера `<div>`, чтобы продемонстрировать возможности резиновой верстки.

Используя значение `border-box`, мы можем добавлять отступ (и границы), гарантируя, что ширина, которую мы задали в процентах, останется неизменной. При этом не нужно помещать контейнеры `<div>` или задавать отступы внутри элементов, как в способах, описанных ранее в этой главе.

Область действия `box-sizing` (с префиксами разработчика `-webkit-` и `-moz-`) довольно обширна: IE8+, Firefox 2+, Chrome 4+, Safari 3.1+ и Opera 9.5+ поддерживают это свойство. Посмотрите на статистику посещений вашего сайта, метод `box-sizing` может быть вполне жизнеспособным (и, конечно, самым простым и элегантным).

ЗАДАНИЕ МИНИМАЛЬНОЙ И МАКСИМАЛЬНОЙ ШИРИНЫ

Одна из основных проблем дизайнеров при работе с резиновыми макетами связана с длиной строки. Предоставляя пользователю возможность менять ширину страницы, мы сталкиваемся с проблемой, когда пользователь настолько уменьшает текст в колонке, что тот перестает читаться, или настолько увеличивает, что меняется внешний вид страницы.

В этой ситуации нам помогут свойства `max-width` и `min-width`. Задавая максимальную или минимальную ширину резинового макета, мы запрещаем растягивание или сжатие колонок в конкретном месте. К сожалению, Internet Explorer 6 и более ранние версии не поддерживают эту функцию. Однако свойства `max-width` и `min-width` можно эффективно использовать в других браузерах.

Вернемся к нашему примеру. Если мы зададим `max-width` для контейнера `<div id="wrap">`, в котором находится весь макет, то гарантируем, что колонки не станут слишком широкими:

```
#wrap {
    max-width: 1200px;
}
```

На рис. 8.14 показан наш пример с использованием правила `max-width`. Макет остается гибким, однако расширение колонки не превысит 1200 пикселей, даже если само окно окажется больше этого размера.

Можно применить `max-width` к отдельным колонкам. Например, чтобы колонка контента прекращала расширяться по достижении размера 600 пикселей, но на весь макет это не влияло (рис. 8.15), используйте следующий код:

```
#content {
    max-width: 600px;
}
```

Мы можем задать `min-width` для макета (или отдельных колонок), чтобы избежать сжатия дизайна:

```
#wrap {
    max-width: 1200px;
    min-width: 600px;
}
```

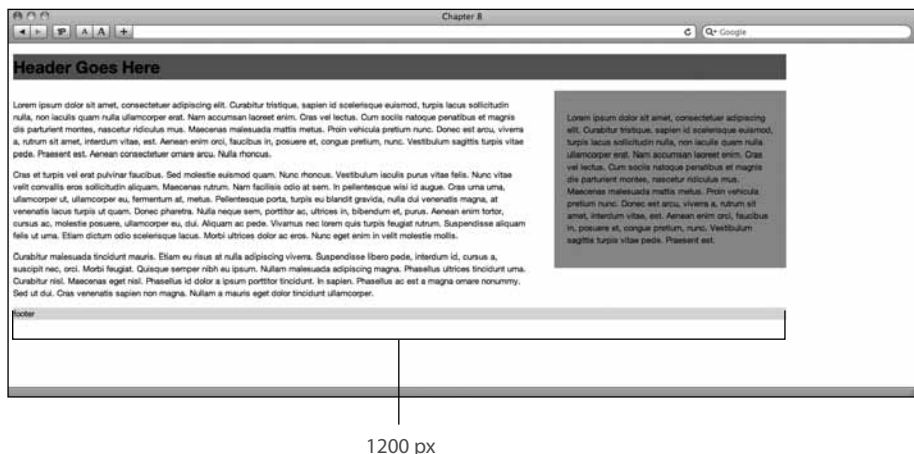


Рис. 8.14. Если попытаться расширить окно до размера более 1200 пикселей, макет прекратит расширение

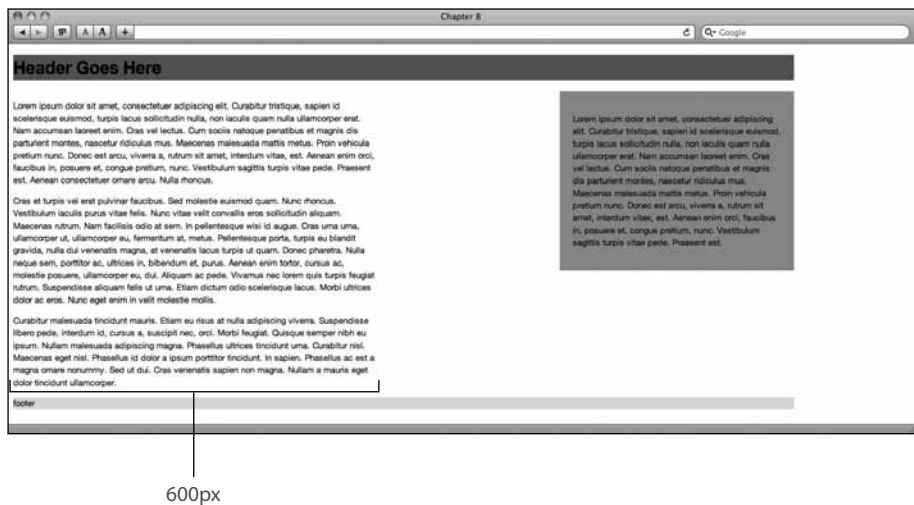


Рис. 8.15. Если размер колонки контента становится больше 600 пикселей, она прекращает расширяться

На рис. 8.16 показан макет с использованием правила `min-width`. Оно запрещает дальнейшее уменьшение размера макета по достижении ширины 600 пикселей. Когда окно браузера становится меньше указанного размера, выводятся горизонтальные полосы прокрутки.

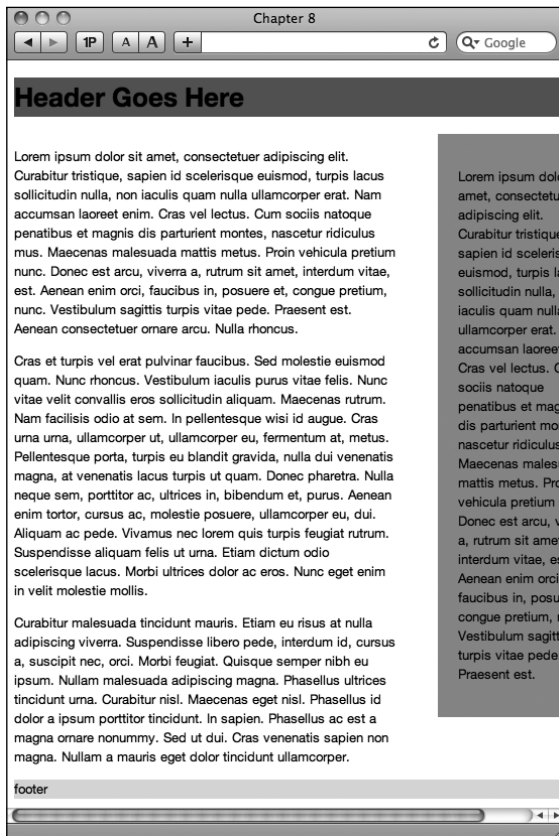


Рис. 8.16. Если ширина страницы меньше 600 пикселей, появляются горизонтальные полосы прокрутки

Это необходимо, когда в колонке размещен объект с фиксированной шириной (например, изображение), который был бы «вытолкнут», если бы ширина колонки стала меньше размеров объекта.

Если мы поместим изображение с шириной 500 пикселей в колонку контента, а ширина окна окажется такой, что ширина колонки контента будет меньше 500 пикселей, колонка контента перекроет боковую колонку (рис. 8.17). А в некоторых браузерах может пострадать даже целостность всего макета.

СОВЕТ

CSS может масштабировать изображения внутри резиновых макетов (почитайте статью и книгу Итана Маркотта (*Ethan Marcotte, Responsive Web Design*)). Этот вопрос мы рассмотрим в главе 9 «Сводим воедино».

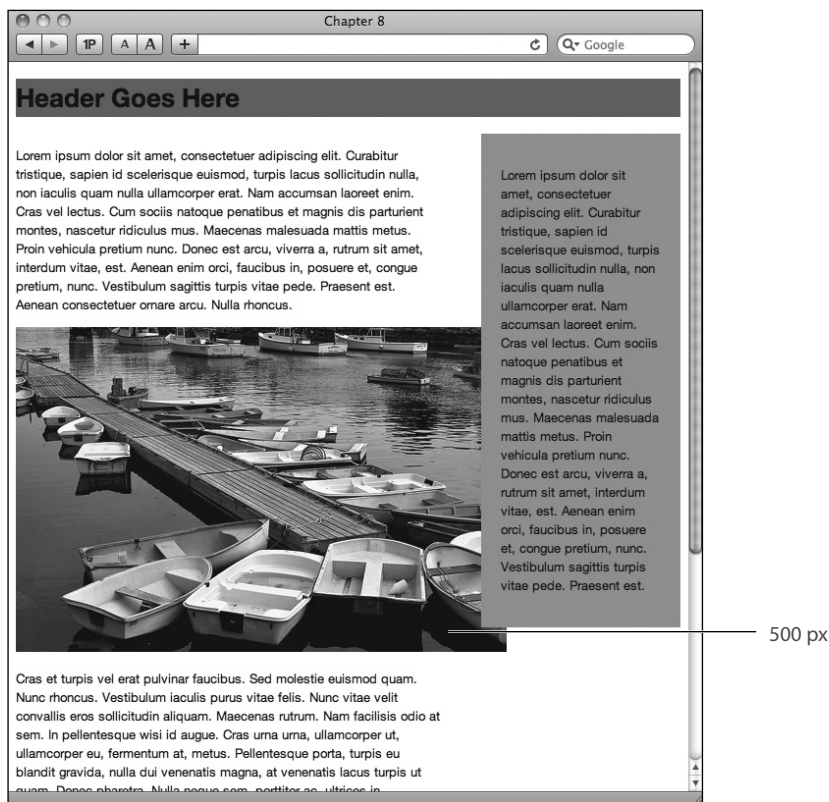


Рис. 8.17. Если ширина изображения 500 пикселей, а колонка контента уже, то колонки перекрываются

Поэкспериментируйте с `min-width` контейнера `<div>` (так же как с `max-width`): вы сможете предотвратить перекрытие или нарушение целостности макета (почти во всех браузерах, кроме IE6). Эти правила не действуют в IE6, но дают преимущества пользователям других браузеров. Кроме того, они необходимы для управления резиновыми макетами.

```
#wrap {
    max-width: 1200px;
    min-width: 750px;
}
```

Скользящие псевдоколонки

Еще раз посмотрите на двухколоночный макет, с которым мы работаем в этой главе. Обратите внимание на фоновую заливку боковой колонки. При изменении количества контента в колонке видно, что ее длина подстраивается под

контент и не совпадает длиной левой колонки. Одинаковые по высоте колонки со сплошной фоновой заливкой и (или) границами — достаточно популярный стиль оформления. Но при использовании резиновых макетов невозможно создать две абсолютно одинаковые по высоте колонки.

Одно из возможных решений проблемы я привел в статье «Faux Columns» («Псевдоколонки») (www.alistapart.com/articles/fauxcolumns/, рис. 8.18). В статье описана методика создания фейковых колонок одной высоты в макетах на базе CSS за счет повторяющегося фонового изображения. Мы уже использовали этот подход в главе 6 «Нет картинок? Нет CSS? Нет проблем!».

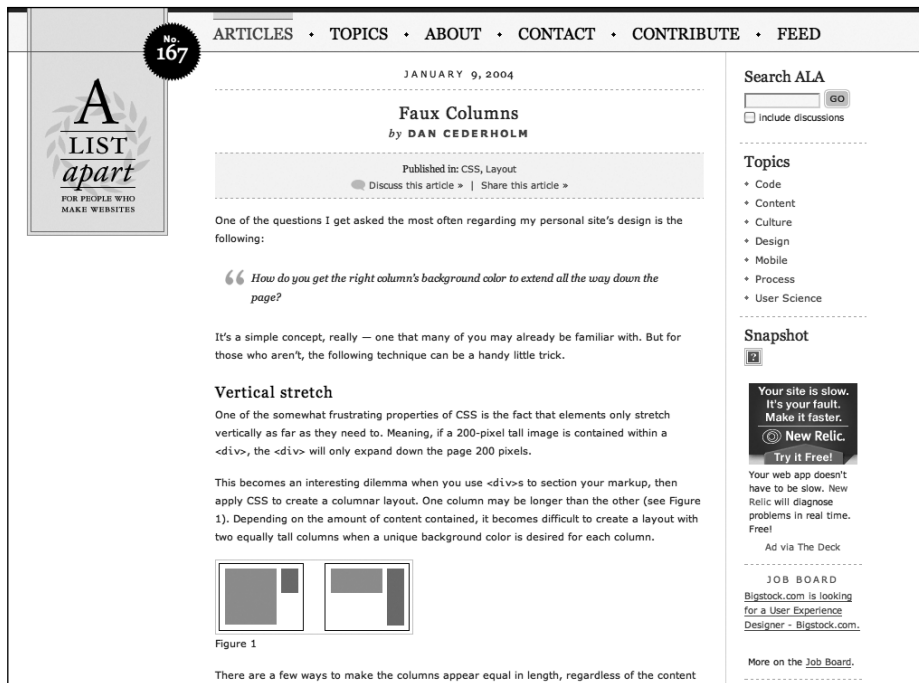


Рис. 8.18. Статья «Faux Columns» была написана для сайта A List Apart, в ней отражена методика создания одинаковых по высоте псевдоколонок с помощью CSS, использующая мозаичное заполнение фона

Метод прост: повторяющееся фоновое изображение размещается под колонками и создает иллюзию того, что колонки имеют одинаковую высоту. В результате фоновая заливка и границы всегда будут находиться позади макета и заканчиваться в одном и том же месте. Этот способ работает только в фиксированных макетах, где ширина колонок predetermined и ее можно использовать в расчете раскладки изображений.

Дуглас Боуман (www.stopdesign.com/log/2004/09/03/liquid-bleach.html) и Эрик Мейер (www.meyerweb.com/eric/thoughts/2004/09/03/sliding-faux-columns/) развили идею, разработав на ее базе принцип скользящих псевдоколонок. Повторяющееся изображение может «скользить» вместе с резиновыми колонками, создавая эффект одинаковой высоты колонок и сохраняя гибкость. Посмотрим, как этот принцип работает.

СОЗДАНИЕ ФОНОВОГО ИЗОБРАЖЕНИЯ

При создании фонового изображения мы сделаем его настолько широким, чтобы оно перекрывало самый большой экран (в расчете на пользователей, которые будут открывать ваш сайт на мониторах с максимальной диагональю). Будем использовать округленное значение ширины, например 2000 пикселей, чтобы упростить расчеты, необходимые для позиционирования изображения.

Нам нужны серая заливка и рамка, заполняющие все вертикальное пространство до нижнего колонтитула. Если ширина боковой колонки 30%, то 30% от 2000 пикселей дадут 600 пикселей.

Добавим серое поле шириной 600 пикселей к 2000-пиксельному белому полю высотой 100 пикселей (рис. 8.19). Высота не имеет значения, потому что изображение все равно будет повторяться по вертикали.

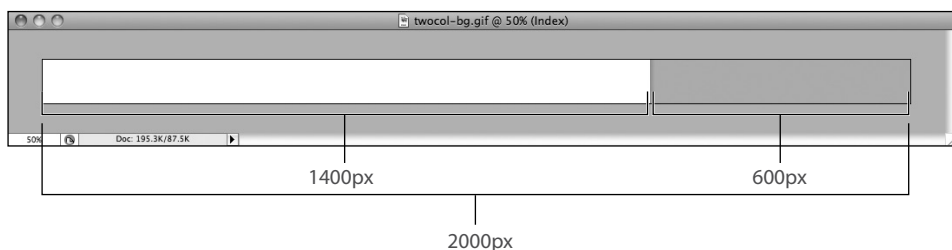


Рис. 8.19. Будем использовать GIF для создания псевдоколонок

К серому полю мы добавили 1-пиксельную границу с небольшой тенью по левому краю (рис. 8.20).



Рис. 8.20. Мы создали границу и тень вдоль левой стороны колонки

ПОЗИЦИОНИРОВАНИЕ ИЗОБРАЖЕНИЯ

Поместим фоновое изображение в контейнер `<div id="wrap">`, в котором находится весь наш макет. Зададим повторение изображения по вертикали. Отступ на 70% слева даст нам гарантию, что серый фон всегда будет находиться под боковой панелью. В зависимости от ширины окна мы будем видеть большую или меньшую часть этого изображения.

```
#wrap {
    max-width: 1200px;
    background: url(img/twocol-bg.gif) repeat-y 70% 0;
}
```

На рис. 8.21 показаны результаты. Боковая панель занимает все пространство до нижнего колонтитула и меняет свои размеры в соответствии с шириной окна.

Чтобы получше разобраться с тем, как это работает, давайте посмотрим, как соотносятся видимая область окна и наше фоновое изображение шириной 2000 пикселей (рис. 8.22). Картинка открывается или скрывается, по мере того как расширяется или сжимается окно. Очень важно, что ширина изображения составляет 2000 пикселей, так ваш дизайн сохранится даже на огромных экранах.

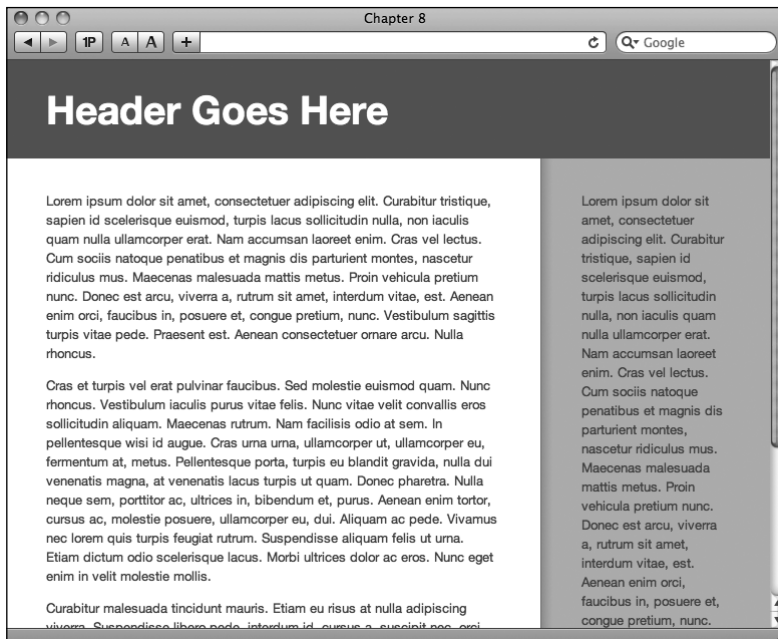


Рис. 8.21. Скользящие псевдоколонки в действии (см. с. 248 продолжение)

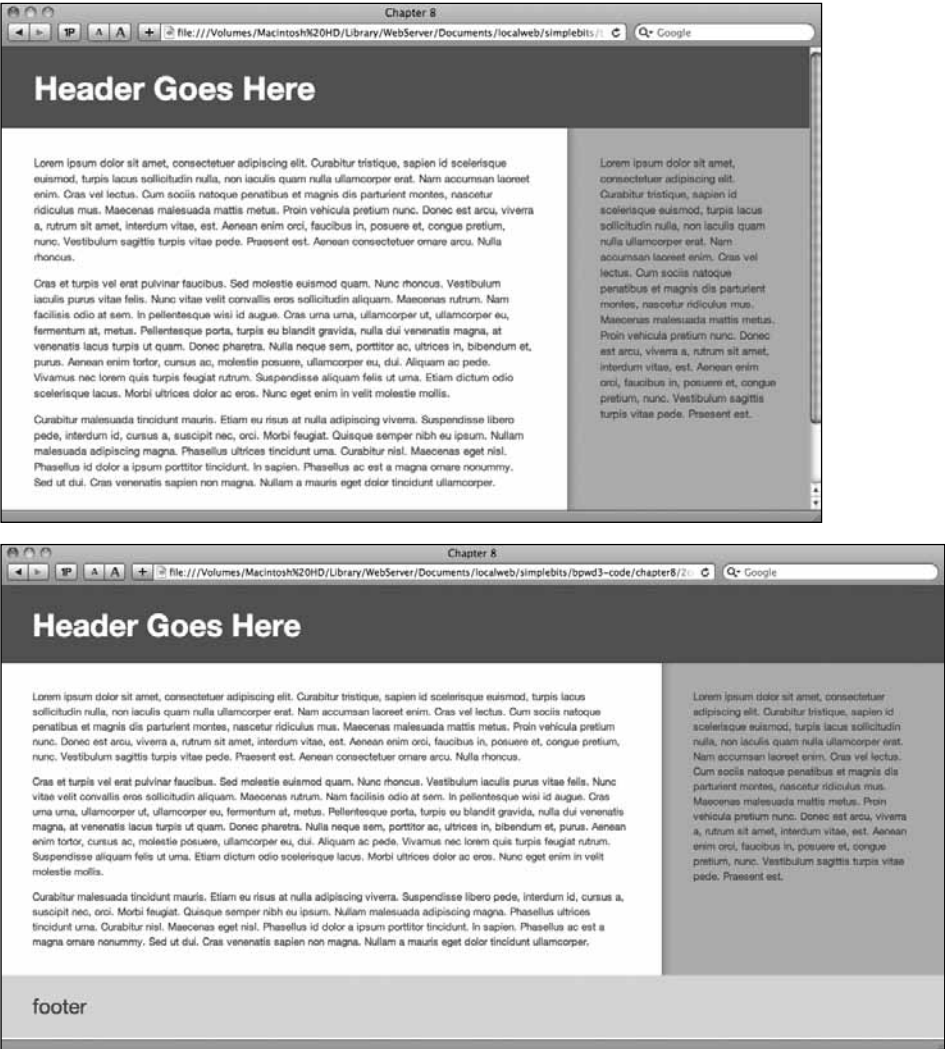


Рис. 8.21 (окончание)

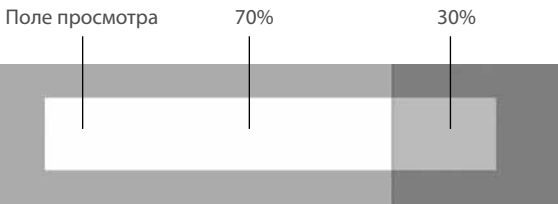


Рис. 8.22. Фрагмент 2000-пиксельного изображения в видимой области окна

ТРЕХКОЛОНОЧНАЯ ВЕРСТКА

До сих пор речь шла только о двухколоночной верстке. На ее примере проще разобраться с основными концепциями. Но давайте двигаться дальше — теперь мы поговорим о трехколоночной. К ней применимо большинство изученных нами ранее принципов, однако способ первоначального создания колонок более сложен. Давайте начнем со структуры разметки.

Структура разметки

При создании трехколоночных макетов хочется воспользоваться преимуществами сортировки исходного кода — сначала в разметке определяется основной контент (для тех пользователей, которые предпочитают текстовые браузеры или приложения для чтения с экрана), а затем — все остальное. Решить эту задачу, а также гарантировать, что ни одна из колонок не будет пересекаться с нижним колонтитулом страницы, можно методом плавающих элементов, так же как и в предыдущем случае с двумя колонками.

Чтобы основной контент шел в разметке первым, нам понадобится дополнительный контейнер `<div>`, который будет объединять только колонку контента и левую боковую панель. Причина такого выбора станет ясна уже через минуту.

```
<div id="wrap">
  <header role="banner">
    <h1>Это заголовок</h1>
  </header>
<div id="main-body">
  <div id="content">
    ... здесь будет контент ...
  </div>
  <div id="sidebar">
    ... это левая колонка ...
  </div>
</div> <!-- end #main-body -->
  <div id="sidebar-2">
    ... это правая колонка ...
  </div>
  <footer role="contentinfo">
    ... это нижний колонтитул ...
  </footer>
</div> <!-- конец #wrap -->
```

Разобьем создание колонок на два этапа. Прежде всего создадим две колонки, сделав `#main-body` и `#sidebar-2` плавающими, и разместим их друг напротив друга. Вспомните, мы уже делали так в макете с двумя колонками, только использовали `#content` и `#sidebar`.

Затем выровняем `#content` по правому краю контейнера, а `#sidebar` — по левому. Эти два контейнера `<div>` формируют две колонки, которые мы разместим слева от `#sidebar-2`. Теперь у нас есть три колонки, и первой в разметке идет колонка с основным контентом.

На рис. 8.23 показано расположение контейнеров `<div>`. Стрелкой обозначено направление выравнивания. Обратите внимание на то, что `#main-body` выравнивается по левому краю, а `#content` и `#sidebar` по его наружным сторонам.

Такая структура и порядок размещения элементов дают нам возможность контролировать разметку, позволяя очищать колонки в полноразмерном нижнем колонтитуле.

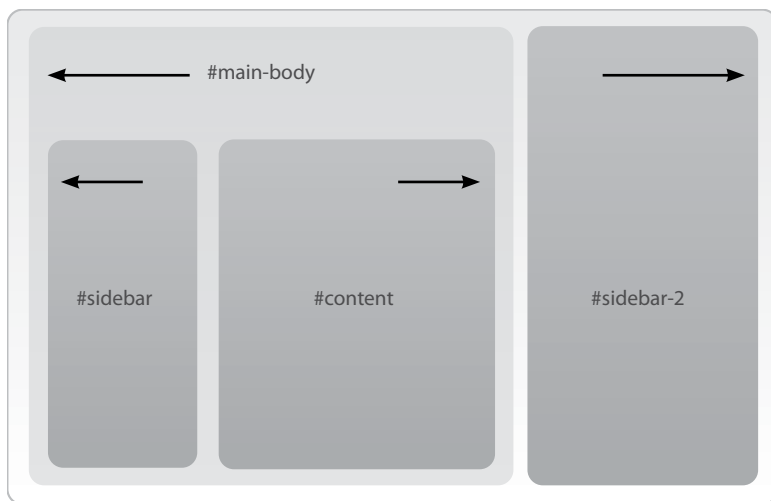


Рис. 8.23. Каждый `<div>` расположен так, чтобы сохранить оптимальную структуру контента

Стилевое оформление

После работы с разметкой нужно описать плавающие элементы и ширину конкретных колонок в процентах. Сначала зададим ширину 70% для `#main-body` (левая боковая колонка и колонка контента) и выровняем этот контейнер по левому краю, потом зададим ширину 30% для `#sidebar-2` и выровняем его по правому. Это позволит нам создать две колонки:

```
header[role="banner"] {  
    background: #666;  
}
```

```
#main-body {
    float: left;
    width: 70%;
}
#sidebar {
    background: #CCC;
}
#sidebar-2 {
    float: right;
    width: 30%;
    background: #999;
}
footer[role="contentinfo"] {
    clear: both;
    background: #DDD;
}
```

Я также использовал фоновую заливку, чтобы показать границы элементов. На рис. 8.24 видна колонка #sidebar-2 на темно-сером фоне, а колонка #sidebar (светло-серый фон) расположена под колонкой основного контента.

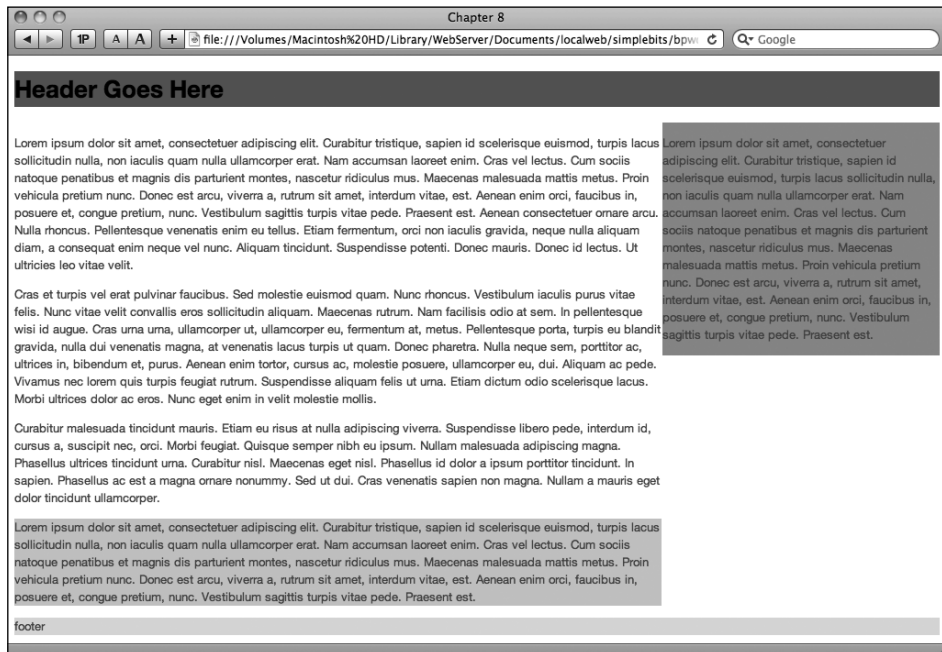


Рис. 8.24. Правая боковая колонка размещается напротив колонки контента и левой боковой панели

Мы создали две из трех колонок, теперь, чтобы завершить разработку базовой структуры, нужно прижать `#content` к правому краю контейнера, а `#sidebar` — к левому. Мы также зададим для них процентные значения ширины, которые будут актуальны в пределах 70% общей ширины страницы (в `#main-body`).

```
header[role="banner"] {
    background: #666;
}
#main-body {
    float: left;
    width: 70%;
}
#content {
    float: right;
    width: 60%;
}
#sidebar {
    float: left;
    width: 40%;
    background: #CCC;
}
#sidebar-2 {
    float: right;
    width: 30%;
    background: #999;
}
footer[role="contentinfo"] {
    clear: both;
    background: #DDD;
}
```

На рис. 8.25 показаны результаты. Теперь у нас есть три колонки.

Средники и отступы

Мы сталкиваемся с такими же проблемами задания средника и отступа, о которых говорили ранее. Совет, который я давал при разработке двухколоночного макета, применим и к макету из трех колонок. Вы можете либо перераспределить проценты между шириной колонок и размерами полей, либо добавить по контейнеру `<div>` для каждой колонки и задать отступы, независимые от ширины экрана. Выбирайте подходящий вариант, учитывая требования дизайна.

Скользящие псевдоколонки в трехколоночной верстке

Создание псевдоколонок одной высоты (о скользких псевдоколонках мы уже говорили в этой главе) — более сложный процесс, если речь идет о трех эла-

стичных колонках. Для этого потребуется дополнительный контейнер `<div>`, в котором будет находиться весь макет и два фоновых изображения.

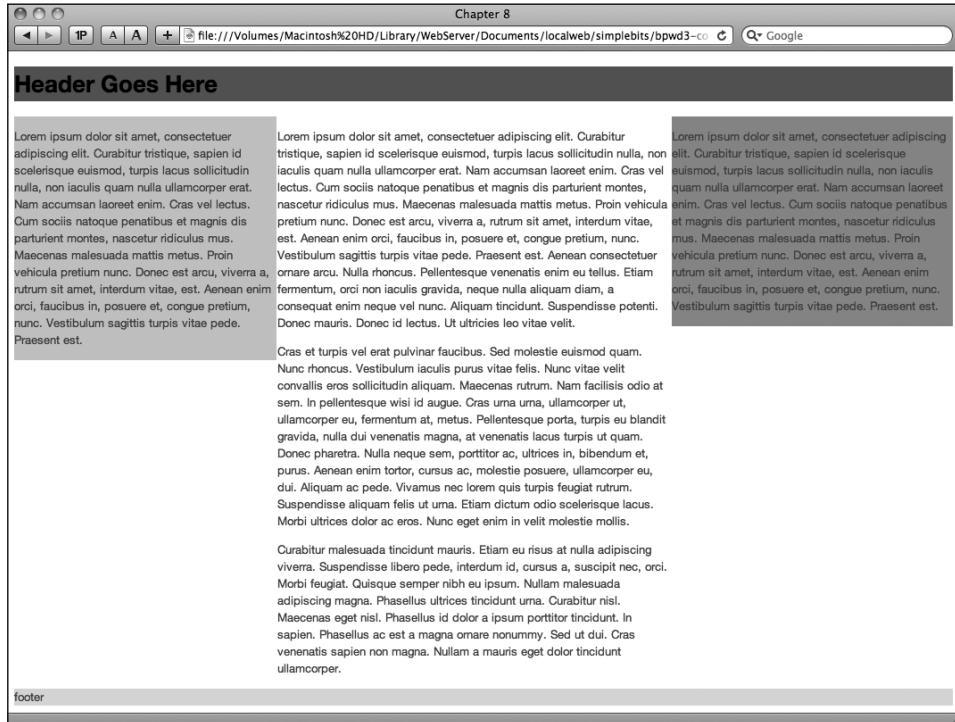


Рис. 8.25. Мы создали три колонки

Давайте добавим дополнительный контейнер `<div>` в уже существующий `<div id="wrap">`. Это позволит нам использовать два изображения для создания фона для двух внешних резиновых колонок:

```
<div id="wrap">
  <div id="wrap-inner">
    ... остальная часть макета будет здесь ...
  </div> <!-- конец #wrap-inner -->
</div> <!-- конец #wrap -->
```

Затем создадим два фоновых изображения, взяв за основу те же принципы, которые мы использовали при создании двухколоночного макета. Ширина каждого изображения составит 2000 пикселей, что позволит сохранить оформление на экранах большого формата. Так как ширина правой колонки равна 30%, соответствующий фрагмент фона для этой колонки должен иметь ширину 600 пикселей (30% от 2000). Назовем это изображение `threecol-r.gif` (рис. 8.26).



Рис. 8.26. Это GIF-изображение мы используем как фон для правой боковой колонки

Ширина левой колонки составляет 40% от общих 70%, которые занимают колонка и контент вместе. Несложные расчеты позволяют узнать, что 40% от 70% составляют 28% общей ширины. Таким образом, фоновое изображение для левой боковой панели будет иметь ширину 28% от 2000, т. е. 560 пикселей. Сделаем остальную часть изображения прозрачной, чтобы сквозь нее могло просвечивать другое изображение (рис. 8.27).

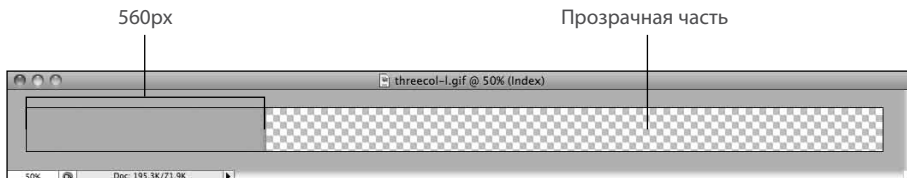


Рис. 8.27. Если мы наложим фоновые изображения друг на друга, прозрачный участок не будет накрывать изображение, расположенное под ним

После создания изображений мы готовы разместить их за двумя `<div>`, которые определили в разметке:

```
#wrap {
    max-width: 1200px;
    background: url(img/threecol-r.gif) repeat-y 70% 0;
}
#wrap-inner {
    background: url(img/threecol-l.gif) repeat-y 28% 0;
}
```

Обратите внимание, что первое фоновое изображение сдвинуто на расстояние 70% от левого края (так же, как было в двухколоночном макете) в основном контейнере `#wrap`. Частично прозрачное изображение мы разместили поверх `#wrap-inner` на расстоянии 28% от левого края (ширина левой колонки).

После вставки изображений мы получили гибкий макет, состоящий из трех цветных фейковых колонок, имеющих одинаковую высоту. Каждая колонка меняет свои размеры в зависимости от размера окна браузера (рис. 8.28).

ПРИМЕЧАНИЕ

На рис. 8.28 я использовал метод дополнительного контейнера для задания отступов всех трех колонок.

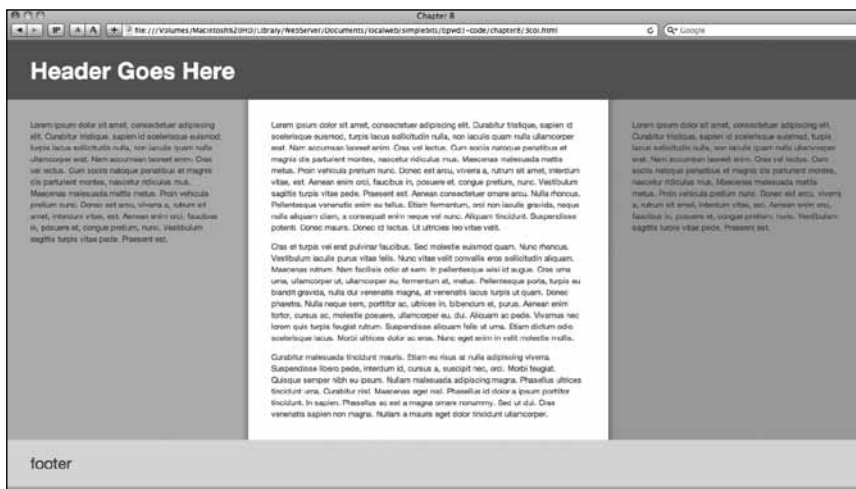
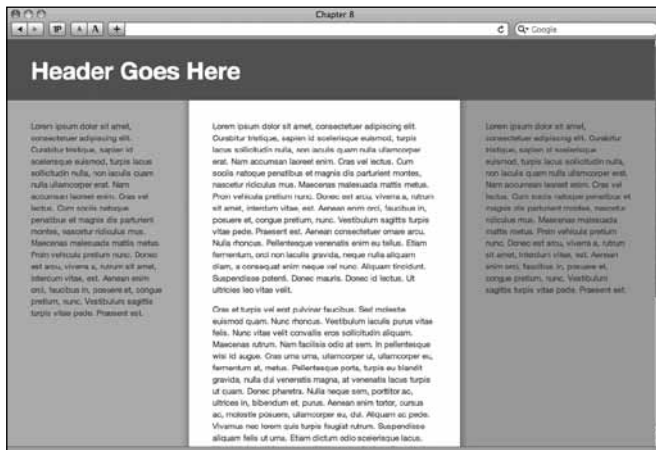


Рис. 8.28. Метод скользящих псевдоколонок в трехколоночном макете

ПРЕИМУЩЕСТВА ПУЛЕНЕПРОБИВАЕМОГО ПОДХОДА

Давайте вспомним, что делает гибкие макеты на базе CSS пуленепробиваемыми (в отличие от макетов на базе таблиц).

Во-первых, такая верстка требует меньше кода. Вместо задания границ, фона и отступов в разметке с помощью вложенных таблиц можно создать многоколоночный макет благодаря нескольким простым контейнерам `<div>`, используя CSS для позиционирования и назначения стиля каждой колонке. Даже добавление избыточных (хотя и необходимых) контейнеров, добавленных для управления размерами средников и (или) скользящих псевдоколонок (в трехколоночных макетах), не увеличивает общий объем кода до значений сопоставимых с кодом при использовании вложенных таблиц.

Во-вторых, такой макет проще обслуживать. Так как элементы оформления находятся в таблице стилей, разметку проще читать. А чтобы внести изменения во внешний вид макета, достаточно будет обновить нескольких правил в таблице стилей. С помощью свойства `float` (CSS) вы можете разместить контент в оптимальном порядке, наиболее важный контент окажется в разметке первым. Это дает преимущество пользователям, предпочитающим приложения для чтения с экрана, текстовые браузеры и иные устройства, не поддерживающие CSS.

И наконец, с помощью резиновой верстки вы предоставляете пользователям дополнительный контроль над страницей. Они могут увеличивать или уменьшать окно при просмотре страницы на небольшом экране. Этот дополнительный уровень контроля — еще один способ увеличить гибкость дизайна и адаптировать его к большинству ситуаций.

ВЕРСТКА НА ОСНОВЕ EM

Верстка на основе `em` обеспечивает другой уровень гибкости. В своей статье для A List Apart (www.alistapart.com/articles/elastic) Патрик Гриффитс пишет, что верстка на основе `em` задает ширину колонок в единицах `em` (а не в пикселах или процентах). Ширина макета и колонок определяется базовым размером шрифта, поэтому изменение размера шрифта ведет к изменению размеров всего оформления.

ПРИМЕЧАНИЕ

Верстка на основе `em` стала менее популярной из-за функции масштабирования, которая сейчас поддерживается большинством браузеров.

Несмотря на то что резиновая верстка имеет фиксированную ширину, при проектировании она требует гибкости мышления. Разметка может меняться в зависимости от размера текста на странице.

Мы уже рассмотрели несколько примеров и использовали `em` для того, чтобы посмотреть, как макет масштабируется при изменении размера шрифта. Давайте проверим эту концепцию на макете из двух колонок и посмотрим, что получится.

Эластичный пример

Давайте возьмем экспериментальную социальную сеть, созданную для людей, которые носят парики, — ToupeePal (<http://toupeepal.com>). На сайте используется двухколоночная структура и `em` в качестве единицы измерения. По мере увеличения размера шрифта оформление макета пропорционально изменяется (рис. 8.29).

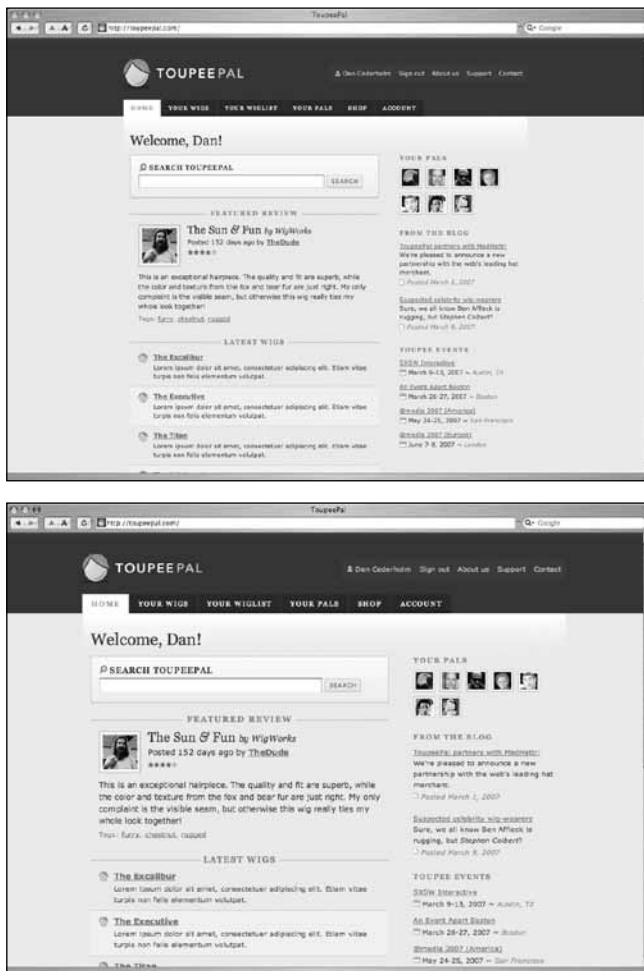


Рис. 8.29. Оформление сайта ToupeePal привязано к `em` и подстраивается под текущий размер шрифта (см. с. 258, продолжение)

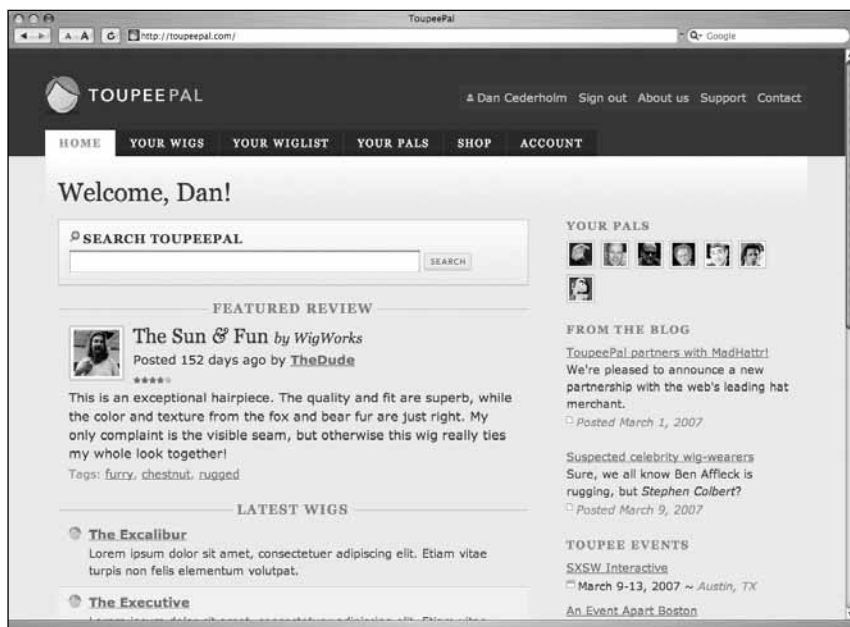


Рис. 8.29 (окончание)

COBET

Обратите внимание на статью «Elastic Lawn» («Эластичный газон») с сайта CSS Zen Garden (www.csszengarden.com/?cssfile=/063/063.css). Это великолепный пример использования `em` в сетке макета, который безропотно следует за всеми изменениями размера шрифта.

Масштабирование оформления обязывает дизайнера задумываться не только о том, что произойдет с сеткой, но и о контенте. Используя пуленепробиваемый подход, вы без труда справитесь и с `em`-дизайном. Речь идет о контроле над изменением размеров страницы с точностью до пикселя.

Давайте начнем с простого двухколоночного макета (рис. 8.30).

РАЗМЕТКА

HTML-структура макета на базе `em` вам хорошо знакома. Мы опять используем плавающие элементы для позиционирования контейнеров `<div>`, содержащих

колонки контента и боковые колонки, обеспечивая оптимальный порядок разметки, включая заголовок и нижний колонтитул страницы.

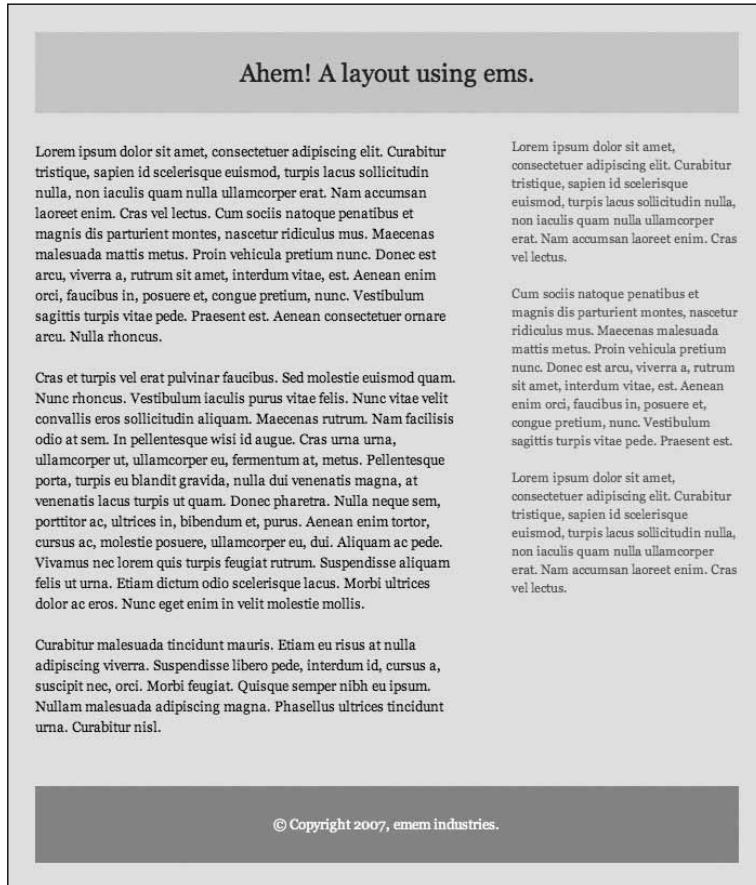


Рис. 8.30. Пример простого двухколоночного макета, который мы разработаем с помощью em

```
<div id="wrap">
<header role="banner">
  <h1>Ahem! A layout using ems.</h1>
</header>
<div id="content">
  <p>Lorem ipsum dolor ... </p>
</div>
<div id="sidebar">
  <p>Lorem ipsum dolor ... </p>
```





```

</div>
<footer role="contentinfo">
  <p>&copy; Copyright 2007, emem industries.</p>
</footer>
</div> <!-- end #wrap -->

```

Основной контейнер `#wrap` содержит все элементы и задает ширину макета. В нем используется по одному элементу (`<div>`, `<header>` или `<footer>`) на каждый компонент страницы: заголовок, контент, боковую панель и нижний колонтитул страницы.

В отличие от резиновых макетов, которые мы разрабатывали ранее, макет на базе `em` не требует дополнительного контейнера `<div>` для создания фиксированного средника. При использовании `em` мы имеем дело с конечной шириной, поэтому можно распределить общую ширину макета между колонками, полями или отступами, чтобы задать средник. Да и сами расчеты становятся проще.

На рис. 8.31 вы видите, что после назначения ширины основного контейнера `<div> 50em` можно указать точное значение, которое будет учитываться в общей ширине двух колонок и просвета между ними ($30em + 2em + 18em = 50em$).

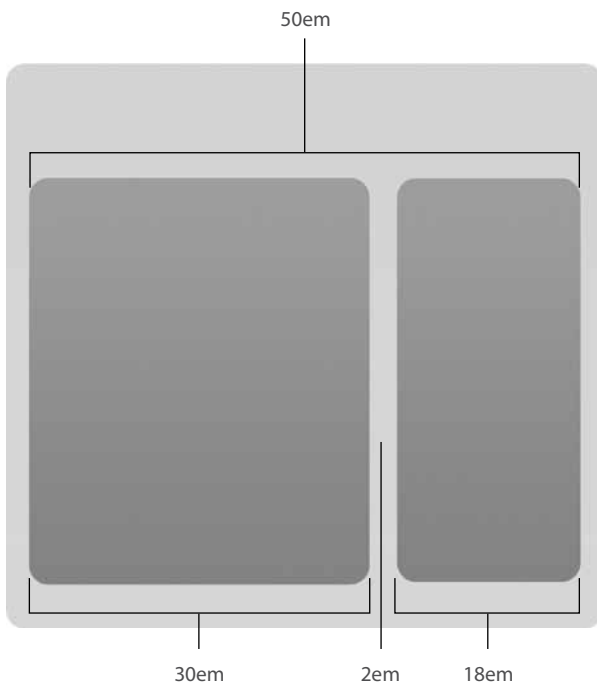


Рис. 8.31. Блок с шириной 50em разбит на две колонки со средником

Создание макета с несколькими колонками на основе em похоже на задание ширины в пикселах: мы просто используем другую единицу измерения, зависящую от текущего размера шрифта. Например, 50em может соответствовать 700 пикселям при среднем размере шрифта или 900 пикселям при увеличении кегля шрифта. Небольшое изменение способа отображения шрифта конкретным браузером может внести коррективы в ширину, заданную в единице измерения em. Нужно понимать, что в данном случае мы не получим точного соответствия пиксельным значениям, что с точки зрения дизайна веб-сайтов даже хорошо.

CSS

Давайте добавим основные стили к разметке, которая позволяет создать эластичный двухколоночный макет. Наиболее важные фрагменты:

```
header[role="banner"] {
    margin: 0;
    padding: 2em;
    text-align: center;
    background: #BACCD8;
}
header[role="banner"] h1 {
    margin: 0;
    padding: 0;
    font-size: 180%;
    line-height: 1em;
    font-weight: normal;
    color: #333;
}
#wrap {
    width: 50em;
    margin: 0 auto;
    padding: 2em;
    background: #DAE3E9;
}
#content {
    float: left;
    width: 30em;
    padding: 2em 0;
}
#sidebar {
    float: right;
    width: 18em;
    padding: 2em 0;
}
```





```
footer[role="contentinfo"] {  
  clear: both;  
  padding: 1em 2em;  
  text-align: center;  
  color: #FFF;  
  background: #8194A1;  
}
```

Для контейнера `#wrap` мы задали ширину `50em`. Затем внутри него определили стили `header[role="banner"]` и `<h1>`. Контейнер `#content` имеет размер `30em` и выровнен по левому краю, ширина `#sidebar` составляет `18em` (дополнительные `2em` — средник), а сам он выравнивается по правому краю. Нижний колонтитул страницы очищает оба плавающих элемента так же, как это происходило в предыдущих примерах резиновых макетов.

Вы можете указать любую ширину колонок и общую ширину. Добавьте колонкам отступы или поля для создания свободного пространства. С помощью `em` расчеты упрощаются и не требуется дополнительной разметки, как при работе с колонками, когда вы указывали ширину в процентах, а средник имел фиксированную ширину.

Теперь у нас есть простой двухколоночный макет, который мы разработали с помощью `em`. Масштаб макета будет изменяться в соответствии с текущим размером шрифта (рис. 8.32).

СОВЕТ

Свойства `max-width` и `min-width` помогут создать полосы прокрутки, ограничивая эластичность при верстке макетов на базе `em`.

ИДЕАЛЬНАЯ СОГЛАСОВАННОСТЬ

Когда основной макет ориентирован на `em`-размеры, внутренние элементы так же предпочтительно измерять в `em` (отступы вокруг элементов, межстрочные интервалы и т. п.). Тогда пропорционально изменяться будет не только сетка, но и все внутренние компоненты. Это не всегда просто реализовать, особенно когда вы работаете с блоками фиксированной ширины, например с рекламой или изображениями. Но если вы хотите сделать ваше творение как можно более пуленепробиваемым (как примеры в данной книге), тогда выбор между фиксированными, резиновыми и эластичными макетами будет довольно прост.

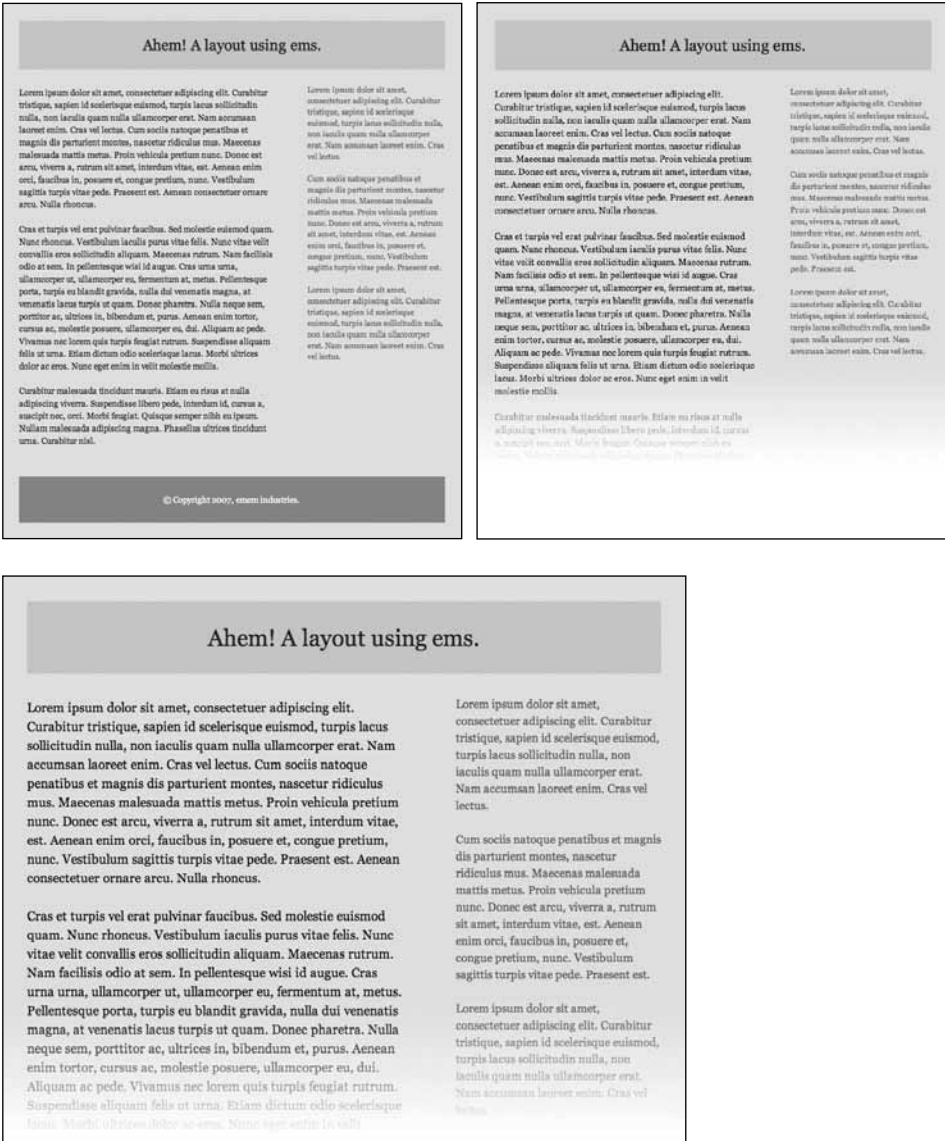


Рис. 8.32. Увеличение размера шрифта в браузере вызовет пропорциональное изменение всего макета, сверстанного на основе em

Полосы прокрутки

Говоря об эластичных макетах, необходимо отметить один важный аспект: так как весь макет будет расширяться по мере изменения размера шрифта, появле-

ние горизонтальных полос прокрутки, когда ширина макета превысит размер области просмотра браузера, может быть нежелательным (рис. 8.33). Этот нюанс нужно учитывать в широких макетах. В более компактном оформлении такой проблемы обычно не возникает.

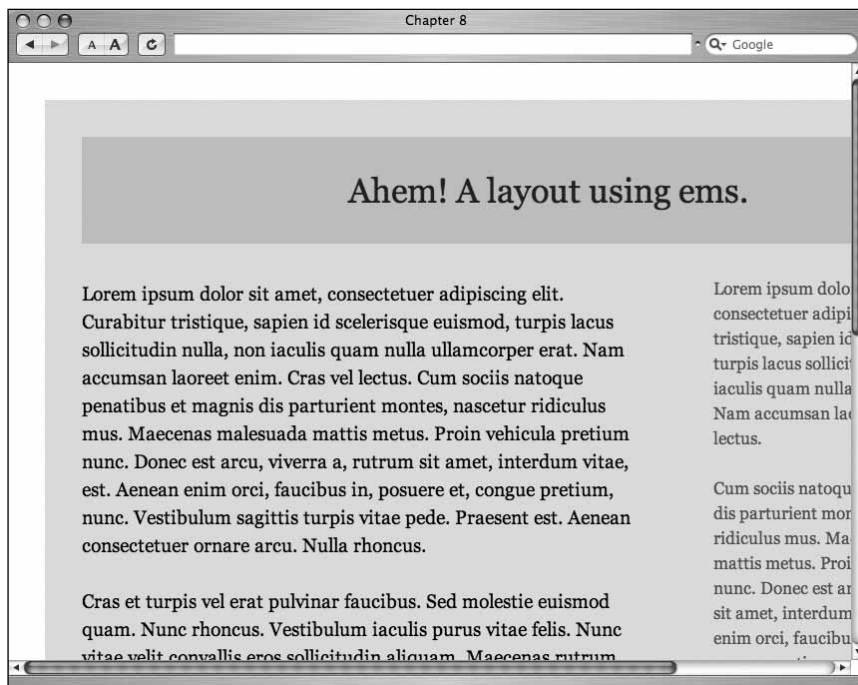


Рис. 8.33. Наш макет на основе em, когда размер шрифта становится таким, что для просмотра всей страницы нужны горизонтальные полосы прокрутки

РЕЗЮМЕ

Фиксированный макет? Резиновый макет? Эластичный макет? Разве какой-либо из них однозначно лучше остальных? При выборе типа верстки важно помнить, что каждая разновидность имеет свои преимущества. Для того чтобы понять принципы работы и узнать о достоинствах и недостатках решений, поэкспериментируйте с каждым из них.

Снова хочу обратить ваше внимание на то, что в этой главе я рассмотрел резиновый и эластичный макеты совсем не потому, что считаю их лучше фиксированного. Наоборот, каждый из них хорош в соответствующей ситуации. Кроме

того, в зависимости от требований дизайна решение о типе макета может принимать не веб-дизайнер.

Если вашему оформлению можно добавить свойства резинового или эластичного макета, то не стоит пренебрегать удобством пользователей. Резиновая верстка является основой отзывчивого веб-дизайна (Responsive Web Design, RWD) — адаптации табличных макетов к конкретным требованиям различных устройств. В следующей главе мы подробно изучим этот принцип.

Вот о чем нужно помнить при разработке гибких макетов:

- для создания многоколоночных макетов, которые можно будет очистить с помощью полноразмерного нижнего колонтитула, используйте свойство `float`;
- при разработке резиновых макетов не забывайте про ширину средников или используйте дополнительные контейнеры `<div>` для задания отступа, независящего от ширины колонки;
- используйте свойство `box-sizing: border-box`; (CSS3), когда задаете поля и границы внутри элемента, ширина которого указана в процентах; это позволяет отказаться от дополнительного контейнера для средника; помните о том, что не все браузеры поддерживают данное свойство;
- в резиновых шаблонах используйте `min-width` и `max-width` для предотвращения чрезмерного увеличения или уменьшения; не обращайтесь к IE6;
- экспериментируйте со скользящими псевдоколонками, чтобы создавать одинаковые по высоте гибкие колонки с границами и фоновыми заливками, не влияющими на нижний колонтитул;
- макеты на основе `em` могут вывести ваш проект на другой уровень гибкости; помните о том, что увеличение шрифта способно заставить часть оформления исчезнуть из области просмотра браузера, что приведет к появлению горизонтальной полосы прокрутки (но только если вы не использовали свойство `max-width`).

После того как мы изучили основы создания гибкого резинового макета на базе CSS, сведем все элементы, рассмотренные в книге, воедино и разработаем полноценный пуленепробиваемый дизайн страницы. Итак, нас ожидает последняя глава.

ГЛАВА 9

СВОДИМ ВОЕДИНО



Используем все пуленепробиваемые концепции в дизайне одной страницы

Настало время опробовать на практике все многочисленные советы и методики пуленепробиваемости на примере дизайна страницы в полном объеме. Пытаясь воссоздать страницу, вы увидите, как все гибкие стратегии работают одновременно.

Для этого примера я создал сайт вымышленной компании — Bulletproof Pretzel Company. Контент не имеет значения (хотя я иногда думаю, что если бы мне не удалось стать веб-дизайнером, я бы мог печь крендельки). Важно то, как пуленепробиваемые концепции будут работать вместе, обеспечивая гибкость дизайна. Этот пример не является идеалом с точки зрения дизайна или пуленепробиваемости. Его цель — освежить знания о пуленепробиваемых концепциях, применив их в реальной жизни для дизайна страницы, состоящей из разных компонентов.

Сначала определим нашу цель, то есть конечный продукт. Затем решим, какая разметка понадобится для разработки резинового двухколоночного макета. После этого мы пройдемся по всем компонентам страницы, вспоминая пуленепробиваемые методики, которые мы изучили.

Цель

На рис. 9.1 показана цель, к которой мы будем стремиться в этой главе. Веб-сайт компании Bulletproof Pretzel Company — прост и информативен. Но еще более важно, что он разработан на базе концепций, о которых мы говорили ранее в этой книге.

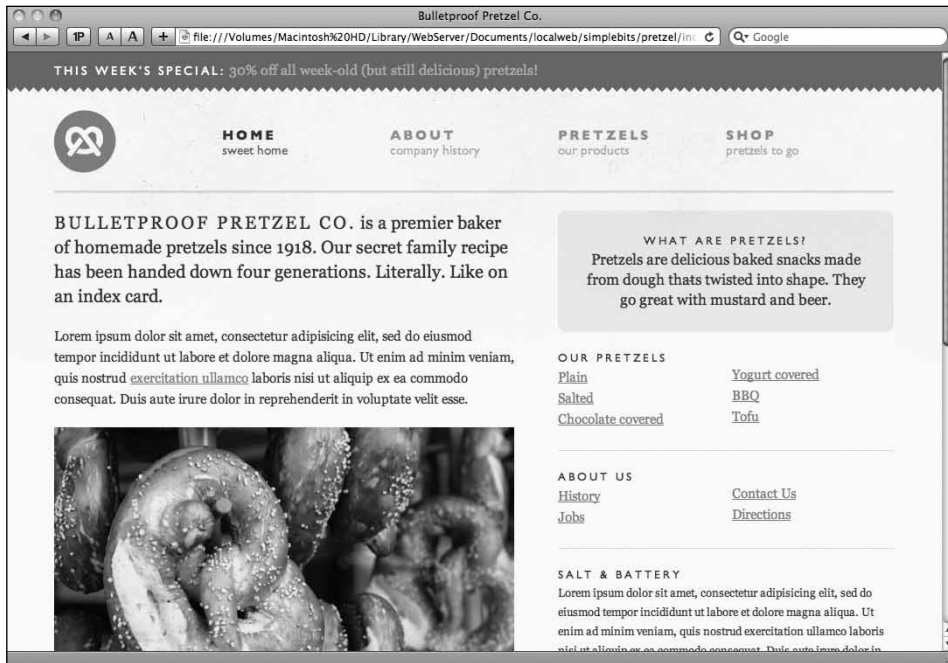


Рис. 9.1. В этой главе мы займемся разработкой целостного дизайна

Двухколоночный макет, используемый на сайте, кажется вам знакомым, так как подобные структуры очень популярны в Сети. Он состоит из заголовка с логотипом и панелью навигации, над которым находится рекламная строка — пространство для отображения новости дня. В колонке контента находится информация о компании и текст, в середине страницы находится изображение, занимающее всю ширину колонки. Вверху справа в боковой колонке располагается серия модулей с небольшим количеством контента, среди которых есть и прямоугольный элемент со скругленными углами. В нижней части страницы находится нижний колонтитул, растягивающийся на ширину обеих колонок.

Пуленепробиваемый подход

У нас есть привлекательный дизайн, при анализе которого становится очевидна его пуленепробиваемость. Шаг за шагом мы будем заниматься его воссозданием. Но сначала поговорим о том, почему я считаю его пуленепробиваемым.

Гибкий резиновый дизайн

Данный двухколоночный дизайн был реализован на базе резинового макета с помощью возможностей CSS, о которых мы говорили в главе 8 «Резиновые и эластичные макеты». При увеличении или уменьшении окна браузера макет будет подстраиваться под любую ширину экрана, которую захочет выбрать пользователь (рис. 9.2).

Как известно, выбор между фиксированным, резиновым или эластичным макетом — довольно сложная задача. Каждая разновидность макета имеет свои достоинства и недостатки, но резиновый макет — отличный способ предоставить пользователям немного больше контроля. Кроме того, вы делаете дизайн таким, что он будет отображаться на экранах с разным разрешением. Повторю еще раз: выбор типа макета зависит от множества факторов, включая требования дизайна, целевую аудиторию, статистику сайта и т. п. Несмотря на то что в данном примере я выбрал именно резиновую верстку, можно было использовать и фиксированный и эластичный макеты.

На рис. 9.2 видно, что элементы дизайна остаются неизменными, когда пользователь увеличивает или уменьшает окно браузера. Простая таблица сжимается, но оригинальное оформление сохраняется. Кроме того, дизайн адаптируется к ширине окна. Позднее я более подробно объясню адаптивный принцип веб-дизайна.

СОВЕТ

Итан Маркотт впервые использовал термин «адаптивный веб-дизайн» в статье для A List Apart (<http://www.alistapart.com/articles/responsive-web-design>), а затем в знаковой (и действительно мною рекомендуемой) одноименной книге (<http://abookapart.com/products/responsive-web-design>). Дизайнеры, использующие принципы пуленепробиваемости, обязательно должны прочитать их обе.

Адаптивный (отзывчивый) дизайн — это гибкая методика разработки веб-страниц, которые подстраиваются под любой размер окна, оставаясь удобочитаемыми и функциональными. Методика базируется на трех столпах — резиновая сетка, резиновые изображения и медиазапросы, позволяющие адаптировать макет к размерам окна просмотра. В этой главе мы рассмотрим каждый из компонентов.

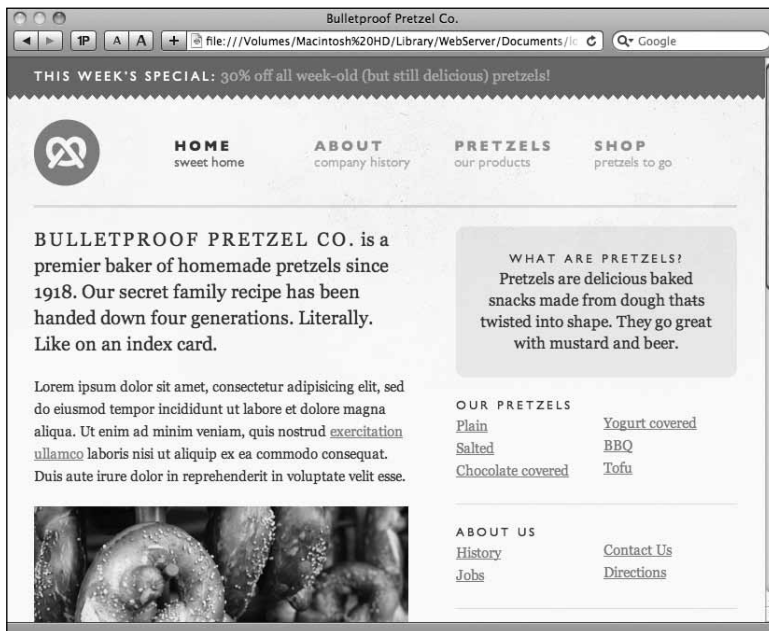
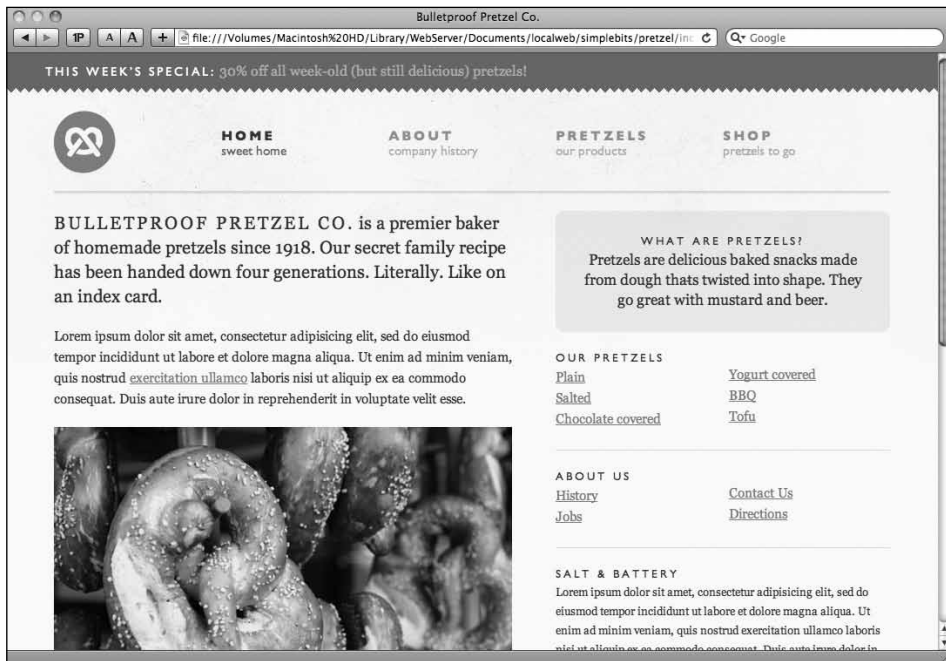


Рис. 9.2. Гибкий резиновый макет может подстраиваться под любую область просмотра (см. с. 272, продолжение)

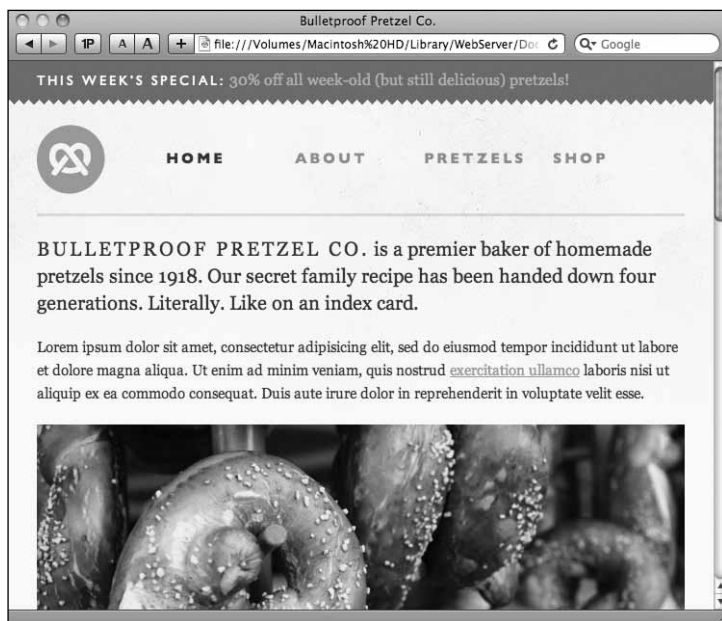


Рис. 9.2. (окончание)

ГИБКОЕ УПРАВЛЕНИЕ ТЕКСТОМ

В дизайне была реализована концепция, которую мы изучили в главе 1 «Гибкое управление текстом». Как показано на рис. 9.3, увеличение размера шрифта (базовый размер шрифта `font-size: 100%;`) на несколько шагов никак не влияет на целостность дизайна. Компоненты страницы увеличивается, а оформление остается неизменным. Заголовок и рекламное поле расширяются, чтобы вместить более крупный текст или дополнительный контент. Модуль со скругленными углами, расположенный в боковой панели, так же сохраняет форму.

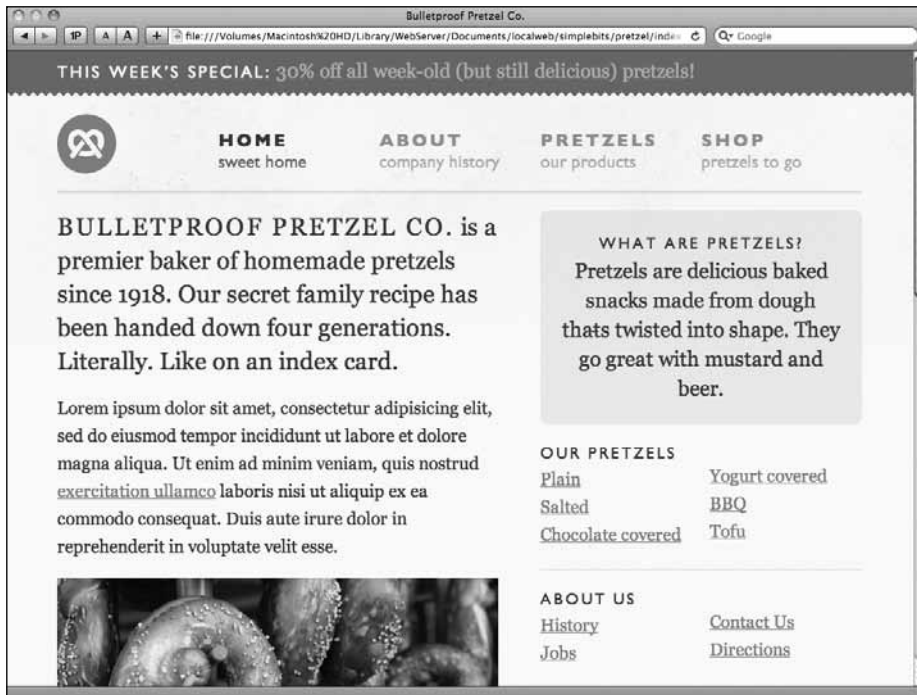


Рис. 9.3. Увеличение размера текста не влияет на внешний вид компонентов страницы

Но важнее всего то, что страница позволяет читателям подстроить размер текста под свои нужды. Пользователи с ослабленным зрением могут увеличивать текст, чтобы без напряжения читать его, независимо от того, какой браузер или устройство используется.

НЕТ КАРТИНОК? НЕТ CSS? НЕТ ПРОБЛЕМ!

Когда мы тестируем работоспособность дизайна страницы в разных ситуациях, мы можем проверить, что произойдет, если пропадут иллюстрации или не

поддерживается CSS (см. одноименную главу 6). На рис. 9.4 показана страница с отключенными изображениями. Люди, использующие медленные соединения, могут увидеть контент еще до того, как загрузятся все изображения. Если пользователь для экономии трафика и повышения скорости загрузки отключил загрузку изображений, страница остается читаемой и функциональной, поскольку фоновые цвета выполняют функцию фоновых изображений.

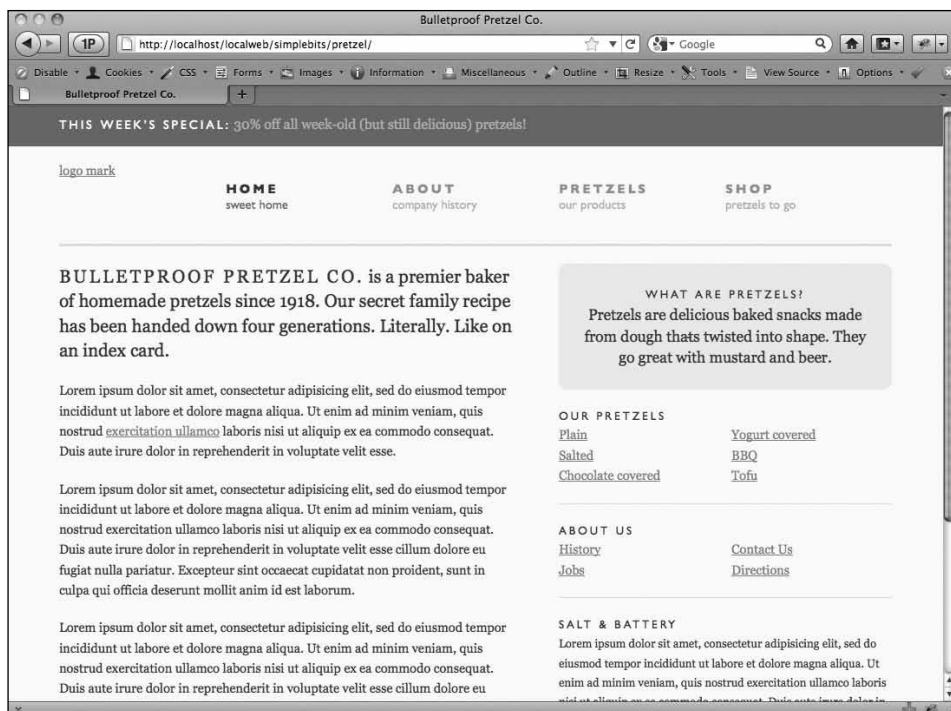


Рис. 9.4. Когда изображения отключены, страница остается читаемой благодаря использованию эквивалентных фоновых цветов

Если отключить CSS, то структура страницы пройдет тест на юзабилити, описанный в главе 6. Текстовые браузеры, вспомогательные приложения (например, приложение для чтения с экрана) или любые устройства, не поддерживающие CSS, смогут отобразить структуру страницы в любой ситуации. Как показано на рис. 9.5, даже когда стили пропадают, страница остается читаемой и логично структурированной.

Если убрать CSS и (или) отключить изображения, вы увидите, что сайт Bulletproof Pretzel Company готов и к этому: он остается читаемым, функциональным, надежным и, бесспорно, очень вкусным.

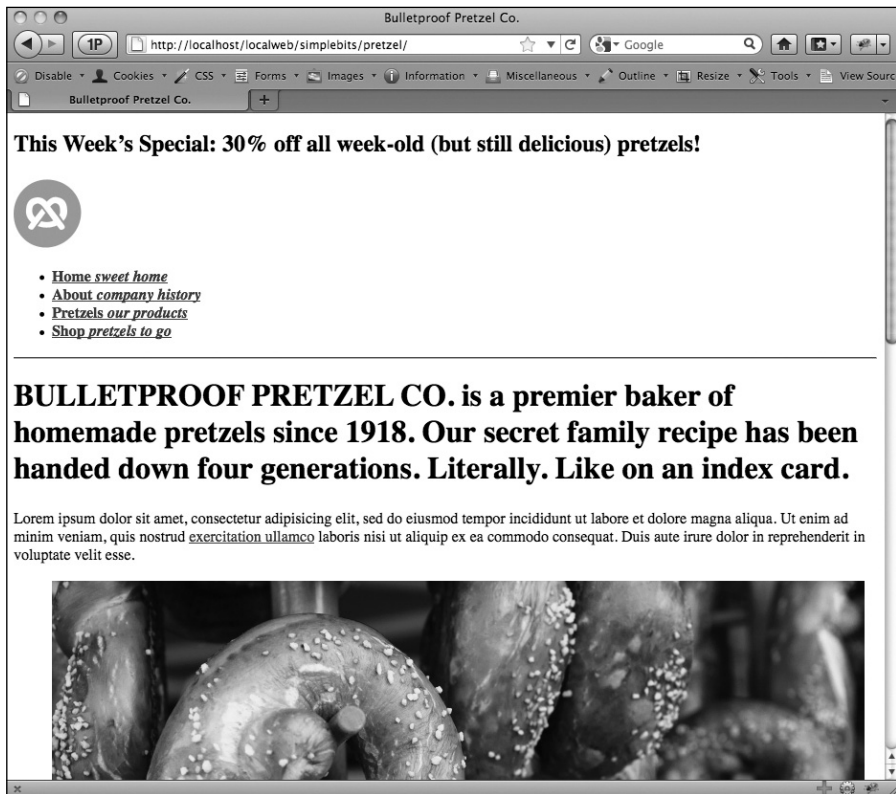


Рис. 9.5. «10-секундный тест на юзабилити» позволяет проверить приспособленность структуры страницы к отображению на различных устройствах

ИНТЕРНАЦИОНАЛИЗАЦИЯ

Еще одним важным побочным продуктом пуленепробиваемого дизайна является то, что он помогает решить задачи интернационализации. Например, на рис. 9.6 показана переведенная на немецкий язык версия сайта Bulletproof Pretzel Company, в которой английский текст заменен немецким.

Разрабатывая гибкий дизайн, вы гарантируете, что международные версии страницы смогут отображать текст и контент другого объема. Короткие слова или фразы на английском языке могут стать существенно более длинными строками (и наоборот). Кроме того, контекст может быть распределен по-другому, чтобы оказаться там, где его привыкли видеть пользователи, проживающие в другой части мира. Пуленепробиваемый дизайн поможет вам и с этой задачей. Это всего лишь еще один пример того, как можно подготовиться к неизвестному.

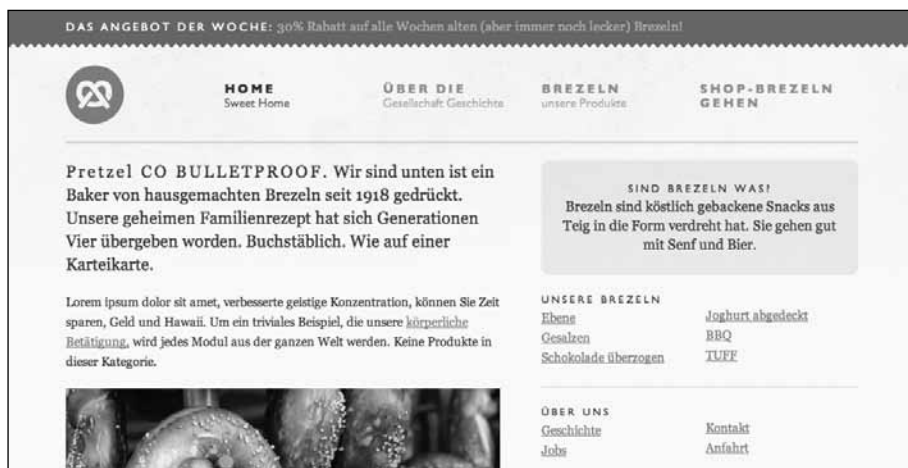


Рис. 9.6. Дизайн Bulletproof Pretzel Company, переведенный на немецкий язык с помощью Google Translate — <http://translate.google.com> (впрочем, качество перевода может вас и не устроить)

СТРУКТУРА

После того как мы кратко описали способы разработки пуленепробиваемого дизайна страницы, пройдемся по структуре дизайна. Я выделяю шаги, которые необходимы, чтобы обеспечить гибкость, не поступившись при этом привлекательным внешним видом. Начнем с места, на котором мы остановились в главе 8, когда в первый раз использовали гибкую двухколоночную верстку. Затем последовательно пройдемся по всем компонентам страницы, освежая в памяти знания, полученные в предыдущих главах. Мы сделаем дизайн адаптивным с помощью гибкой сетки и изображений и добавим медиазапросы (CSS3) для адаптации макета к уменьшению размеров окна просмотра.

РАЗМЕТКА

Начнем создание гибкого двухколоночного макета с базовой разметки. Мы используем элементы HTML5 для структурирования документа.

Структура разметки рекламного поля, заголовка, двух колонок и нижнего колонтитула будет выглядеть следующим образом:

```
<div id="special">
  <h2><strong> This Week's Special:</strong> 30% off
    ▶ all week-old (but still delicious) pretzels!</h2>
</div>
<div id="wrap">
```

```

<header role="banner">
  <div id="logo">
    <a href="/"></a>
    </div>
    <nav role="navigation">
      ...навигационная панель...
    </nav>
  </header>
<div id="main" role="main">
  ...основной контент...
</div>
<div id="sidebar" role="complementary">
  ...боковая панель...
</div>
<footer role="contentinfo">
  ...здесь будет нижний колонтитул...
</footer>
</div> <!-- /wrap -->

```

Думаю, что мы удачно выбрали элементы разметки. Будем использовать элементы HTML5 — `<header>`, `<nav>` и `<footer>`, — а также ARIA-роли (о которых уже говорили ранее) для разметки основных разделов страницы.

ПРИМЕЧАНИЕ

Как и в предыдущих примерах, мы будем обнулять поля и отступы, размеры шрифта часто используемых элементов в таблице стилей. За более подробной информацией обратитесь к введению.

ОСНОВНЫЕ СТИЛИ

После того как мы создали основную структуру разметки, перейдем к стилям CSS. Добавим информацию о шрифте в основной элемент `<body>`:

```

body {
  font-family: Georgia, serif;
  font-size: 100%;
  color: #4e443c;
  background: #fcfaf5;
}

```

Используя знания о гибком тексте, полученные в главе 1, сделаем базовый размер шрифта `font-size` равным 100% (около 16px при настройках по умолчанию). Позднее мы скорректируем размер текста с помощью `em`. Мы выбрали шрифт Georgia (это вполне безопасно как для Mac, так и для Windows) и определили цвета фона и текста.

СТРУКТУРА МАКЕТА

Добавим правила для создания двухколоночного макета. Сделаем колонку основного контента и боковую колонку плавающими. Очистка этих плавающих элементов будет возложена на элемент `<footer>`, находящийся под ними. Мы зададим размер средника, добавив отступ `50px` справа от `#main`. Чтобы не создавать дополнительных контейнеров `<div>`, мы используем свойство `box-sizing` (о котором говорили в главе 8 «Резиновые и эластичные макеты») для отступа внутри элемента (вместо добавления к ширине `60%`).

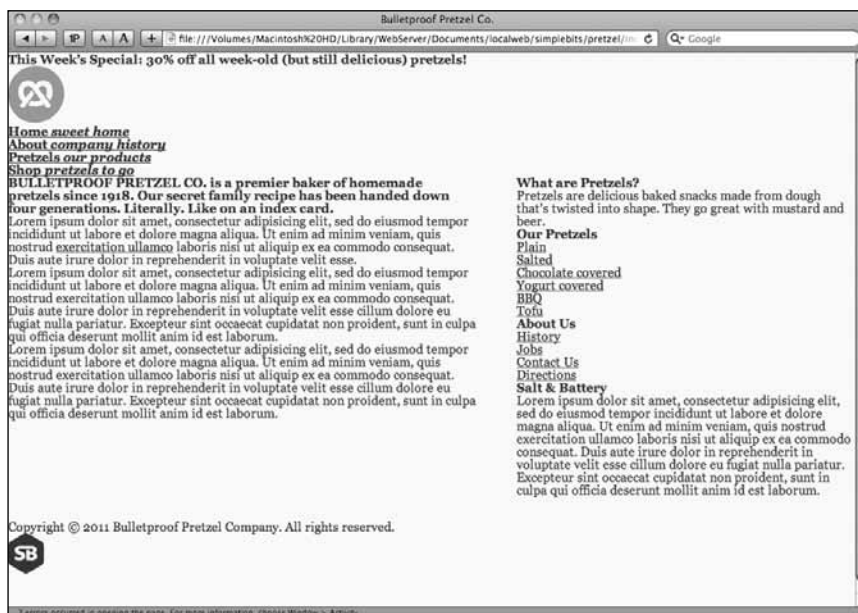


Рис. 9.7. Колонки находятся друг напротив друга. Ширина каждой из них задана в процентах

```
/* структура макета */
#main {
    float: left;
    width: 60%;
    margin: 0 0 30px 0;
    padding: 0 50px 0 0;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}
#sidebar {
    float: right;
```

```

width: 40%;
margin: 0 0 30px 0;
}
footer[role="contentinfo"] {
clear: both;
}

```

ПРИМЕЧАНИЕ

Пока что я пропустил фотографию, которая находится в колонке основного контекста, позднее мы сделаем гибкой и ее.

Выравниваем колонку контента по левому краю, ее ширина составит 60%, а боковую панель сместим к правому краю и отдадим ей остальные 40%. Под колонками расположен нижний колонтитул, который их очищает. На рис. 9.7 показаны результаты после применения стилей. Внешний вид колонок (и пространство между ними) постепенно приближается к нашей цели.

ГИБКАЯ СЕТКА

Процентные значения, которые мы присвоили ширине колонок, не взяты с потолка. Они стали результатом расчета сетки, которая используется в нашем дизайне. На рис. 9.8 показана простая сетка, состоящая из пяти колонок, на базе которой строится наш дизайн. Каждая из пяти колонок занимает 20% общей ширины макета.



Рис. 9.8. Гибкая сетка из пяти колонок

Гибкая сетка — первый из трех шагов к отзывчивому дизайну, о котором мы поговорим немного позднее.

ЗАДАЕМ MAX-WIDTH

Зададим максимальное (`max-width`) значение ширины макета 960 пикселей и отцентрируем дизайн по середине поля просмотра.

Контейнер `<div id="wrap">` содержит всю страницу, кроме верхнего рекламного баннера (так как он занимает всю ширину страницы). Мы добавили несколько правил: центрование и ограничение ширины до 960px. Размер может быть меньше 960 пикселей, но не сможет превысить это значение, что позволит сохранять пропорции колонок. Позднее мы оформим и рекламную строку, расположив ее по центру и ограничив ширину контента.

```
/* структура макета */
#wrap {
    max-width: 960px;
    margin: 0 auto;
    padding: 0 30px;
}
#main {
    float: left;
    width: 60%;
    margin: 0 0 30px 0;
    padding: 0 50px 0 0;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}
#sidebar {
    float: right;
    width: 40%;
    margin: 0 0 30px 0;
}
footer[role="contentinfo"] {
    clear: both;
}
```

ПРИМЕЧАНИЕ

Браузер IE6 не поддерживает свойства `max-width` и `min-width`. Не забудьте об этом, если использование этого браузера будет обязательным условием.

Указав значение `auto` для левого и правого поля, мы отцентрировали макет в браузере (рис. 9.9). Отступ 30px справа и слева не позволят ему выйти на край поля просмотра, когда ширина области обозревателя будет меньше 960px.

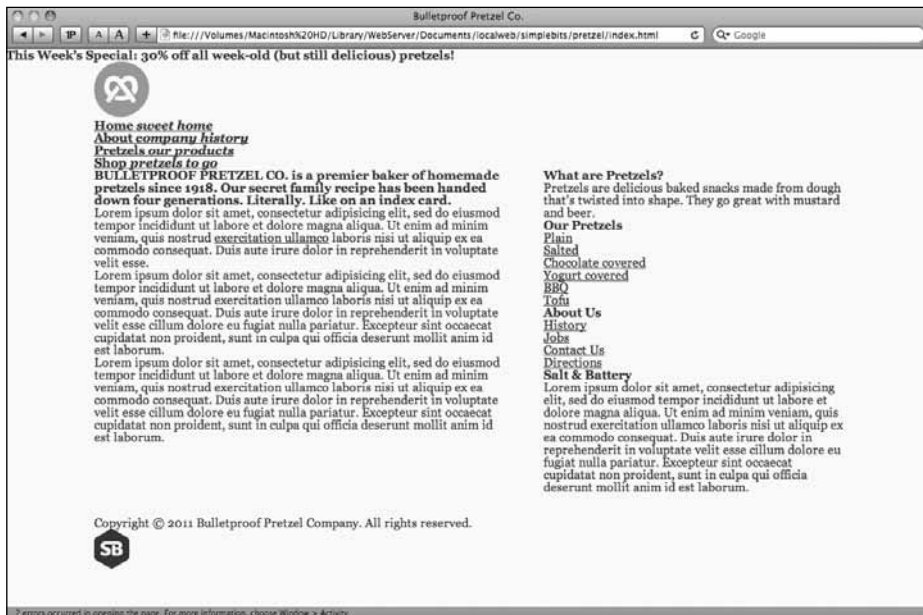


Рис. 9.9. Теперь область основного контента находится в центре окна просмотра

Давайте зададим стиль верхней рекламной строки, учитывая, что ее фон должен захватить всю ширину окна просмотра. При этом контент должен быть привязан к сетке макета.

СОВЕТ

При размещении фонового изображения я задаю значение по горизонтали –80%, чтобы иметь возможность добавить небольшой эффект при изменении размера браузера. Когда повторяющиеся изображения растягиваются по горизонтали с большим отрицательным процентным значением, они смещаются в противоположном направлении от края, который в браузере считается правым. Этот простой способ позволяет облегчить работу с горизонтально повторяющимися фоновыми изображениями.

```
/* специальная панель */
#special {
    margin: 0 0 20px 0;
    padding: 10px 30px 16px 30px;
    line-height: 1.3;
    color: rgba(255,255,255,.5);
    background: #6a8087 url(img/special-bg.png) repeat-x -80% 100%;
}
```





```
#special h2 {
    max-width: 960px;
    margin: 0 auto;
    font-weight: normal;
}

#special h2 strong {
    font-family: "Gill Sans", Helvetica, Arial, sans-serif;
    font-size: .9em;
    font-weight: normal;
    text-transform: uppercase;
    letter-spacing: 3px;
    color: #fff;
}
```

Кроме полей, отступов и шрифта на рис. 9.10 видно, как повторяющееся фоновое изображение создает зубчатый нижний край рекламной строки.

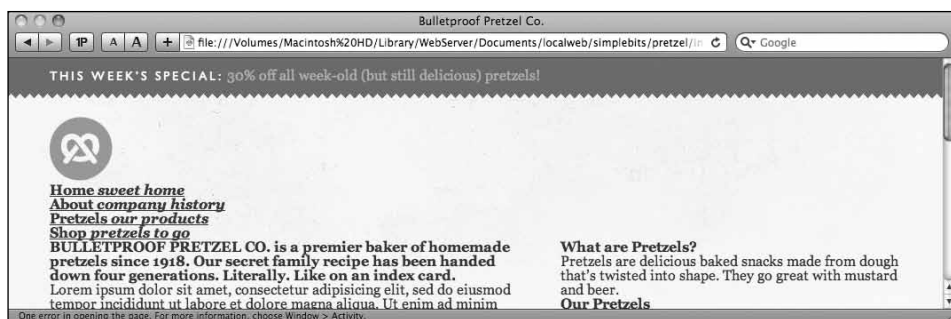


Рис. 9.10. Зубчатый нижний край рекламной строки

Прозрачная заливка зубцов фонового изображения (рис. 9.11) пропускает цветной фон. При повторении изображения по горизонтали создается желаемый эффект зубчатой границы. Из главы 3 «Эластичные строки» мы знаем, что если выровнять фоновое изображение по нижнему краю, то высоту плашки можно будет увеличить, подстраивая под размер контента или кегль шрифта.

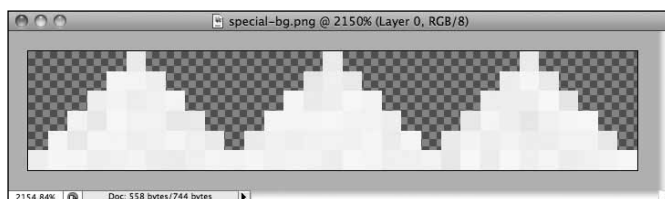


Рис. 9.11. Увеличенный фрагмент фонового изображения с прозрачной заливкой зубцов

Заголовок

После завершения работы над баннером перейдем к следующему элементу — заголовку. Прежде всего мы назначим поля и отступы, а также нижнюю границу, воспользовавшись ARIA-ролью, которую мы задали в `<header>`.

```
header[role="banner"] {
    margin: 0 0 20px 0;
    padding: 0 0 20px 0;
    border-bottom: 3px solid rgba(68,44,24,.12);
}
```

Вставим логотип и панель навигации. Если вы еще раз взглянете на рис. 9.8, то вспомните, что макет основан на простой сетке из пяти колонок. При этом логотип и каждый из четырех элементов панели навигации привязаны к соответствующим колонкам. Благодаря расчетам мы знаем, что ширина каждой из колонок составляет 20% общей ширины. Поэтому логотип и элементы панели навигации мы сместим влево, чтобы они оказались в одной строке по горизонтали. Кроме того, все элементы будут иметь ширину 20%.

```
header[role="banner"] {
    margin: 0 0 20px 0;
    padding: 0 0 20px 0;
    border-bottom: 3px solid rgba(68,44,24,.12);
}
/* логотип */
#logo {
    float: left;
    width: 20%;
}
/* панель навигации */
nav[role="navigation"] ul li {
    float: left;
    width: 20%;
    margin: 20px 0 0 0;
}
nav[role="navigation"] ul li a {
    float: left;
    width: 100%;
    ...
}
```

Я не стал приводить все стили, использованные для оформления навигационных ссылок, но самые важные настройки остались: например, настройка ширины логотипа и элементов системы навигации, `width: 20%;`, а также ARIA-метка роли элемента `<nav>`, позволяющая использовать эти стили для панели навигации (если на странице будут использоваться дополнительные элементы `<nav>`).

На рис. 9.12 показан результат нашей работы. Теперь логотип и панель навигации заняли свои места на сетке из пяти колонок. Так как мы задавали ширину компонентов в процентах, макет остается гибким и эластичным, а пропорции сетки неизменны.

ПРИМЕЧАНИЕ

Я намеренно опускаю свойства `width` и `height` элемента ``, чтобы мы могли использовать CSS.

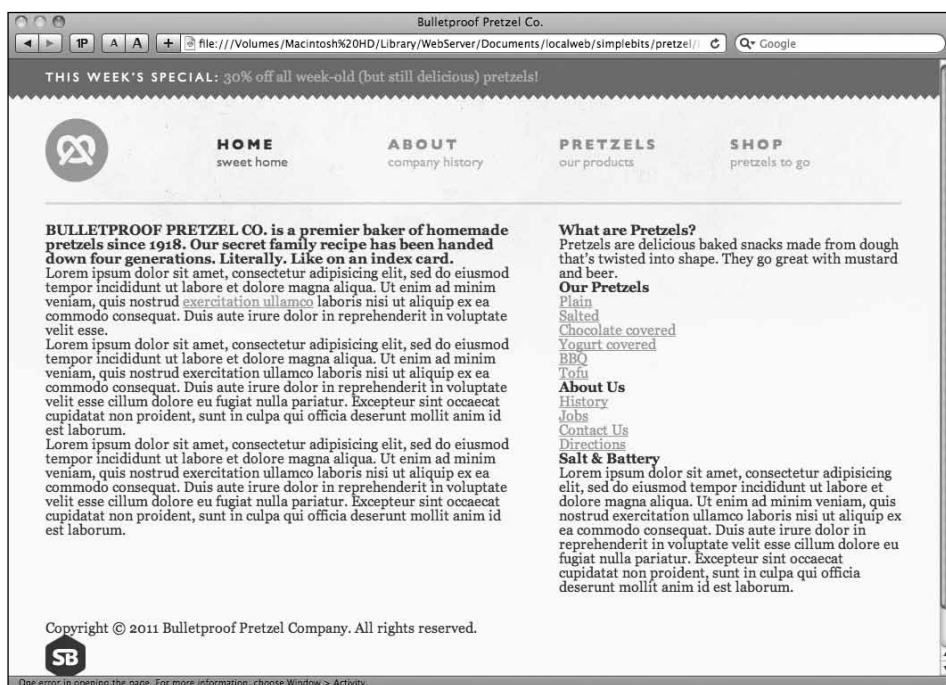


Рис. 9.12. Макет постепенно обретает форму

ГИБКИЕ ИЗОБРАЖЕНИЯ

Вторым этапом разработки отзывчивого дизайна является создание гибких изображений. Мы используем удивительно гибкую сетку, но представьте, что если бы в колонке основного контента была фотография с шириной больше колонки?

Вставим фотографию в разметку. Для этого используем `<figure>` — новый и весьма подходящий элемент HTML5. Элемент `<figure>` предназначен для

отображения и аннотирования иллюстраций, диаграмм, фотографий и т. п., на которые есть ссылки из основного контента, как в нашем случае.

```
<figure></figure>
```

Изображение с аппетитными крендельками намного шире колонки основного контента (даже при максимальной ширине окна). На рис. 9.13 показано, что происходит, когда в макет попадает слишком широкое изображение. Неужели наше решение не является пуленепробиваемым?

ПРИМЕЧАНИЕ

Браузеры IE6 и более старые версии не поддерживают свойство `max-width`. Если же необходим этот браузер, придется использовать другое решение (пусть и ценой головной боли). Иногда (в зависимости от размера изображения и максимальной ширины контейнера) может пригодиться свойство `width: 100%`. Но осторожно: размер маленького изображения может увеличиться настолько, что вы увидите размытую картинку.



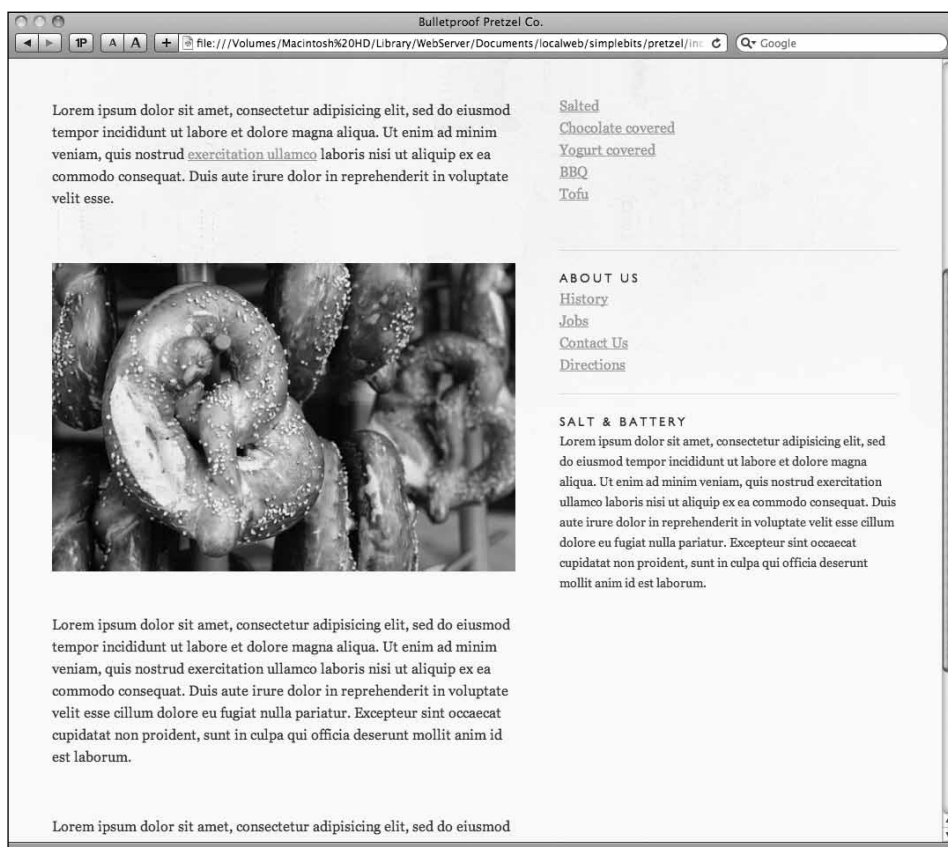
Рис. 9.13. Фотография намного шире колонки основного контента

Мы можем разрешить проблему, опираясь на книгу Итана Маркотта по отзывчивому дизайну. Для этого сделаем изображения гибкими. Нам понадобится лишь одно простое правило CSS:

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

Этот код заставит все изображения, размещенные в элементах `<figure>`, заполнить 100% ширины контейнера. Иными словами, ширина изображения не превысит ширину колонки основного контента, а высота будет автоматически пересчитываться, сохраняя пропорции сторон.

Мы можем менять ширину окна и наслаждаться, глядя как размер изображения подстраивается под сетку (рис. 9.14). Последние версии браузеров великолепно изменяют размеры, сохраняя картинку четкой.



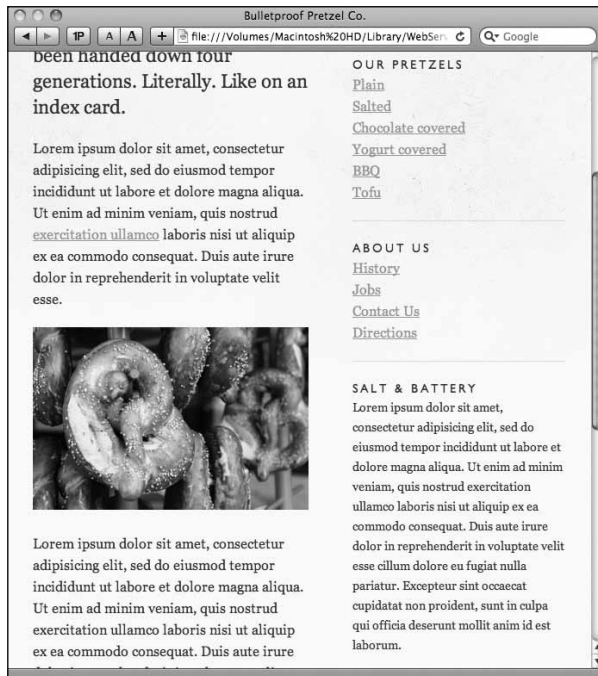


Рис. 9.14. Теперь размеры фотографии меняются вместе со всем макетом

ДРУГИЕ ТИПЫ МЕДИАЭЛЕМЕНТОВ

Великолепный трюк, в основе которого лежит `max-width`, помогает и с другими разновидностями медиаэлементов, например с видео. Воспользуйтесь этим же методом для внедрения видеороликов:

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

Размеры объекта изменятся точно так же, как это произошло с фотографией, вставленной в макет.

СОВЕТ

Более подробную информацию о гибких изображениях и мультимедиа можно прочитать в статье Итана Маркотта «Резиновые изображения» (Fluid Images, <http://alistapart.com/articles/fluid-images/>).

БОВОКАЯ ПАНЕЛЬ

В боковую панель я хочу добавить несколько элементов. Один из них — прямоугольный блок со скругленными углами и текстом «What are pretzels?» (рис. 9.15). Мы используем CSS3 для создания скругленных углов и полупрозрачной фоновой заливки, о котором говорили в главе 5 «Несокрушимые элементы».

Разметка довольно проста. Я создаю контейнер `<div>` с классом, который будет включать заголовок и абзац текста. Мы используем класс `.callout` для задания стиля элемента, чтобы добавить фон, скруглить углы и назначить шрифты.

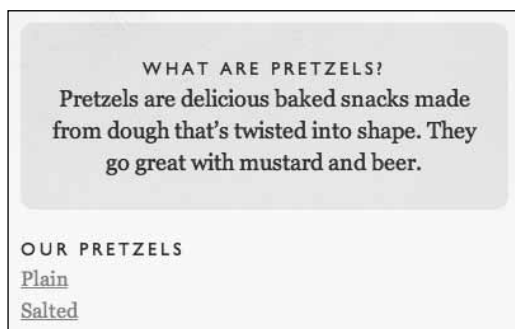


Рис. 9.15. Создадим прямоугольный блок со скругленными углами с помощью CSS3

```
<div class="callout">
  <h3>What are Pretzels?</h3>
  <p>Pretzels are delicious baked snacks made from dough
    ▶ that's twisted into shape. They go great with
    ▶ mustard and beer.</p>
</div>
```

Прежде всего зададим поля и отступы и полупрозрачную фоновую плашку.

```
/* блок выноски */
div.callout {
  margin: 0 0 20px 0;
  padding: 25px;
  text-align: center;
  background: rgba(223,174,42,.2);
}
```

На рис. 9.16 показаны результаты нашей работы. Теперь прямоугольный модуль с отступами занял свое место. Мы использовали палитру RGBA для задания цвета фоновой заливки и задали коэффициент непрозрачности 0,2 (или 20%). Палитра RGBA позволяет создавать эффект полупрозрачности с помощью CSS. Это очень гибкий, настраиваемый и простой инструмент. К сожалению, он не поддерживается IE8 и более старыми версиями, поэтому нам придется внести дополнительные изменения. Мы можем и должны предусмотреть альтернативные варианты обработки для браузеров, не поддерживающих RGBA. В данном случае мы зададим резервный цвет, указав его до правила RGBA. Старые браузеры проигнорируют RGBA и отобразят сплошную заливку. А браузеры, поддерживающие RGBA, будут использовать правило с RGBA.

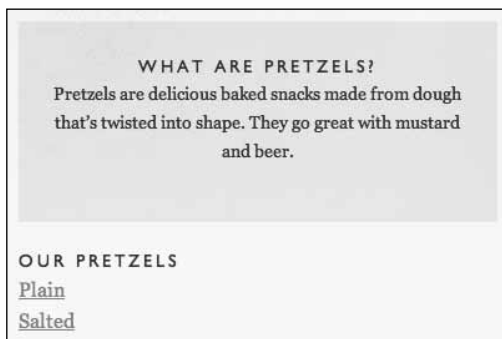


Рис. 9.16. Для задания цвета заливки мы используем палитру RGBA

```
/* блок выноски */
div.callout {
  margin: 0 0 20px 0;
```





```
padding: 25px;
text-align: center;
background: #f5eacb; /* для браузеров, не поддерживающих RGBA */
background: rgba(223,174,42,.2);
}
```

Скруглим углы с помощью `border-radius`, как и ранее. Опять же, мы будем использовать правила с префиксами разработчиков для браузеров WebKit и Mozilla. В конце всегда будет идти правило со свойством CSS3, предназначенное для браузеров, которые уже поддерживают его или будут поддерживать в будущем.

```
/* блок выноски */
div.callout {
    margin: 0 0 20px 0;
    padding: 25px;
    text-align: center;
    background: rgba(223,174,42,.2);
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    border-radius: 10px;
}
```

На рис. 9.17 показаны результаты. Углы будут скруглены в большинстве браузеров (включая IE9+). В тех браузерах, которые не поддерживают `border-radius`, углы останутся обычными.

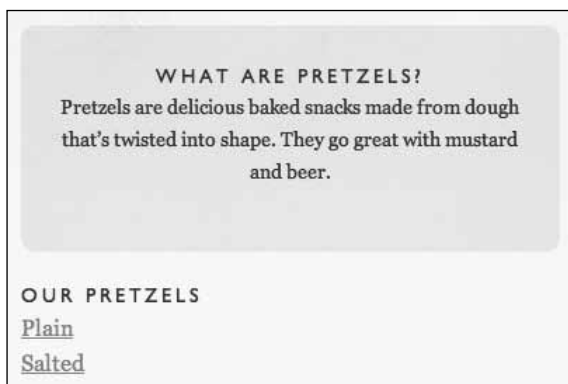


Рис. 9.17. Используем `border-radius` для скругления углов

Осталось только позаботиться о стиле текста, находящегося внутри прямоугольного блока. Мы удалим поля, используемые по умолчанию, увеличим кегль шрифта и интерлиньяж.

```

/* блок выноски */
div.callout {
    margin: 0 0 20px 0;
    padding: 25px;
    text-align: center;
    background: rgba(223,174,42,.2);
    border-radius: 10px;
}
#sidebar div.callout p {
    margin: 0;
    font-size: 1.1em;
    line-height: 1.4;
}

```

Итак, мы создали гибкий модуль (рис. 9.18), который можно использовать в любом месте макета. При его разработке мы использовали современные возможности CSS3, но если просматривать наш элемент в старых браузерах, ничего страшного не произойдет, вы увидите обычный прямоугольник со сплошной заливкой. Самое замечательное, что мы смогли отказаться от изображений и скриптов при разработке визуального дизайна.

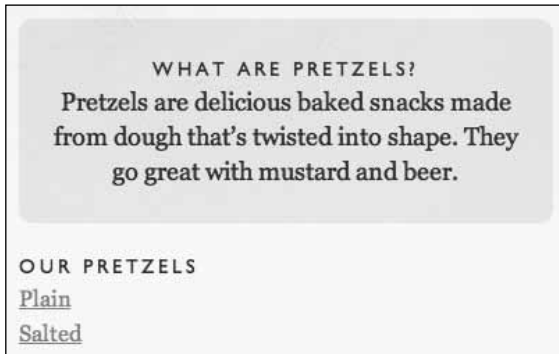


Рис. 9.18. Завершенный прямоугольный модуль

МАКЕТ ИЗ НЕСКОЛЬКИХ КОЛОНОК НА БАЗЕ CSS3

Далее на боковой панели расположены два небольших списка со ссылками, они озаглавлены «Our Pretzels» и «AboutUs» (рис. 9.19). Ссылки довольно короткие, поэтому в колонке (при максимальной ширине) остается много свободного пространства.

В CSS3 существует множество готовых модулей макетов с разной степенью проработки. Один из них — Multi-Column Layout (<http://www.w3.org/TR/css3-multicol/>) — я использую в ситуациях, когда планирую постепенно

дорабатывать макет. Многоколоночные макеты отображаются в большинстве современных браузеров (Safari, Chrome, Firefox, Opera и IE10+). Как правило, они используются, чтобы распределить по нескольким колонкам узкие списки коротких ссылок. В браузерах, не поддерживающих CSS, многоколоночность пропадет.

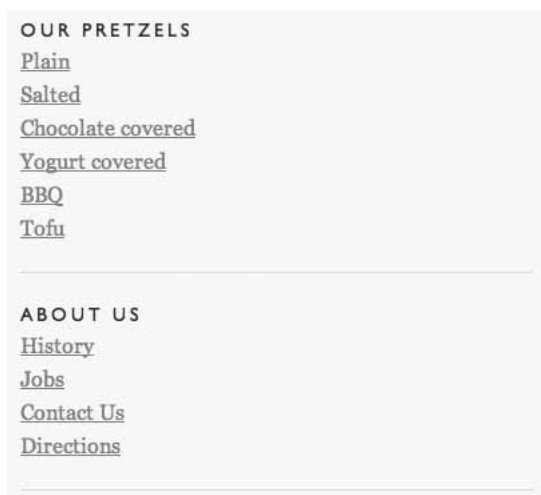


Рис. 9.19. Два вертикальных неупорядоченных списка на боковой панели

Добавим соответствующие правила к спискам ссылок. Как и с другими свойствами CSS3, будем использовать префиксы разработчиков, чтобы браузеры «понимали» эти правила.

```
#sidebar ul {  
    padding-bottom: 20px;  
    border-bottom: 1px solid rgba(68,44,24,.12);  
    -webkit-column-count: 2;  
    -moz-column-count: 2;  
    column-count: 2;  
}
```

СОВЕТ

У многоколоночного макета есть и другие свойства: `column-gap` (для задания средника), `column-width` (для задания ширины колонок) и `column-rule` (для вставки вертикальных линий между колонками). Более подробную информацию можно найти на сайте <http://www.quirkmode.org/css/multicolumn.html>.

Мы создали две колонки для списков ссылок. Правило позволяет распределить пункты списка по двум колонкам так, как мы хотим (рис. 9.20). Вам не нужно указывать ширину или использовать плавающие элементы.

Что мне нравится в многоколоночных макетах, так это то, что они отлично совместимы с резиновой версткой. Использованное свойство автоматически делит доступное пространство на равные колонки.



Рис. 9.20. Использование многоколоночного макета позволяет автоматически разделить список на две равные колонки

И помните: если браузер не поддерживает многоколоночный макет, то ссылки превратятся в обычный вертикальный список. Таким образом, никакого негативного воздействия на дизайн не произойдет.

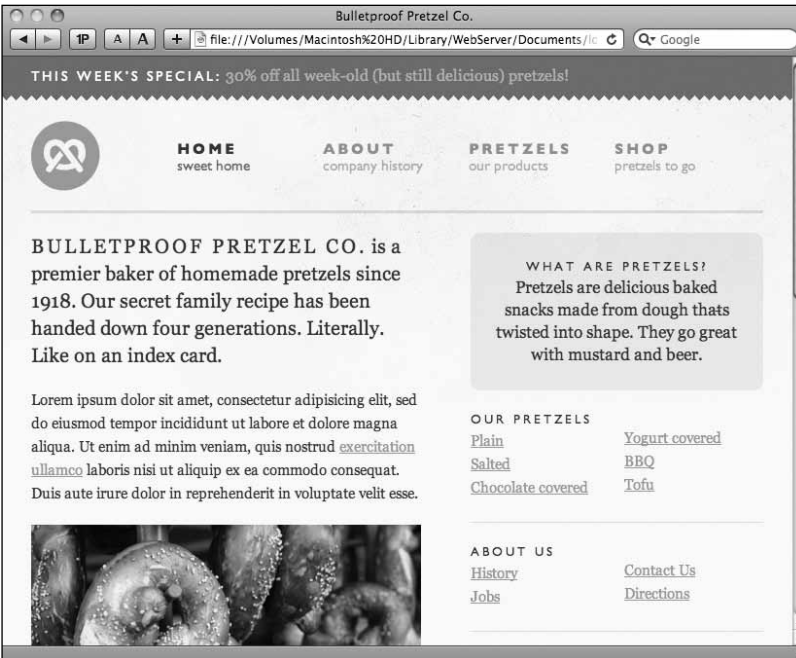
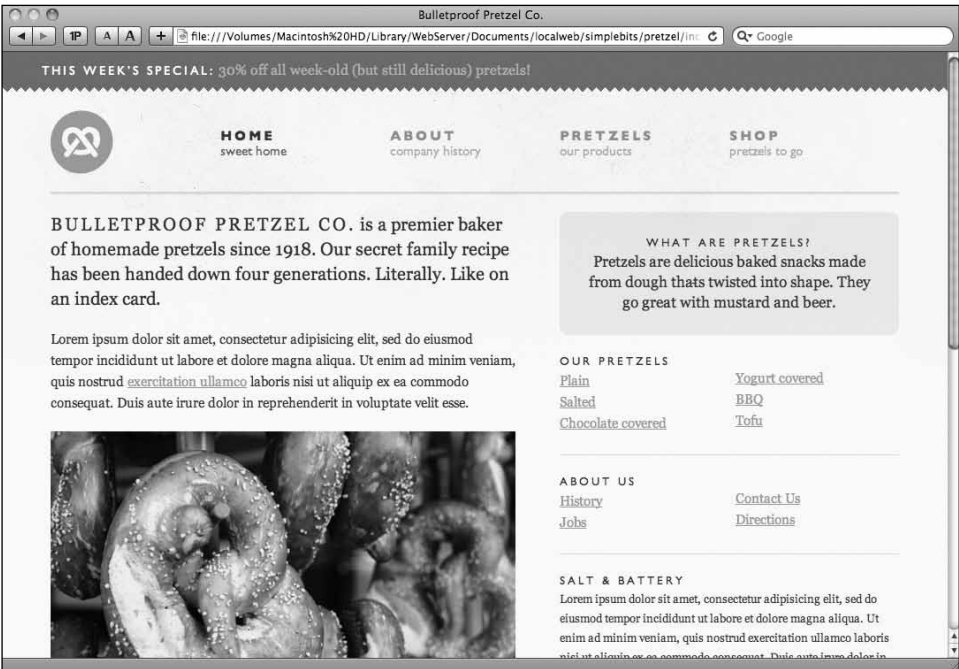
МАГИЯ МЕДИАЗАПРОСОВ

Нам остается последний элемент адаптивного веб-дизайна — медиазапросы.

Медиазапросы позволяют «программировать» CSS-правила под конкретные критерии. В нашем случае медиазапросы помогут создать несколько «контрольных точек», которые позволят нам изменять дизайн в зависимости от ширины окна браузера (рис. 9.21).

Существует много вариантов использования медиазапросов, но мы с их помощью будем изменять таблицу стилей. Нам понадобится отменить некоторые из основных стилей, чтобы макет элегантно вписался в размер области просмотра при уменьшении окна обозревателя.

Если пользователь открыл сайт на планшете, iPad или коммуникаторе, дизайн должен автоматически подстроиться под эти устройства. Это пуленепробиваемость в наилучшем своем проявлении. Мы уже разработали резиновую сетку с изображениями, меняющими размер, поэтому осталось совсем немного — внести изменения с помощью медиазапросов.



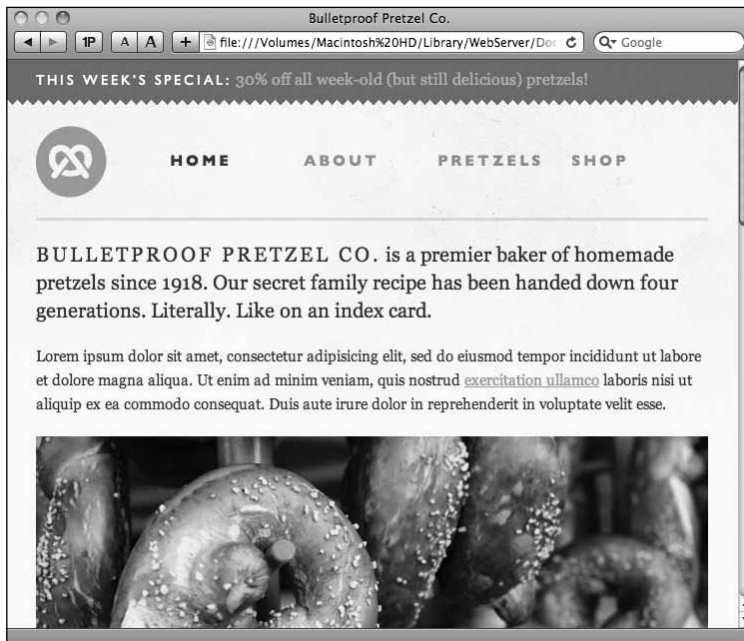


Рис. 9.21. Макет становится гибким благодаря медиазапросам

МЕТАЭЛЕМЕНТ VIEWPORT

Перед тем как вставить медиазапросы, нужно добавить немного разметки в `<head>` документа. Некоторые устройства с небольшими экранами (в том числе iPhone и iPad) сначала пытаются отобразить страницу на всю ширину, а потом уменьшить ее, чтобы пользователь при просмотре увеличивал фрагменты страницы. Например, iPhone отображает страницу с шириной 980px (по умолчанию), а затем сжимает ее до ширины экрана.

Разрабатывая резиновый адаптивный дизайн, мы хотим, чтобы ширина макета подстраивалась под фактическую ширину экрана. При этом пользователь не должен увеличивать масштаб страницы. К счастью, у нас есть элемент `viewport`, который позволяет сделать ширину макета, равной ширине окна обозревателя устройства. Так как макет резиновый, он сожмется и адаптируется к нашим требованиям.

При разработке адаптивных дизайнов добавляйте в `<head>` следующую строку:

```
<meta name="viewport" content="width=device-width" />
```

ДОБАВЛЕНИЕ АДАПТИВНЫХ СТИЛЕЙ

После того как мы разобрались с полем просмотра мобильных устройств, добавим в нижнюю часть таблицы стилей пару блоков медиазапросов для «программирования» свойства `max-width`. Я хочу задать контрольные точки для адаптации дизайна при ширинах 800px и 550px. Некоторые дизайнеры задают «контрольные точки», отталкиваясь от стандартной ширины устройства (портретная и альбомная ориентация iPad, iPhone и т. п.). Однако в данном случае я тщательно изучаю оформление при разных значениях ширины и только после этого выбираю «контрольные точки».

Добавим новый раздел — «адаптивность» — в таблицу стилей. В нем мы определим две контрольные точки для `max-width` (800px или меньше и 550px или меньше). Медиазапрос состоит из двух частей: тип (в данном случае `screen`) и собственно запрос (в скобках).

```
/* адаптивность
----- */
@media screen and (max-width: 800px) {
    ... правила ...
}
@media screen and (max-width: 550px) {
    ... правила ...
}
```

Мы начинаем с ширины окна браузера не более 800px, затем скорректируем элементы оформления, которые требуется переформатировать. На рис. 9.22 показано поведение резиновой сетки на экране шириной около 800px.

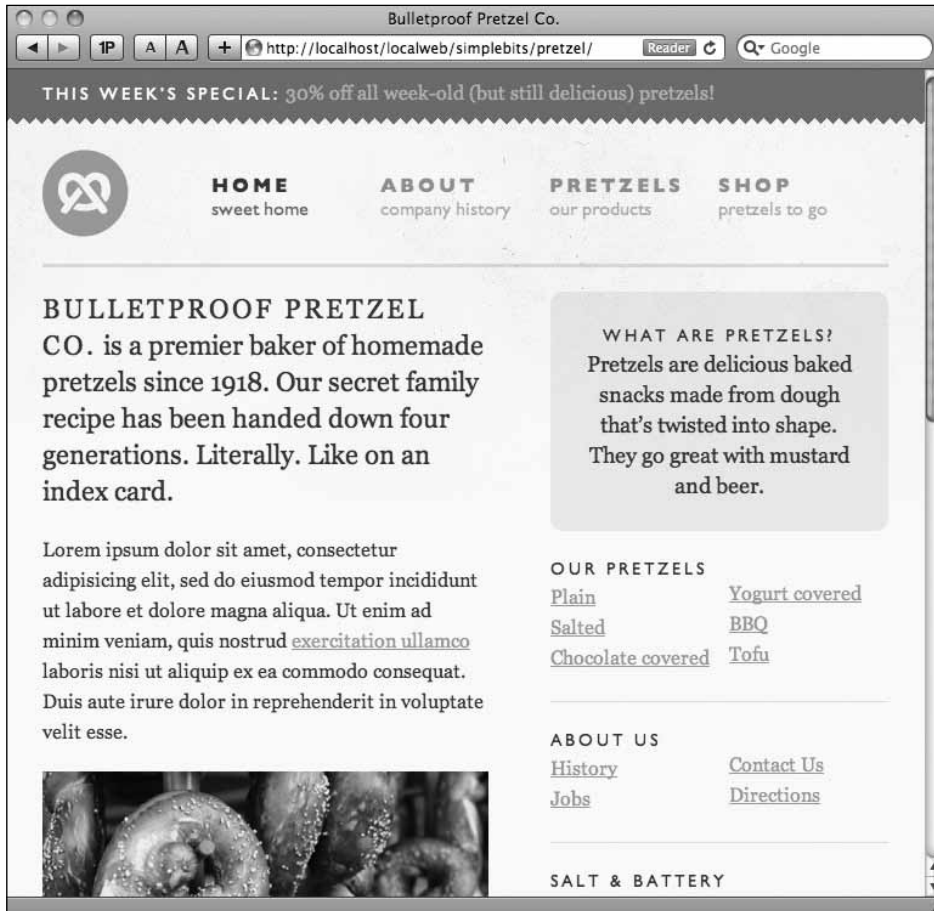


Рис. 9.22. Сайт при ширине экрана приблизительно 800px

Все выглядит не так уж плохо, однако колонки кажутся слегка сплюснутыми. Если мы продолжим сжимать дизайн, то пострадает целостность системы навигации.

Добавим в медиазапросы несколько правил (`max-width: 800px`). Колонки перестанут быть плавающими, а вторые строки каждого элемента навигации окажутся скрытыми. Это даст макету больше «воздуха». Мы уменьшим размер шрифта для системы навигации и контейнеров `#main` и `#sidebar`.

/* адаптивность

```
----- */
@media screen and (max-width: 800px) {
  nav[role="navigation"] ul li {
```



```

✎
margin: 26px 0 0 0;
}
nav[role="navigation"] ul li a {
    font-size: .9em;
}
nav[role="navigation"] ul li a em {
    display: none;
}
#main,
#sidebar {
    float: none;
    width: auto;
    padding: 0;
    font-size: .9em;
    line-height: 1.5;
}
}
@media screen and (max-width: 550px) {
    ... правила ...
}

```

ПРИМЕЧАНИЕ

По возможности постарайтесь не использовать свойство `display: none` для сокрытия контента в узких колонках. Ведь данные все равно загружаются в устройство, поэтому отказ от их использования не является оптимальным решением.

На рис. 9.23 показаны результаты после добавления правил в медиазапросы, использующего эти стили (ширина браузера меньше 800px). Двухколоночный макет превращается в одноколоночный.

Теперь наш макет можно читать на устройствах с шириной экрана менее 800px. При этом две плавающие колонки превращаются в одну прокручиваемую.

Перейдем к следующему этапу и займемся уже 550px-м полем просмотра, добавив правила в медиазапросы.

Следующие правила позволят сделать страницу привлекательной на устройствах с небольшим экраном:

- уменьшите отступы, чтобы в то же пространство вписался больший объем контента;
- расположите логотип и навигационные ссылки по центру, чтобы увеличить пространство для этих элементов без добавления контейнера;
- уменьшите кегль шрифта элементов навигации и заголовка `<h1>`.

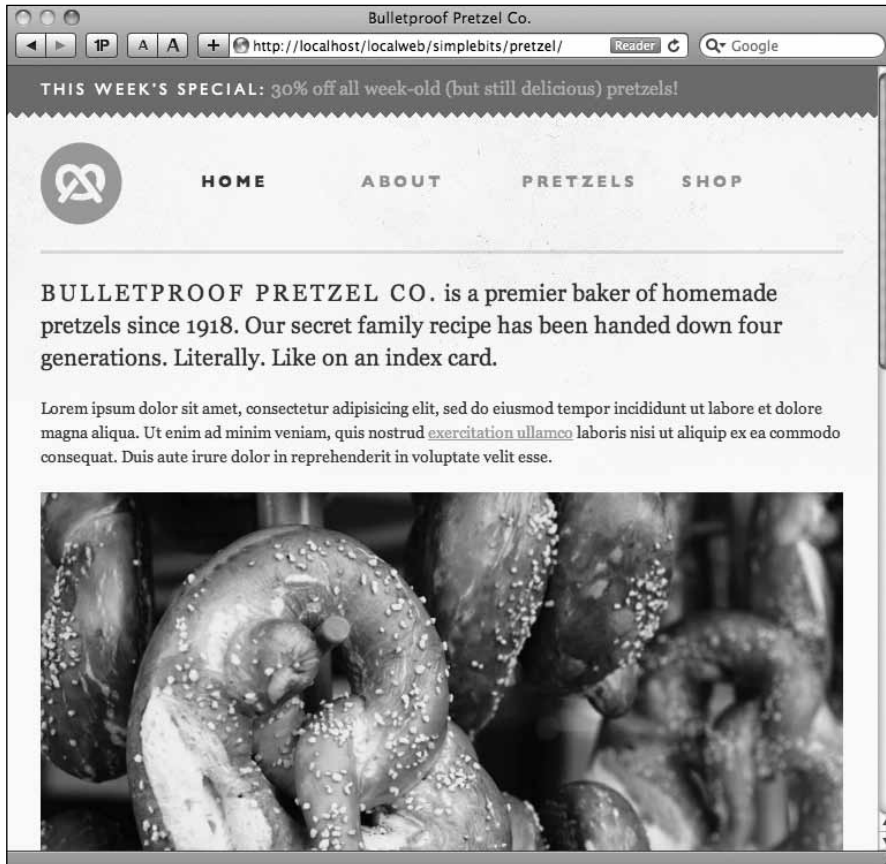


Рис. 9.23. При ширине менее 800px макет превращается в одноколоночный

Применим все правила:

/* адаптивность

```
----- */
@media screen and (max-width: 800px) {
  nav[role="navigation"] ul li {
    margin: 26px 0 0 0;
  }
  nav[role="navigation"] ul li a {
    font-size: .9em;
  }
  nav[role="navigation"] ul li a em {
    display: none;
  }
  #main,
  #sidebar {
```





```
float: none;
width: auto;
padding: 0;
font-size: .9em;
line-height: 1.5;
}
}
@media screen and (max-width: 550px) {
  #wrap {
    padding: 0 15px;
  }
  #special {
    padding-left: 15px;
    padding-right: 15px;
    font-size: 1.2em;
    line-height: 1.3;
  }
  #logo {
    float: none;
    margin: 0 auto 20px auto;
  }
  nav[role="navigation"] ul {
    text-align: center;
  }
  nav[role="navigation"] ul li {
    float: none;
    display: inline;
    width: auto;
    margin: 0 4px;
    font-size: .75em;
  }
  nav[role="navigation"] ul li a {
    float: none;
    display: inline;
    width: auto;
  }
  nav[role="navigation"] ul li a strong {
    display: inline;
  }
  #main h1 {
    font-size: 1.2em;
    line-height: 1.4;
  }
  footer[role="contentinfo"] p {
    margin-right: 20px;
    margin-left: 20px;
  }
}
```

Мы отменяем конкретные селекторы в основной таблице стилей, двигаясь сверху вниз. Так как в медиазапросах указано значение ширины менее 500 пикселей, предыдущий медиазапрос для 800 пикселей продолжает работать. Стили наследуются, поэтому нам не нужно их дублировать.

На рис. 9.24 показаны результаты применения новых правил для узкого поля просмотра (уже 550 пикселей). После назначения нескольких дополнительных стилей работать на телефоне или небольшом планшете стало намного удобнее.



Рис. 9.24. Узкий экран с шириной меньше 550 пикселей отобразит одноколоночный макет в удобочитаемом виде

Что мне больше всего нравится в адаптивном дизайне, так это то, что он похож на настоящий веб-дизайн — мы разрабатываем дизайн под соответствующий контекст, а не копируем печатные страницы на экран. Что же такое соответствующий контекст? Я этого не знаю. Именно поэтому комбинация пуленепробиваемости и адаптивности гарантирует наилучший результат в разных ситуациях.

Можно ли применять адаптивный дизайн к любому проекту? Возможно, нет. Однако в некоторых случаях он будет лучшим решением, так как устраняет необходимость в разработке отдельных версий сайтов для мобильных устройств или использовании нескольких разных макетов для одного и того же устройства. Адаптивный дизайн предполагает свободу и учет информации о том, как пользователи могут использовать контент сайта. Именно эту мысль я хочу донести до вас.

ПОДДЕРЖКА МЕДИАЗАПРОСОВ

Когда я писал эту книгу, медиазапросы стали поддерживать большинство браузеров (это и стало причиной появления адаптивного дизайна). Safari, Chrome, Firefox, Opera и IE9+ поддерживают медиазапросы. Для более старых версий IE я рекомендую использовать Respond.js (<http://github.com/scottjehl/Respond>) — удобную программку на JavaScript, написанную Скоттом Джейлем, которая позволяет использовать медиазапросы `min/max-width` в браузерах IE6-8.

ЗАКЛЮЧЕНИЕ

Цель этой главы — свести воедино все аспекты, о которых мы говорили в предыдущих главах. Надеюсь, что увидев, как описанные методики вместе работают на одной странице, вы поймете, что принцип пуленепробиваемости — это предвидение, планирование неизвестного, непрерывный процесс, который никогда нельзя завершить. Одного окончательного варианта решения не существует. Вам придется принимать множество решений и идти на компромиссы, которые, в конце концов, повысят целостность веб-сайта и станут гарантией того, что ваш сайт будет доступен самой широкой аудитории, сохранив свою привлекательность.

Если заняться пуленепробиваемостью сразу на начальном этапе проекта, расширение или адаптация дизайна становится намного проще. Помните, как легко мы добавили адаптивные медиазапросы в нашу резиновую верстку на пуленепробиваемой основе?

Инструменты (HTML и CSS) у нас в руках и готовы к работе. Постоянно появляются новые методики. Мы живем в удивительное время. Веб-дизайнеры могут

создавать красивые сайты, которые могут подстраиваться для отображения информации на большинстве устройств.

Я надеюсь, что вы воспользуетесь советами и методиками, о которых шла речь в этой книге, и разработаете собственные, еще более гибкие.

Спасибо!

Дэн Седерхольм
Пуленепробиваемый веб-дизайн.
Библиотека специалиста. 3-е изд.
Перевела с английского А. Струсевич

Заведующий редакцией
Руководитель проекта
Ведущий редактор
Литературный редактор
Художественный редактор
Корректоры
Верстка

А. Кривцов
А. Юрченко
Ю. Сергиенко
О. Некруткина
Л. Адуевская
Л. Казарина, В. Листова
Л. Родионова

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 02.05.12. Формат 70х100/16. Усл. п. л. 24,510. Тираж 2000. Заказ
Отпечатано с готовых диапозитивов в ИПК ООО «Ленинградское издательство».
194044, Санкт-Петербург, ул. Менделеевская, 9.