Дэн Седерхольм

при участии Итана Маркотта

# CSS ручной работы


New Riders

ПИТЕР®

Dan Cederholm

with Ethan Marcotte

# Handcrafted CSS

## More Bulletproof Web Design


New Riders

VOICES THAT MATTER™

Дэн Седерхольм
при участии Итана Маркотта

# CSS ручной работы


ПИТЕР®
Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2011

ББК 32.988.02-018
УДК 004.738.5
С28

**Д. Седерхольм, И. Маркотт**
С28 CSS ручной работы. Библиотека специалиста. — СПб.: Питер, 2011. — 240 с.: ил. — (Серия «Библиотека специалиста»).

ISBN 978-5-49807-749-9

Книга посвящена современным технологиям веб-дизайна на основе CSS — каскадных таблиц стилей, использующихся при создании большинства современных интернет-сайтов. Издание затрагивает такие вопросы, как использование в веб-дизайне новых свойств CSS3, гибкая работа с цветом при помощи RGBA, работа с «плавающими» элементами, использование «резиновой» верстки и гибких элементов дизайна, искусство работы с типографикой, jQuery, фоновыми элементами, а также массу других аспектов по совершенствованию веб-дизайна с помощью технологий CSS.

В дизайне нет мелочей; есть детали, которые оказываются наиболее важными. «CSS ручной работы» — это книга о таких деталях, которые отличают хороший веб-дизайн от первоклассного.

Автор книги — Дэн Седерхольм, талантливый веб-дизайнер, написавший несколько бестеллеров по искусству веб-дизайна, в том числе знаменитый «Пуленепробиваемый веб-дизайн».

ББК 32.988.02-018
УДК 004.738.5



ISBN 978-0321643380 (англ.)
ISBN 978-5-49807-749-9

# Содержание

# Введение

**Любой разумный дурак может делать вещи больше, сложнее и грубее. Требуется чуть-чуть гения, — и много смелости — чтобы действовать в противоположном направлении.**

Альберт Эйнштейн

Я вырос в Вермонте. Именно поэтому я восхищаюсь настоящим мастерством и ценю его. Я хорошо помню, как по утрам каждое воскресенье мы с мамой ходили на фермерский рынок, где все местные художники и народные умельцы продавали свои товары прямо с пикапов. Их качество можно было оценить по-настоящему: взять товар в руки и внимательно рассмотреть, попробовать.


Вермонт стал символом безупречного качества. В некотором смысле этот штат является самым настоящим брендом. Так, например, Пол Гринберг из газеты «Boston Globe» в статье «The Brand Called Vermont» (*http://boston.com/news/globe/ideas/articles/2003/10/12/the_brand_called_vermont/*) пишет следующее.

> *Любой продукт с отметкой «Made in Vermont» — будь то кленовый сироп с экстрактами трав, ананасовый джем с перцем или йогурт из молока буйволицы со вкусом чая — ценится на 10 процентов больше, чем то же самое, произведенное в любом другом месте.*

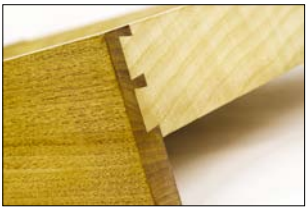
Почему так происходит? В моем воображении продукт, сделанный в Вермонте, вызывает образ старого человека с длинной бородой, который на вершине горы делает из клена обеденный стол. Это совсем не похоже на конвейер.

Так что мое представление о мастерстве выглядит так: если какая-то вещь сделана хорошо, то за ее созданием стоит

**ПРИМЕЧАНИЕ**

Так случилось, что Итан Маркотт (автор главы 6) тоже вырос в Вермонте. Ура «штату Зеленых гор»!

*человек* — пара рук, которая тщательно отбирала детали, чтобы в результате получилось мастерски выполненное, высококачественное изделие.



Выбор деталей не всегда очевиден. Возьмем какой-нибудь предмет мебели, сделанный искусным мастером: вы можете и не заметить его превосходных качеств, пока не начнете им пользоваться. А потом вы выдвинете ящик и увидите, к примеру, что какие-то его части соединены «ласточкиным хвостом».

И все это можно перенести на область веб-дизайна. Совсем не очевидные с первого взгляда мелочи могут отделять хороший веб-дизайн от первоклассного. И вы можете не оценить высокого качества сайта до тех пор, пока не начнете его использовать, заглядывая под капот, всячески проверяя его.

*CSS ручной работы* — это попытка рассказать о некоторых деталях, которые оказываются наиболее важными. Таким образом, мы продолжаем говорить о создании гибких, пуленепробиваемых, высокоэффективных и адаптируемых интерфейсов, которые формируют надежную систему взаимодействия с пользователем.

## О ЧЕМ ЭТА КНИГА

В каждой главе этой книги содержатся практические примеры, касающиеся трех аспектов мастерства (с точки зрения использования CSS): пуленепробиваемый дизайн, прогрессивное оформление и переоценка старых методов и самых лучших технических приемов.

### ПУЛЕНЕПРОБИВАЕМЫЙ ДИЗАЙН

Если вы знакомы с моей предыдущей книгой «*Пуленепробиваемый Web-дизайн*», то вы уже поняли, насколько полезно при создании дизайна помнить о гибкости — и насколько важно продумывать наихудшие сценарии.

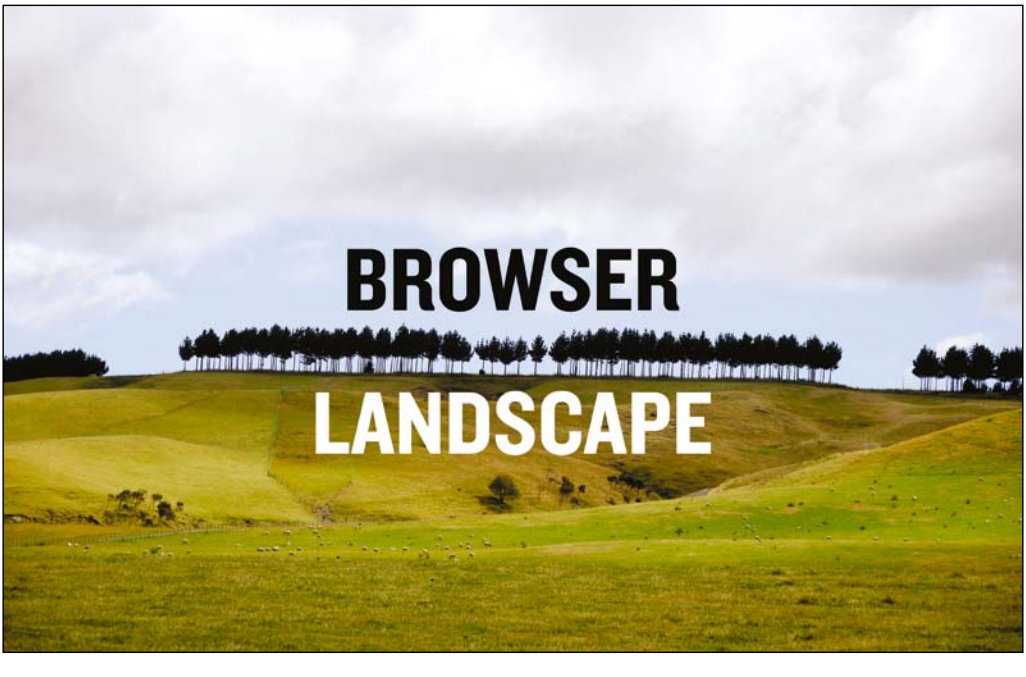
Чтобы с головой погрузиться в эту книгу, вам не обязательно иметь у себя дома «*Пуленепробиваемый Web-дизайн*». Но если вы ее читали, то заметите, что здесь в центре внимания остаются те же основные понятия — гибкость и адаптируемость; но при этом используются новые идеи и методы (некоторые из них не были доступны на момент написания книги «*Пуленепробиваемый Web-дизайн*»).

В каком-то смысле реализация дизайнерских проектов с использованием этих пуленепробиваемых идей является одним из аспектов мастерства. Вы определяете, насколько гибким является дизайн, или насколько он может адаптироваться к изменению объема контента или размера текста, — такого рода детали не всегда очевидны до того, как вы начнете использовать дизайн, редактировать его или же применять более суровые меры: повседневное использование.

В этой книге мы также будем стараться достичь пуленепробиваемости, а начнется все с напоминания о том, насколько важна гибкость, в главе 1.

## ПРОГРЕССИВНОЕ ОФОРМЛЕНИЕ

Сейчас отличное время для веб-дизайна! Ландшафт браузеров меняется очень быстро, и браузеры вводят новые и развивающиеся стандарты со стремительно возрастающей скоростью. Это значит, что можно экспериментировать с передовыми технологиями, а иногда даже использовать их, уже *сегодня*, пока они постепенно добавляются в последние версии браузеров.




Вы, вероятно, уже знакомы с термином «прогрессивное улучшение» — он используется, когда речь идет об изящном снижении качества дизайна, использующего JavaScript: автор должен

предусмотреть запасной вариант на случай, если, например, скрипты будут отключены или недоступны. Я буду использовать термин «прогрессивное *оформление*» только применительно к свойствам CSS и CSS3, которые на сегодняшний день работают в прогрессивных браузерах.

Существенная часть книги — это разговор о будущем CSS и о возможностях, с которыми можно экспериментировать и работать уже сейчас. Я постараюсь показать, что при использовании современных стилей качество оформления снижается красиво (в случае если они не поддерживаются); вы также узнаете, что сайты не должны выглядеть одинаково во всех браузерах.

### ПЕРЕОЦЕНКА СТАРЫХ МЕТОДОВ И САМЫХ ЛУЧШИХ ТЕХНИЧЕСКИХ ПРИЕМОВ

Сейчас, когда скорость внедрения новых стандартов стремительно возрастает, пришло время переоценить старые методы и решения — вот еще одна мысль, которая проходит сквозь идеи, рассматриваемые в данной книге.

Существуют ли новые, более простые и эффективные методы решения конкретной проблемы, которая раньше требовала хитрых приемов и «заплаток»? Важно иногда пересматривать технические приемы, которые раньше считались лучшими, чтобы можно было упростить и модернизировать код.

В этой книге вы найдете несколько примеров, где мы занимались переоценкой, пытаясь понять, существует ли лучший способ реализации, учитывая количество браузеров на данный момент и их возможности в поддержке CSS.


## Для кого эта книга?

Стопка книг по CSS, существующих на сегодняшний день, достигла внушительной высоты. Так нужна ли нам еще одна? Я надеюсь, что для большинства людей CSS- и веб-стандарты стали привычным делом. И там не нужны новые книги, в которых рассматриваются базовые вещи и объясняется, что CSS — это здорово. Вместо этого мы можем сразу заняться обсуждением технических моментов, используя те инструменты, которые находятся в нашем распоряжении сейчас. И именно это, как мне кажется, делает книгу по CSS интересной.

*CSS ручной работы* — это книга для дизайнера, который хочет расширить круг своих возможностей. Она научит вас добавлять к интерфейсам те мелочи, которые делают дизайн лучше. Кроме того, она поможет подготовиться к будущему. Так, знание свойств CSS3, которые уже сейчас внедряются браузерами, дает вам преимущество в области технологий, способных в будущем облегчить жизнь веб-дизайнеру.

## НЕКОТОРЫЕ ИСХОДНЫЕ ПРЕДПОЛОЖЕНИЯ

Относительно читателя я делаю несколько исходных предположений. Я буду считать, что вы по крайней мере знакомы с CSS и HTML, а также с понятием семантической разметки и преимуществами использования веб-стандартов. Как я уже говорил, мы сразу займемся делом: будем рассматривать типичные ситуации и разбирать по частям наш учебный пример.

Нужно ли вам быть экспертом в CSS? Ни в коем случае. Но знание основных понятий, безусловно, окажется полезным.

## HTML РУЧНОЙ РАБОТЫ?

Помимо того, что в мире CSS происходит много всего (связанного с CSS3 и стремительным внедрением этих развивающихся стандартов), в мире HTML тоже многое меняется. HTML5 — следующая версия языка Сети — набирает обороты. И хотя эта спецификация в ближайшее время не будет закончена, браузеры уже сейчас внедряют некоторые возможности HTML5.

**ПРИМЕЧАНИЕ**

Объявление W3C о прекращении работы над XHTML2 в конце 2009 года можно считать подтверждением того, что будущее разметки — за HTML5 (*http://www.w3.org/News/2009#item119*).

Конечно, многое еще необходимо доработать, и значительная часть HTML5 на момент написания этой книги еще не готова к тому, чтобы ее использовать. Так что для примеров этой книги мы используем тип XHTML 1.0 Transitional. Другими словами, для написания кода нашего учебного примера используется шаблон, который выглядит приблизительно так.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
     Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
     transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
   <meta http-equiv="content-type" content="text/
     html; charset=utf-8" />
   <title>Tugboat Coffee Company</title>
   ...
</head>
<body>
   ... сюда добавляется код примера ...
</body>
</html>
```

Сейчас наиболее удобными для меня являются правила *синтаксиса* первой версии XHTML: теги и атрибуты пишутся строчными буквами, все элементы должны быть закрыты, значения атрибутов нужно заключать в кавычки и т. д. Независимо от типа, который я буду использовать в будущем, я, скорее всего, и дальше буду придерживаться этих правил — это поможет мне сохранить документы чистыми, понятными и правильно построенными.

**СОВЕТ**

Более интересные мысли о месте XHTML в будущем, когда в центре внимания окажется HTML5, можно прочитать на *http://adactio.com/journal/1595* и *http://www.zeldman.com/2009/07/07/in-defense-of-web-developers/*.

Я также убежден, что когда HTML5 достигнет переломного момента в вопросе браузерной поддержки, не так уж трудно будет изменить существующую разметку с учетом новых правил.

Кстати, а ведь это может стать темой для продолжения книги! :)

## ОБНУЛЕНИЕ СТИЛЕЙ

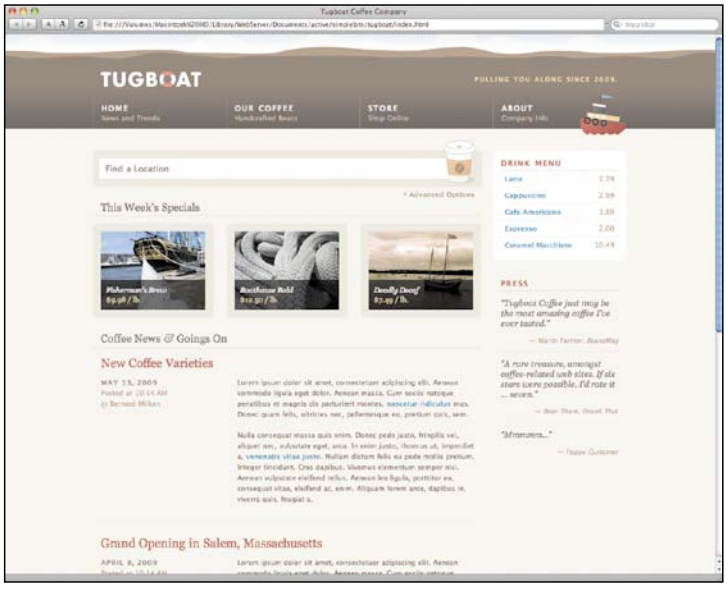
Следует также упомянуть, что примеры этой книги предполагают использование *таблицы стилей reset*. Файл *reset.css* импортируется перед остальными таблицами стилей, обнуляя оформление, которое большинство браузеров использует по умолчанию. Это избавляет нас от необходимости повторять часто используемые правила (такие как `margin: 0;` `padding: 0;`) в большом количестве объявлений. Кроме того, таким образом вы создаете прочный фундамент, к которому вы будете добавлять свои собственные стили.

Более подробно о тонкостях, связанных с использованием *reset.css*, я расскажу в главе 5. А пока я просто хочу, чтобы вы знали, что обнуление стилей имеет место для всех примеров данной книги.

## Наш учебный пример

На протяжении всей этой книги мы будем использовать шаблон, который я разработал для кофейной компании Tugboat (вымышленной). Он ни в коем случае претендует на то, чтобы считаться лучшим интерфейсом из когда-либо созданных. Скорее он построен так, чтобы его можно было использовать в качестве примера к каждой главе, и чтобы при этом он был логичным.

Загрузить файлы шаблона и код примера можно на сайте книги: *http: //handcraftedcss.com*, а также с сайта издательства на странице, посвященной этой книге.




Так что налейте себе чашечку хорошего кофе (или вашего любимого напитка), и давайте приступим к созданию пикселов и текста своими руками.

# Глава 1

## Всегда задавайтесь вопросом: «А что, если...?»

**Вы можете разодрать стихотворение на части с целью посмотреть, что именно делает его живым… Вы вернетесь — неся в себе таинство быть растроганным, взволнованным — словами. Наивысшее мастерство всегда оставляет в стихах лазейки и бреши — с тем, чтобы нечто, чего в стихотворении не было, могло ползать, красться, вспыхивать или греметь.**

Дилан Томас, «Поэтический манифест»[1], 1961

[1] Перевод Е. Кузьминой, *http://elenakuzmina.blogspot.com*.

Поэт Дилан Томас жил и работал в Уэльсе в середине двадцатого века, но его размышления о мастерстве можно отнести практически к чему угодно, даже к нашей удивительной и непростой профессии веб-дизайнера. Оставлять возможность для осуществления того, что не планировалось заранее, — одна из важнейших составных частей понятия «пуленепробиваемости». В этой главе мы снова хотим привлечь внимание к этому вопросу, рассмотрев простой список ссылок, который используется в правой части нашего примера.

Зачастую решения, которые мы принимаем как дизайнеры и разработчики, направлены на сохранение целостности интерфейса. Но что, если в результате на странице нужно будет уместить больше (или меньше) контента, чем планировалось? Что, если понадобится уменьшить или увеличить размер шрифта? Что, если вместо одного абзаца потребуется вставить два? И что если, один из этих абзацев будет переведен, к примеру, на немецкий?

*Гибкость* в веб-дизайне — признак настоящего мастерства. Умение работать в редакторе изображений уже само по себе является достижением, однако именно обеспечение возможностей для изменения границ в разметке страницы отличает первоклассный веб-дизайн от просто хорошего. Гибкость — это когда мы включаем Сеть *как среду* в процесс дизайна и не обращаем внимания на пиксельную точность.

Давайте обратимся к примеру, демонстрирующему «пуленепробиваемый» дизайн в шаблоне Tugboat, и дадим волю своему «внутреннему Дилану»: сделаем так, чтобы то, что может появиться в будущем, могло «ползать, красться, вспыхивать и греметь».

## Простой список ссылок

Посмотрев на шаблон Tugboat (рис. 1.1), обратим внимание на модуль Drink Menu справа вверху (рис. 1.2). Это обычная дизайнерская задача: вертикальный список ссылок, активируемых щелчком мыши, где каждая ссылка снабжена дополнительной информацией (в данном случае это цена напитка), выровненной по правому краю. Каждая строка отделена от следующей едва заметной горизонтальной линией.

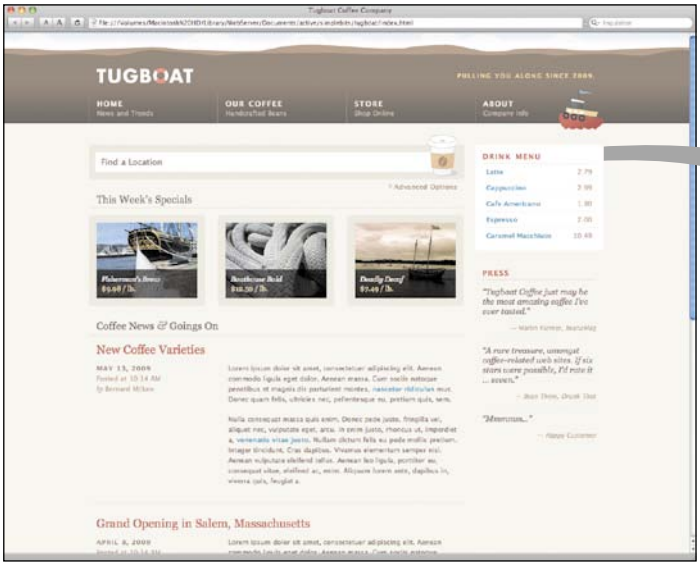




**Рис. 1.2**

**Рис. 1.1.** Кофейная компания Tugboat: вымышленный пример, который будет использоваться в этой книге

Десять долларов за Caramel Macchiato! Знаю, знаю… но когда эта книга появится на прилавках магазинов, вполне возможно, что цена будет именно такой.

## Блоковые ссылки

В первую очередь хочется сделать так, чтобы вся строка (текст ссылки *и* цена, а не только название напитка) реагировала на щелчок мыши. Благодаря расширению активной области ссылки дизайн становится в большей степени ориентированным на пользователя. Если при этом использовать hover-эффект, то читатель сможет выбрать ссылку, не заслоняя текст курсором мыши.

Расширяя активную область ссылки, мы на самом деле следуем закону Фиттса. Американский психолог Пол Фиттс известен благодаря выдвинутому им принципу, который гласит: время достижения цели есть функция, зависящая от расстояния до цели и размера цели. То есть чем больше цель, тем быстрее и проще до нее добраться. Это достаточно разумно, особенно когда речь идет о дизайне интерфейсов. А теперь давайте обратимся к коду, который потребуется для осуществления этих идей.


**Рис. 1.3.** Активной является вся строка, а не только синий текст ссылки

**СОВЕТ**

Более подробно о применении закона Фиттса к гиперссылкам можно прочитать на сайтах *http://www.uie.com/brainsparks/2008/02/28/usability-tools-podcast-applying-fitts-law/*, *http://www.1976design.com/blog/archive/2004/09/07/link-presentation-fitts-law/* и *http://www.mezzoblue.com/archives/2004/08/19/fitts_law/*.

## РАЗМЕТКА

В нашем случае мы можем использовать простой неупорядоченный список, элементами которого будут строки: название напитка и цена. При этом цену лучше выделить в отдельный элемент, чтобы в дальнейшем можно было выровнять ее по правому краю блока.

```
<ul class="lst">
   <li><a href="/drinks/latte">Latte <em>2.79
     </em></a></li>
   <li><a href="/drinks/capp">Cappucino
     <em>2.99</em></a></li>
   <li><a href="/drinks/amer">Cafe Americano
     <em>1.80</em></a></li>
   <li><a href="/drinks/espr">Espresso <em>2.00
     </em></a></li>
   <li><a href="/drinks/carm">Carmel Macchiato
     <em>10.49</em></a></li>
</ul>
```

Вы, должно быть, заметили, что цена в нашем примере обрамляется тегами `<em>`, однако подойдет и любой другой строковый элемент (например, `<span>`, `<strong>` и др.).

**ПРИМЕЧАНИЕ**

Здесь обязательно использовать именно строковый элемент (например, `<span>`, `<strong>` и т. д.), а не блоковый (`<div>`, `<h2>` и т. д.), так как мы находимся внутри элемента `<a>`, который сам является строковым. Иными словами, нельзя размещать блоковые элементы внутри строковых.

## ЗАДАЕМ СТИЛЬ ССЫЛКИ

Сначала создадим глобальный шаблонный стиль для ссылок: по умолчанию будем использовать синий полужирный шрифт без подчеркивания.


**Рис. 1.4**

```
a:link {
   font-weight: bold;
   text-decoration: none;
   color: #3792b3;
   }
```

Далее, следуя закону Фиттса и используя CSS, сделаем ссылки внутри элементов списка *блоковыми*, во всю ширину списка. Также зададим отступ и нижнюю рамку.

```
ul.lst li a {
   display: block;
   padding: 7px;
   border-bottom: 1px solid #f3f2e8;
   }
```

## ЗАДАЕМ СТИЛЬ ЦЕНЫ

Далее необходимо добавить стиль для цены. Так как большинство браузеров отображает текст внутри тега `<em>` курсивом, специально зададим прямой стиль шрифта. Также присвоим параметру `font-weight` значение `normal` и изменим цвет шрифта (ведь по умолчанию для ссылок используется синий полужирный шрифт).


**Рис. 1.5**

```
ul.lst li em {
   font-style: normal;
   font-weight: normal;
   color: #9c836e;
   }
```

В результате цена будет располагаться сразу после названия напитка (рис. 1.5). Благодаря этому мы приблизились к цели еще на один шаг.

## РАСПОЛОЖЕНИЕ ЦЕНЫ

Теперь мы можем задать расположение цены у правого края рамки. Здесь мы будем использовать *абсолютное позиционирование*, задавая координаты относительно строки, в которой находится цена. Для этого потребуется сначала в объявление стиля ссылки добавить `position: relative;`. После этого мы сможем задать расположение цены внутри ссылки: 7 пикселов от правого верхнего угла (что равняется отступу, который мы задали для ссылок).

```
ul.lst li a {
   position: relative;
   display: block;
   padding: 7px;
   border-bottom: 1px solid #f3f2e8;
   }

ul.lst li em {
   position: absolute;
   top: 7px;
   right: 7px;
   font-style: normal;
   font-weight: normal;
   color: #9c836e;
   }
```


**Рис. 1.6**

В результате мы получим список (рис. 1.6), в котором цена расположена у правого края рамки, а шрифт цены отличается от шрифта, использующегося для названия напитка. При этом вся строка является гиперссылкой, активируемой щелчком мыши. Отлично! Но теперь самое время поговорить о том, какие проблемы могут возникнуть при такой реализации простого списка.

## Незапланированное наложение

Давайте вспомним, что старик Дилан Томас говорил о лазейках и брешах, через которые может проникать непредвиденное. Повнимательнее посмотрев на наш пример и его реализацию, мы сразу же обнаружим, какую опасность представляет собой использование абсолютного позиционирования при изменении размера контента.


**Рис. 1.7**

Наш список (с теми названиями напитков и ценами, которые в нем сейчас используются) выглядит очень здорово. Но что произойдет, если название напитка будет длиннее?

Если добавить в список напиток с длинным названием (например, мой любимый «Half-caf, non-fat, triple-shot latte (with a twist of lemon)»), мы с ужасом обнаружим, что название и цена накладываются друг на друга (рис. 1.7).

При использовании абсолютного позиционирования мы отказываемся от нормального упорядочивания элементов в документе, и если мы не будем аккуратны, могут произойти

неприятные ситуации (в частности, когда наложение затруднит чтение). Бывают случаи, когда наложение действительно необходимо, и тогда абсолютное позиционирование — это то, что нужно. Но в нашем примере мы имеем дело с двумя различными частями гипертекста, и мы уж никак не заинтересованы в коллизиях (слышится грустное глиссандо тромбона).

Но ведь мы считаем себя мастерами веб-дизайна! А настоящий мастер умеет *смотреть вперед*, постоянно задаваясь вопросами вроде «Что, если название напитка будет таким длинным, что для него потребуется две строки?», и тем самым значительно улучшая гибкость и целостность проекта. А это и есть первоклассный веб-дизайн.

## Что, если будет изменен размер текста?

Стоит задать себе еще один вопрос: что произойдет, если читатель изменит размер текста в браузере? Даже при сравнительно коротких названиях напитков могут возникнуть проблемы, если увеличить размер шрифта на пару пунктов.

На рисунке 1.8 показан еще один вариант наложения. Название «Caramel Machiato», которое раньше было достаточно коротким и не доставляло проблем, в результате изменения размера текста стало частично перекрывать цену.

При хорошем веб-дизайне учитываются и такие ситуации. При реализации проекта важно помнить, что контент может сдвигаться во время полета. Иными словами, объем, размер и расположение гипертекста могут меняться, элементы — перегруппировываться и становиться больше или меньше, чем планировалось. Но это не страшно. Предугадав эти изменения, можно позаботиться о том, чтобы они не разрушили ваш проект. Это мы и называем «пуленепробиваемым дизайном», и именно это является отличительной чертой настоящего мастерства в Сети.


**Рис. 1.8.** Список напитков после увеличения размера текста на два деления

Это достаточно тонкий момент, и он не относится к разряду очевидных. Он не привлекает к себе внимания, но является ключевым в веб-дизайне. Так что давайте теперь позаботимся о *пуленепробиваемости* нашего примера со списком, обеспечив возможность изменения размера контента.

## Угадать размер контента

Один из способов исправить ситуацию — увеличить для ссылок отступ справа на величину, достаточную для устранения наложения.

```
ul.lst li a {
   position: relative;
   display: block;
   padding: 7px 50px 7px 7px;
   border-bottom: 1px solid #f3f2e8;
   }
```

Увеличение правого отступа до 50 пикселов (при изначальном отступе в 7 пикселов) дает результат, показанный на рисунке 1.9, где перенос названия на новую строку препятствует его наложению на цену. При этом для получения значения `50px` пришлось перебирать различные варианты до тех пор, пока текст названия не перестал накладываться на цену. А это значит, что теперь ширина цены не должна быть больше 50 пикселов.


**Рис. 1.9**

Проблема этого подхода состоит в том, что мы гадаем, какой может быть длина контента. Но даже если сейчас отступа в 50 пикселов может быть достаточно, то, как только цены на кофе превысят отметку в 100 долларов (а это когда-нибудь произойдет), мы снова получим наложение. Кроме того, при увеличении размера текста отступ в 50 пикселов потеряет смысл, поскольку ширина цены легко может превысить это значение.

**СОВЕТ**

Для задания здесь отступов лучше использовать единицы `em`, поскольку тогда ширина отступа будет зависеть от размера текста (например, `padding: 7px 4em 7px 7px;`).

## Ситуация, в которой стоит использовать абсолютное позиционирование

Существует ситуация, в которой абсолютное позиционирование является оптимальным вариантом. Представим, что вместо цены, представленной в текстовом виде, мы хотим поместить какое-нибудь изображение. Так как изображение имеет ограниченные размеры, мы можем использовать его максимальную ширину в качестве правого отступа для названия напитка.

На рисунке 1.10 изображен наш список, в котором вместо цены используется изображение в формате GIF, обозначающее оценку (от 1 до 4 звезд). Чтобы задать расположение, заключим изображение в тег `<em>` (так же, как мы делали это для цены).


**Рис. 1.10**

```
<ul class="lst">
   <li><a href="/drinks/latte">Latte <em>
     <img src="/img/stars-3.gif" alt="3/4 stars"
     /></em></a></li>
   <li><a href="/drinks/capp">Cappuccino <em><img
     src="/img/stars-2.gif" alt="2/4 stars" /></
     em></a></li>
   <li><a href="/drinks/amer">Cafe Americano
     <em><img src="/img/stars-4.gif" alt="4/4
     stars" /></em></a></li>
   <li><a href="/drinks/espr">Espresso <em>
     <img src="/img/stars-1.gif" alt="1/4 stars"
     /></em></a></li>
   <li><a href="/drinks/trip">Half-caf, non-fat,
     triple-shot latte (with a twist of lemon)
     <em><img src="/img/stars-3.gif" alt="3/4
     stars" /></em></a></li>
</ul>
```

Ширина каждого изображения — 57 пикселов. Зная это число, мы можем смело использовать его, чтобы задать правый отступ для ссылок (57 пикселов + еще чуть-чуть для отбивки), и будем уверены в том, что длина отступа останется неизменной.

```
ul.lst li a {
   position: relative;
   display: block;
   padding: 7px 60px 7px 7px;
   border-bottom: 1px solid #f3f2e8;
   }
```

Так что для элементов с фиксированной шириной (например, изображений) или в случаях, когда ограничен размер контента, абсолютное позиционирование значительно облегчает процесс реализации. Но если элементы могут различаться по длине или объему, лучшим решением остается использование плавающих элементов. Этим мы и займемся далее.

## Улучшаем гибкость при помощи свойства Float

Давайте для размещения цены у правого края будем использовать не абсолютное позиционирование, а атрибут `float`. Это немного повысит гибкость дизайна и поможет избежать

наложений, которые могут возникнуть при применении абсолютного позиционирования.

## НОВЫЙ ПОРЯДОК РАЗМЕТКИ

Мы немного изменим разметку, чтобы проще было использовать плавающие элементы. Давайте перенесем элемент в теге `<em>` (цену) так, чтобы он располагался *перед* названием напитка.

```
<ul class="lst">
   <li><a href="/drinks/latte"><em>2.79</em>
     Latte</a></li>
   <li><a href="/drinks/capp"><em>2.99</em>
     Cappuccino</a></li>
   <li><a href="/drinks/amer"><em>1.80</em> Café
     Americano</a></li>
   <li><a href="/drinks/espr"><em>2.00</em>
     Espresso</a></li>
   <li><a href="/drinks/trip"><em>10.49</em>
     Half-caf, non-fat, triple-shot latte (with a
     twist of lemon)</a></li>
</ul>
```

**ПРИМЕЧАНИЕ**

Вы можете со мной поспорить, утверждая, что это не оптимальный порядок и что цена должна располагаться после названия напитка. Или можете сказать, что цена должна быть *перед* названием напитка, даже если мы отображаем их в противоположном порядке. Не важно. Это одна из причин, по которой я предпочитаю использовать в таких случаях теги `<em>` или `<strong>`. Если программа, читающая текст с экрана, прочитает это вслух, она будет отделять фрагменты текста друг от друга с помощью громкости, темпа и т. д. Если вам не нравится порядок, который я здесь использую, не пугайтесь. Совсем скоро мы поговорим о том, как его изменить.

## КАК СДЕЛАТЬ ЦЕНУ ПЛАВАЮЩИМ ЭЛЕМЕНТОМ

Теперь, когда мы изменили порядок, разместив цену перед названием напитка, мы можем сделать так, чтобы цена была прижата к правому краю, но при этом не использовалось абсолютное позиционирование (нам, следовательно, необходимо будет удалить то, что к нему относится).

```
ul.lst li a {
   position: relative;
```


Рис. 1.11

```
    display: block;
    padding: 7px;
    border-bottom: 1px solid #f3f2e8;
    }
ul.lst li em {
    position: absolute;
    top: 7px;
    right: 7px;
    float: right;
    font-style: normal;
    font-weight: normal;
    color: #9c836e;
    }
```

Довольно-таки просто, не правда ли? Всего лишь применение свойства `float` к цене задает ее расположение вдоль правого края, как мы и хотели. А кроме этого, цена и название напитка будут знать о существовании друг друга, так что наложения никогда не произойдет (рис. 1.11).


Рис. 1.12

Теперь можно спать спокойно, зная, что в этот небольшой список поместится сколь угодно длинное название напитка, которое только может прийти в голову хитроумным баристам.

### ТЕСТ НА ИЗМЕНЕНИЕ РАЗМЕРА ТЕКСТА

Можно провести быструю проверку на целостность дизайна, увеличив размер текста на несколько пунктов. Как показывает рисунок 1.12, если сделать цену плавающим элементом, увеличение размера текста не повлияет на удобство чтения списка.

## Более оптимальный порядок разметки

Возможно, вам не понравилось, что в разметке цена напитка предшествует его названию. Тогда можно использовать плавающие элементы, располагая их у противоположных сторон рамки (метод «opposing floats»). Это значит, что мы можем закрепить название у левого края, а цену — у правого, и тогда порядок разметки будет более оптимальным. Для этого нам всего лишь потребуется выделить название в отдельный элемент, чтобы к нему можно было применить свойство `float`.

Для нашего примера мы выберем замечательно универсальный элемент , хотя подойдет и любой другой строковый элемент (например, ).

```
<ul class="lst">
   <li><a href="/drinks/latte"><span>Latte</span>
     <em>2.79</em></a></li>
   <li><a href="/drinks/capp"><span>Cappuccino</
     span> <em>2.99</em></a></li>
   <li><a href="/drinks/amer"><span>Cafe
     Americano</span> <em>1.80</em></a></li>
   <li><a href="/drinks/espr"><span>Espresso</
     span> <em>2.00</em></a></li>
   <li><a href="/drinks/trip"><span>Half-caf,
     non-fat, triple-shot
latte (with a twist of lemon)</span> <em>10.49</
     em></a></li>
</ul>
```

Когда и название, и цена выделены в отдельные элементы, можно сделать эти элементы противоположно плавающими, и в результате список будет выровнен так, как мы хотим.

```
ul.lst li a {
   display: block;
   padding: 7px;
   border-bottom: 1px solid #f3f2e8;
   overflow: hidden;
   }

ul.lst li span {
   float: left;
   }

ul.lst li em {
   float: right;
   font-style: normal;
   font-weight: normal;
   color: #9c836e;
   }
```

Результат изображен на рисунке 1.13. Как вы могли заметить, мы добавили строку `overflow: hidden;` в объявление элемента <a>, в который заключены плавающие элементы  и . Это один из методов самоочистки для плавающих элементов внутри содержащего их элемента. Каждая строка

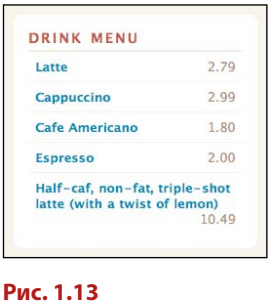



**Рис. 1.13**

останется независимой, так что плавающие элементы одной строки не будут мешать другим соседним элементам. Подробнее о плавающих блоках рассказывается в главе 6.

Вы также, наверное, заметили, что при такой реализации наложения не происходит, а в последнем ряду списка цена оказывается под названием напитка. Если для плавающего элемента не задана фиксированная ширина, он будет занимать столько места, сколько ему нужно. Поскольку данное название длиннее других, оно занимает всю ширину рамки, оттесняя цену вниз.

## ЗАДАЕМ ШИРИНУ ПЛАВАЮЩИХ ЭЛЕМЕНТОВ

Чтобы решить этот вопрос, нам необходимо задать ширину названия напитка. И здесь нам опять придется угадывать длину контента, как и в случае с абсолютным позиционированием. Однако здесь есть одно существенное отличие: даже если мы ошибемся с выбором правильного значения, мы никогда не столкнемся с проблемой наложения, делающей текст нечитаемым. Просто цена будет располагаться ниже названия. Хотя визуально это будет выглядеть не так, как нам бы хотелось, все же важным преимуществом здесь является уверенность в том, что читабельность текста не будет нарушена.

Можно задать ширину названия, указав его долю в процентах. Для этого добавим строку `width: 75%;` в объявление стиля элемента `<span>`, закрепленного у левого края. Так как мы указываем ширину в процентах, уменьшается вероятность возникновения коллизии в случае, если изменится размер рамки или даже разметка страницы. А под *коллизией* мы понимаем смещение цены под название напитка без наложения (что произошло бы в случае абсолютного позиционирования).

```
ul.lst li a {
   display: block;
   padding: 7px;
   border-bottom: 1px solid #f3f2e8;
   overflow: hidden;
   }
ul.lst li span {
   float: left;
   width: 75%;
   }
ul.lst li em {
   float: right;
```

```
    font-style: normal;
    font-weight: normal;
    color: #9c836e;
    }
```

Как показано на рисунке 1.14, для длинного названия напитка используется такое разбиение на строки, поскольку на него отводится 75% ширины рамки. Таким образом, мы получили гибкую систему, позволяющую отображать названия и цены напитков.


**Рис. 1.14**

## А КАК НАСЧЕТ ТАБЛИЦЫ?

Возможно, вы скажете что-то вроде: «Дэн, а почему бы не использовать для списка тег `<table>`?» И действительно, нетрудно найти аргумент в вашу пользу. При том, что тег `<table>` настоятельно не рекомендуется использовать для *верстки*, нет причин отказываться от него, если мы имеем дело с тем, для чего он исконно предназначен: *табличными данными*. И вы можете просто утверждать, что наш список из названий напитков и их цен есть не что иное, как табличные данные, и работать с ним нужно именно так.

Я всегда говорю, что в веб-дизайне существует примерно 3296 способов достижения одной и той же цели. И тот небольшой пример, который обсуждается в этой главе, не является исключением.

Вот как можно оформить данные из нашего примера в виде таблицы (названия напитков будут заголовками, а цены — ячейками таблицы).

```
<table>
   <tr>
      <th><a href="/drinks/latte">Latte</a></th>
      <td>2.79</td>
   </tr>
   <tr>
      <th><a href="/drinks/capp">Cappuccino</a>
      </th>
      <td>2.99</td>
   </tr>
   <tr>
      <th><a href="/drinks/amer">Cafe Americano
      </a></th>
```

```
      <td>1.80</td>
   </tr>
   <tr>
      <th><a href="/drinks/espr">Espresso</a>
      </th>
      <td>2.00</td>
   </tr>
   <tr>
      <th><a href="/drinks/trip">Half-caf,
      non-fat,
triple-shot latte (with a twist of lemon)</a>
     </th>
      <td>10.49</td>
   </tr>
</table>
```

## ЗАКЛЮЧЕНИЕ БЛОКОВЫХ ЭЛЕМЕНТОВ В ТЕГ ССЫЛКИ

Я решил использовать в качестве ссылки только название напитка, однако здесь уже кроется проблема: невозможно сделать блоковый элемент гиперссылкой, и поэтому нет простого способа добиться того, чтобы вся строка реагировала на щелчок мыши. Иными словами, если бы нам разрешили заключить каждый из элементов `<td>` и `<th>` в тег `<a>`, мы могли бы достичь нужного результата. Но сейчас это невозможно.

Я все же приведу пример того, что я имею в виду (теоретически) под заключением в тег ссылки блоковых элементов `<td>` и `<th>`, хотя это в настоящее время не поддерживается/недопустимо.

```
<table>
   <tr>
      <a href="/drinks/latte">
         <th>Latte</th>
         <td>2.79</td>
      </a>
   </tr>
   <tr>
      <a href="/drinks/capp">
         <th>Cappuccino</th>
         <td>2.99</td>
      </a>
```

```
    </tr>
    <tr>
      <a href="/drinks/amer">
        <th>Cafe Americano</th>
        <td>1.80</td>
      </a>
    </tr>
    <tr>
      <a href="/drinks/espr">
        <th>Espresso</th>
        <td>2.00</td>
      </a>
    </tr>
    <tr>
      <a href="/drinks/trip">
        <th>Half-caf, non-fat, triple-shot latte
        (with a twist of lemon) </th>
        <td>10.49</td>
      </a>
    </tr>
</table>
```

**ПРИМЕЧАНИЕ**

Возможно, в HTML5 будет разрешено заключать в тег `<a>` несколько элементов и блоковые элементы (*http://www.brucelawson.co.uk/2008/anyelement-linking-in-html-5/*). Вот здорово будет, не правда ли?

Так что для удобства пользователя, и учитывая, что это всего лишь очень простой список с двумя разновидностями данных, мы все же останемся верными неупорядоченному списку с активными строками.

## Добавим визуализацию данных

А теперь, веселья ради, почему бы нам не визуализировать какие-нибудь данные в нашем списке? Раз мы уже имеем дело с вертикальным списком, содержащим активные строки данных, не так уж сложно будет расположить на заднем плане гистограмму, чтобы подчеркнуть разницу между элементами списка.

В нашем примере мы будем демонстрировать различие в цене, хотя нетрудно придумать примеры использования гистограмм для других вертикальных списков — например, для списка категорий статей с указанием количества статей в каждой категории. В этом случае с помощью фоновой гистограммы можно показать различие в количестве статей, чтобы читатель смог быстро сравнить эти данные, просмотрев список.


**Рис. 1.15**

На рисунке 1.15 показано, как можно использовать фоновую гистограмму для списка напитков компании Tugboat: ширина полоски соответствует цене напитка. Этот прием описан в статье Вилсона Майнера (Wilson Miner), опубликованной на A List Apart («Accessible Data Visualization with Web Standards», *http://www.alistapart.com/articles/accessibledatavisualization*). Там автор объясняет, как сделать поистине роскошную визуализацию, используя лучшие приемы CSS, и среди множества превосходных примеров там есть и пример с гистограммой.

Теперь давайте быстро сделаем все, что нужно, чтобы добавить этот полезный «наворот» к нашему списку напитков.

## ДОБАВЛЯЕМ ДАННЫЕ В РАЗМЕТКУ

Нам придется немного изменить разметку, чтобы вставить гистограмму. По сути, мы хотим добавить значение в процентах, соответствующее ширине каждой полоски. Будем сравнивать это значение с наибольшей ценой. Иными словами, наибольшую цену из списка примем за 100%, а значения для остальных напитков будут вычисляться с учетом цены самого дорогого напитка.

**ПРИМЕЧАНИЕ**

Для этого примера я подобрал приблизительные значения в процентах. На самом же деле вычислением этих значений должна заниматься серверная часть приложения.

```
<ul class="lst">
   <li><a href="/drinks/latte"><em>2.79</em>
      Latte</a> <span>60%</span></li>
   <li><a href="/drinks/capp"><em>2.99</em>
      Cappuccino</a> <span>68%</span></li>
   <li><a href="/drinks/amer"><em>1.80</em> Cafe
      Americano</a> <span>35%</span></li>
   <li><a href="/drinks/espr"><em>2.00</em>
      Espresso</a> <span>50%</span></li>
   <li><a href="/drinks/carm"><em>10.49</em>
      Carmel Macchiato</a> <span>100%</span></li>
</ul>
```

Теперь в нашей разметке есть дополнительная информация: доля в процентах относительно наибольшей цены, принятой за 100%. Для этих значений я решил использовать универсальный элемент `<span>`, поместив его вне элемента `<a>`.

## ПРИМЕНЯЕМ БАЗОВЫЕ СТИЛИ

Итак, разметка готова. Теперь воспользуемся нашими базовыми стилями, чтобы заработали блоковые ссылки, а цена снова располагалась у правого края.

```
ul.lst li {
   margin: 0 0 2px 0;
   }
ul.lst li a {
   display: block;
   padding: 7px;
   }
ul.lst li em {
   float: right;
   font-style: normal;
   font-weight: normal;
   color: #9c836e;
   }
```


**Рис. 1.16**

Результат показан на рисунке 1.16. Как видите, значение в процентах расположилось под названием и ценой.

Но вообще-то мы хотим получить гистограмму, отражающую это процентное значение в качестве фона для активной строки, а вовсе не значение в текстовом виде. Так что самое время для какого-нибудь хитроумного трюка.

## КАК СКРЫТЬ ЗНАЧЕНИЕ В ПРОЦЕНТАХ И СОЗДАТЬ ПОЛОСКУ

Далее мы хотим сделать так, чтобы процентное значение не отображалось в текстовом виде. Поэтому зададим для элемента `<span>` длину, ширину и фон, а затем разместим его позади текста ссылки. Считая, что эти данные в разметке указаны правильно, спрячем их и добавим в гистограмму, используя CSS.

Наш первый шаг — задать ширину каждого элемента `<span>`. Мы могли бы создать свое имя класса для каждого нужного значения, однако проще всего добавить ширину в разметку, используя строковый стиль. Как правило, я не одобряю добавление атрибутов стилей в разметку. Но обычно вычисление ширины относится к серверной части, и поэтому в нашем случае мы выносим это значение в разметку, просто чтобы легче было работать с данными. Поэтому здесь использование строкового стиля не доставит неприятностей. Конечно, вы можете сказать, что задание ширины в разметке на самом деле больше соответствует правилу «контент — отдельно, оформление — отдельно». Как будто на сервере создается картинка этим кодом.

```
<ul class="lst">
   <li><a href="/drinks/latte"><em>2.79</em>
      Latte</a> <span style="width: 60%;">60%
      </span></li>
   <li><a href="/drinks/capp"><em>2.99
      </em> Cappuccino</a><span style="width:
      68%;">68%</span></li>
   <li><a href="/drinks/amer"><em>1.80</em> Cafe
      Americano</a> <span style="width: 35%;">35%
      </span></li>
   <li><a href="/drinks/espr"><em>2.00</em>
      Espresso</a> <span style="width: 50%;">50%
      </span></li>
   <li><a href="/drinks/carm"><em>10.49
      </em> Carmel Macchiato</a> <span
      style="width: 100%;">100%</span></li>
</ul>
```

Теперь давайте объявим стили, ведь в них-то и состоит все волшебство. Здесь мы как раз *хотим*, чтобы произошло наложение, то есть чтобы текст ссылки перекрывал полоску гистограммы. Для этого будем использовать абсолютное позиционирование (вот и еще один случай, где удобно его использовать). Сперва добавим строку `position: relative;` для элементов `<li>` и `<a>`, чтобы задавать координаты каждой полоски относительно соответствующей ей ссылки.

```
ul.lst li {
   position: relative;
   margin: 0 0 2px 0;
   }

ul.lst li a {
   position: relative;
   display: block;
   padding: 7px;
   }

ul.lst li em {
   float: right;
   font-style: normal;
   font-weight: normal;
   color: #9c836e;
   }
```

Далее для элемента  зададим оформление и расположение, чтобы создать полоску гистограммы.

```
ul.lst li span {
   position: absolute;
   top: 0;
   left: 0;
   display: block;
   height: 100%;
   text-indent: –9999px;
   background: #f3f2e8;
   }
```

В этом небольшом стиле есть несколько важных моментов. Сначала мы поместили полоску гистограммы в левый верхний угол. Далее, поскольку элемент  является строковым, мы сделаем так, чтобы он отображался как блоковый (`display: block;`), после чего добавим строку `height: 100%;`, чтобы полоска занимала по высоте всю строку. Кроме того, мы добавили строку `text-indent: –9999px;`, чтобы отодвинуть текст с процентным значением далеко-далеко влево и таким образом скрыть его. Он по-прежнему будет виден программе, преобразующей текст в речь, и другим программам, но не будет отображаться на странице. Для фона мы выбрали светло-коричневый цвет, чтобы сохранить разборчивость текста на полоске.

### Проблема порядка наложения

На рисунке 1.17 показан наш текущий результат, и все бы ничего, да вот только полоска гистограммы полностью закрыла текст. Чтобы решить эту проблему с порядком наложения и передвинуть название и цену напитка на передний план, мы воспользуемся магией `z-index`.


**Рис. 1.17**

`Z-index` — свойство CSS, позволяющее задать порядок наложения позиционируемых элементов. Если добавить в объявление стиля ссылки строку `z-index: 2;`, ее приоритет станет выше, чем у гистограммы, и последняя окажется на заднем плане.

```
ul.lst li a {
   position: relative;
   display: block;
   padding: 7px;
```


Рис. 1.18

```
    z-index: 2;
    }
```

Теперь, когда мы определили значения `z-index`, мы получаем готовый список, в котором полоски гистограммы располагаются за названиями и ценами (рис. 1.18).

### Добавим изменение цвета при наведении курсора

Можно сделать так, чтобы при наведении курсора мыши на строку с названием напитка менялся цвет полоски гистограммы. Так как в разметке полоска находится вне тега ссылки, задать изменение цвета можно через псевдокласс `hover` для элемента <`li`>.

```
ul.lst li: hover span {
    background: #ecebe1;
    }
```

**ПРИМЕЧАНИЕ**

Чтобы применить в IE6 свойство `:hover` к элементам, которые не являются ссылками, используйте JavaScript. То, как это можно делать, иллюстрируют фрагменты скриптов в примере «Suckerfish Dropdowns» на странице *http: //www.htmldog.com/articles/suckerfish/dropdowns/*.


Рис. 1.19

На рисунке 1.19 показано, как будет меняться цвет гистограммы при наведении курсора мыши, если использовать только что описанный способ. Правда такой вариант не будет работать в Internet Explorer 6, ну и ничего страшного. Пускай в Internet Explorer 6 цвет полоски при наведении курсора не меняется. Однако есть другие и более важные проблемы IE6, которые нужно попытаться решить, чтобы гистограммы отображались правильно. Давайте займемся этим прямо сейчас.

## КАК РЕШИТЬ ПРОБЛЕМЫ В INTERNET EXPLORER 6

На рисунке 1.20 показано, как разметка и оформление нашего списка напитков (включая гистограмму) будут отображаться в IE6. Далеко от совершенства, как видите. Обычно я, как и многие из вас, пишу код для Safari или Firefox; создаю чистый, правильный код, основанный на стандартах; а потом открываю его в IE6, огорчаюсь и исправляю. Потом снова запускаю и т. д. К счастью, в нашем случае все не настолько

плохо, и с помощью нескольких исправлений мы сделаем так, чтобы он работал в IE6 так же, как и в других браузерах.

В нашем примере мы видим несколько проблем.

- Расстояние между рядами по вертикали слишком большое.
- Полоска гистограммы занимает по высоте не весь ряд.
- Полоска гистограммы перекрывает цену (но не перекрывает название напитка).


**Рис. 1.20.** При отображении списка в IE6 возникает ряд проблем

Две из этих проблем можно решить с помощью волшебного и таинственного приема `height: 1%;`, применив его к элементам `<li>` и `<a>`. Этот трюк позволяет уплотнить список и решает проблему с наложением гистограммы на цену.

**СОВЕТ**

Почему же `height: 1%;` решает нашу проблему? В IE6 эта команда запускает нечто под названием *hasLayout* (избавлю вас от спутанных объяснений). Для смелых и любопытных даю ссылку, где об этом можно прочитать: *http: //www.satzansatz.de/cssd/onhavinglayout.html*.

Хотя мы хотим использовать этот трюк в IE6, мы в то же время не хотим, чтобы он был виден другим браузерам. Поэтому перед объявлением этих правил мы поместим префикс `* html`. Это удобный способ показать, что CSS-код предназначен для IE6, и только для него.

```
* html ul.lst li,
* html ul.lst li a {
   height: 1%;
   }
```

**СОВЕТ**

Если вам не нравится префикс `*html`, есть еще один способ указать, что код предназначен только для IE6: условные комментарии. Прочитать об этом можно на странице *http: //www.quirksmode.org/css/condcom.html*. Или же объедините все приемы для IE6 в отдельную таблицу стилей (например, *ie. css*), и тогда ваши основные стили смогут соответствовать стандартам и не содержать этих ухищрений.


**Рис. 1.21.** Разрядка по вертикали в IE6 стала лучше, но она еще далека от совершенства

На рисунке 1.21 показано, как изменился наш список после применения трюка `height: 1%;` к элементам списка и ссылкам (список уплотнился, а проблема наложения гистограммы на цену в IE6 исчезла).

Наконец нам необходимо растянуть полоску гистограммы, чтобы она занимала по высоте всю строку. Мы уже задали для элемента `<span>` высоту 100%, но здесь IE6 сработал неудачно. Не в первый и не в последний раз. Так что в качестве дополнительных мер будем использовать атрибут `line-height`, чтобы добиться желаемого результата.

```
ul.lst li span {
   position: absolute;
   top: 0;
   left: 0;
   display: block;
   height: 100%;
   line-height: 2.55em;
   text-indent: -9999px;
   background: #f3f2e8;
   }
```

Я остановился на значении `2.55em` методом проб и ошибок, перебирая варианты до тех пор, пока фоновая гистограмма не достигла нужной высоты. Хотя это правило было добавлено для IE6, оно является безобидным для других браузеров. Поэтому здесь мы не будем специально указывать, что эта строка предназначена для IE6 (и не будем использовать условные комментарии). Мы просто добавим это правило в объявление, общее для всех браузеров.


**Рис. 1.22**

На рисунке 1.22 показан законченный вариант нашего примера, работающий в IE6 и в IE7.

## Выбирая правильное решение

В этой главе мы рассмотрели несколько решений одной задачи и обсудили возможные проблемы и подводные камни. Если для вас не важен порядок разметки, наиболее простым и гибким решением будет закрепить цену у правого края рамки, используя свойство `float` (цена в этом случае будет предшествовать названию напитка). При работе с изображениями, имеющими фиксированную ширину, лучший вариант — это, как правило, абсолютное позиционирование, позволяющее легко избежать наложения. А можно смотреть в будущее, понимая, что в ситуациях вроде этой использование гиперссылок для нескольких блоковых элементов не только может упростить задачу, но и будет выглядеть семантически более правильным.

Есть еще одна вещь, которую важно запомнить: задаваясь вопросами по ходу работы над проектом, вы заботитесь о целостности дизайна. Мы добились поистине пуленепробиваемой гибкости; ее можно считать признаком мастерства: мы позаботились о том, чтобы незапланированные ситуации не влияли на наш дизайн, хотя понять, как это сделать, было подчас очень непросто.

Дальше мы продолжим внимательно и критически рассматривать отдельные части шаблона Tugboat. Мы попробуем понять, в каких случаях можно немного расширить горизонты, и уже сейчас начнем использовать более современные свойства CSS3. Так давайте и займемся этим. Но перед этим… не желаете ли еще чашечку triple-shot latte?

**ПРИМЕЧАНИЕ**

Возможно, вы заметите, что на скриншотах, сделанных в IE6, белый прямоугольник имеет острые углы. Об этом мы поговорим в следующей главе.

# Глава 2

## Закругленные углы с помощью свойства border-radius

**Прежде, чем вы станете скульптором, вам придется закатать рукава и обучиться профессии каменотеса — ремесло всегда предшествует искусству: ученик, подмастерье, мастер.**

Филип Джерард, писатель

Друзья, я хочу сделать смелое заявление: закругленные углы скоро станут *новейшим* трендом в веб-дизайне!

(барабанная дробь…)

Шутка, конечно же. Но никто не станет отрицать, что закругленные углы очень популярны (и, что важнее, практичны), и поэтому занимают свое место среди инструментов дизайнера. Помните, что проблема — не в закругленных углах, а в злоупотреблении ими. По большому счету, скругленный угол — это лишь слегка видоизмененный квадрат. Это метод, который всегда будет применяться в определенных дизайнерских задачах.

При этом *реализация* закругленных углов всегда была рутинной работой. Но ведь исторически наш труд — труд каменотесов, разве не так? Мы вручную добавляем в разметку дополнительные фрагменты кода и вставляем картинки особого цвета, чтобы добиться эффекта, которого лучше было бы добиться при помощи CSS.

В этой главе мы совершим атаку на понятие закругленных углов с помощью свойств CSS3, которые мы уже *сейчас* можем начать использовать. Как увлекательно и невероятно удобно экспериментировать с новой версией технологии CSS, которая раньше на десятилетия отставала от нужд реального мира! К счастью, у нас за плечами богатый опыт каменотесов. И мы осознаем всю трудность этого пути. Вспомнив слова, Филипа Джерарда, приведенные в эпиграфе, и закатаем рукава, чтобы когда-нибудь стать скульпторами.

Итак, коллеги, пришло время лепить.

Мы развивали наше умение, добиваясь эффекта закругленных углов трудными способами, но сейчас пришло время встречать с распростертыми объятьями технологии будущего, переоценивая старые методы и понемногу своими руками облегчая себе жизнь. Хороший мастер знает различные подходы к решению конкретных проблем и способен выбрать наиболее адекватный вариант. Давайте поговорим о том, как немного выйти за рамки привычного, используя продвинутую технологию CSS, которая станет очередным инструментом в копилке дизайнера.

Но сначала давайте перечислим методы, которые используются сейчас, чтобы лучше представить себе, насколько проще может стать жизнь.

# Закругление раньше и сейчас

Как вы могли заметить, в шаблоне Tugboat (рис. 2.1) закругленные углы используются в нескольких местах: прямоугольник с формой Find a Location, модуль Drink Menu, а также рамки вокруг каждого из This Week's Specials.

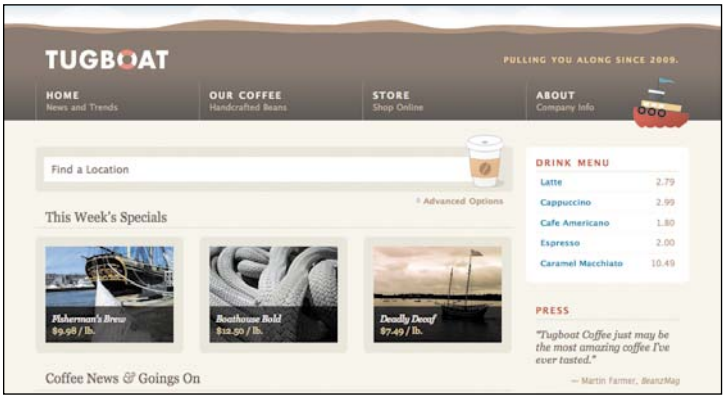



**Рис. 2.1**

Раньше, чтобы это сделать, вам бы пришлось создать несколько изображений и приложить их к квадратному блоку в качестве фона. Если ширина блока фиксирована, вы могли обойтись двумя такими изображениями (одно для левого верхнего и правого верхнего углов, второе — для левого нижнего и правого нижнего).

Давайте вспомним, как создавать блоки фиксированной и переменной ширины с закругленными углами обычными методами. А потом перейдем к новым, интересным вещам.

## Создание закругленного блока фиксированной ширины

На рисунке 2.2 показано, как разбивается на части блок шириной 226 пикселов. Рисунок *round-top.gif* используется как фоновый рисунок для создания закругленных углов в верхней части блока, *round-bottom.gif* — для создания закругленных углов

**ПРИМЕЧАНИЕ**

Я специально не собираюсь вдаваться в подробности относительно фрагментов кода, которые я здесь привожу, так как моя цель — лишь напомнить о том, как работают эти методы, и я не хочу тратить время на детальный разбор устаревших решений.

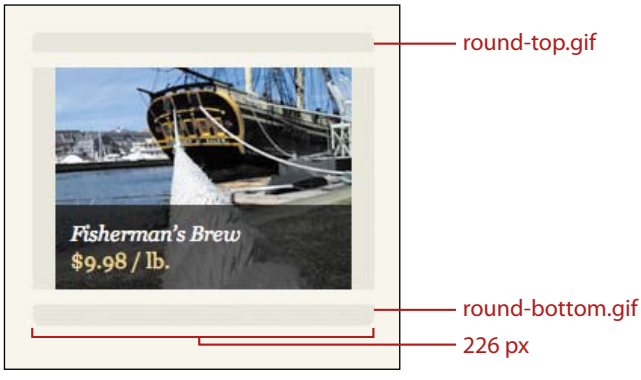



**Рис. 2.2**

в нижней части. Часть блока, в которой закругленных углов нет, можно заполнить фоновым цветом.

Этот метод требует присоединить к картинке с лодкой как минимум два элемента. Поскольку в спецификации CSS2.1 разрешено добавлять лишь одно фоновое изображение к одному элементу, потребуется усложнить разметку вокруг фотографии.

```
<div class="box">
   <div class="box-inner">
      <img src="boat.jpg" alt="boat" />
   </div>
</div>
```

Далее, таблицы стилей для каждого фонового изображения будут выглядеть примерно так.

```
.box {
   width: 226px;
   background: #e2e1d4 url(round-bottom.gif)
     no-repeat bottom left;
   }
.box-inner {
   padding: 15px;
   background: url(round-top.gif) no-repeat top
     left;
   }
```

Цвет фона обязательно задавать во внешнем элементе `<div class="box">`. Если сделать это во внутреннем (`<div class="box-inner">`), он перекроет картинку, и ее будет не видно. А отступ, наоборот, нужно задавать во внутреннем

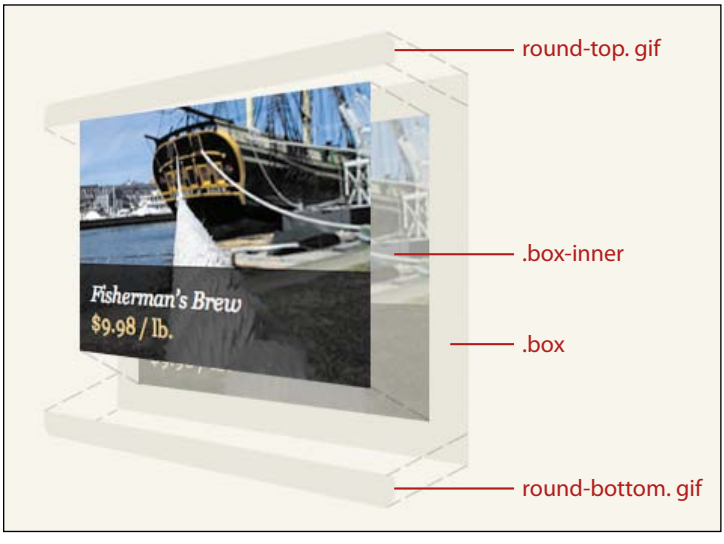



**Рис. 2.3**

элементе `<div class="box-inner">`, чтобы верхняя картинка заняла всю ширину и ее углы совпали с углами блока (рис. 2.3).

Если вы уже занимались закругленными углами, все это должно было показаться вам знакомым. Конечно же, нет ничего страшного в том, что приходится использовать дополнительный элемент `<div>` для добавления еще одного фонового рисунка, и такой метод хорошо работает в разных браузерах. Однако он требует, чтобы блок имел фиксированную ширину.

## СОЗДАНИЕ ЗАКРУГЛЕННОГО БЛОКА ПЕРЕМЕННОЙ ШИРИНЫ

Если блок имеет переменную ширину (или если вы не хотите ограничиваться конкретным значением), то вам придется создать четыре изображения — по одному для каждого угла. Кроме того, вам понадобятся четыре элемента-контейнера, чтобы присоединить каждое из четырех фоновых изображений.

На рисунке 2.4 показано, как должны выглядеть четыре отдельных изображения: *round-tl.gif*, *round-tr.gif*, *round-bl.gif* и *round-br.gif* (по одному для каждого угла).

Разметка может быть организована множеством различных способов, но вокруг основного изображения должно быть не менее четырех элементов, чтобы все четыре угла можно

было прикрепить в качестве фона. Для создания следующего примера будем использовать элементы .

```
<div class="box">
   <div class="box-inner">
      <div class="box-inner2">
         <div class="box-inner3">
            <img src="boat.jpg" alt="boat" />
         </div>
      </div>
   </div>
</div>
```

Чтобы превратить эти четыре элемента в закругленный блок переменной длины, создадим такие таблицы стилей.

```
.box {
   background: #e2e1d4 url(round-br.gif)
     no-repeat bottom right;
   }
.box-inner {
   background: url(round-bl.gif) no-repeat bottom
     left;
   }
.box-inner2 {
   background: url(round-tl.gif) no-repeat top
     left;
   }
.box-inner3 {
```


**Рис. 2.4.** Изображения четырех углов, которые можно наложить на прямоугольный блок

```
    padding: 15px;
    background: url(round-tr.gif) no-repeat top
      right;
    }
```

Как и при создании блока фиксированной ширины, цвет фона должен быть задан во внешнем элементе `<div>`, а отступ — во внутреннем.

Ну что, такой метод создания закругленных углов еще не кажется вам абсурдным?

**СОВЕТ**

Чтобы узнать, как создавать закругленные блоки переменной ширины с помощью JavaScript и DOM, посмотрите статью Саймона Уиллисона (Simon Willison) «Rounded Corners with CSS and JavaScript» (*http://articles.sitepoint.com/article/rounded-corners-css-javascript*).

Конечно, существуют и другие методы создания блоков переменной величины с закругленными углами. Например, с помощью JavaScript можно *динамически* создавать нужные фрагменты разметки, а также управлять расположением четырех угловых изображений. В таком случае вам придется создавать минимум разметки, однако для того, чтобы закругленные углы появились, должен быть включен JavaScript.

## ПРОБЛЕМА С ЦВЕТОМ И РАДИУСОМ

У всех решений, описанных выше, есть недостатки. Например, при создании фоновых изображений должен быть известен цвет закругленного блока и окружающего его фона. А если после этого цвета изменятся, придется создавать новые изображения.

Часто углы сглаживаются за счет того, что часть фонового изображения сливается с фоном страницы (рис. 2.5). Поэтому если после создания этих изображений цвета меняются, возникают серьезные неудобства.

Степень закругленности должна быть определена одновременно с цветами. Радиус закругленного угла — часть изображения, и для изменения этого изгиба потребуется создавать новые фоновые изображения каждый раз, когда это придет в голову вашему заказчику или боссу.


**Рис. 2.5.** Изображение угла в увеличенном масштабе. При его наложении на фон страницы, цвет которого известен заранее, острый угол сглаживается

Короче говоря, закругленные углы — упрямые, требующие большого труда существа. Наверное, излишне вам об этом напоминать. Но есть способ лучше, и это новый способ.

## Свойство border-radius

CSS3 обещает, что свойство `border-radius` позволит очень просто добавлять закругленные углы к любому элементу, причем не нужно будет создавать никаких изображений и перестраивать разметку. А цвет и степень закругленности можно будет менять на ходу, просто обновляя соответствующие строки CSS.

Чтобы показать, как работает свойство `border-radius`, я приведу короткие фрагменты кода. Для нашего последнего примера разметка может выглядеть очень просто.

```
<div class="box">
   <img src="boat.jpg" alt="boat" />
</div>
```

Если мы хотим сделать круглыми все четыре угла, нам понадобится такой стиль.

```
.box{
   padding: 15px;
   background: #e2e1d4;
   border-radius: 8px;
   }
```

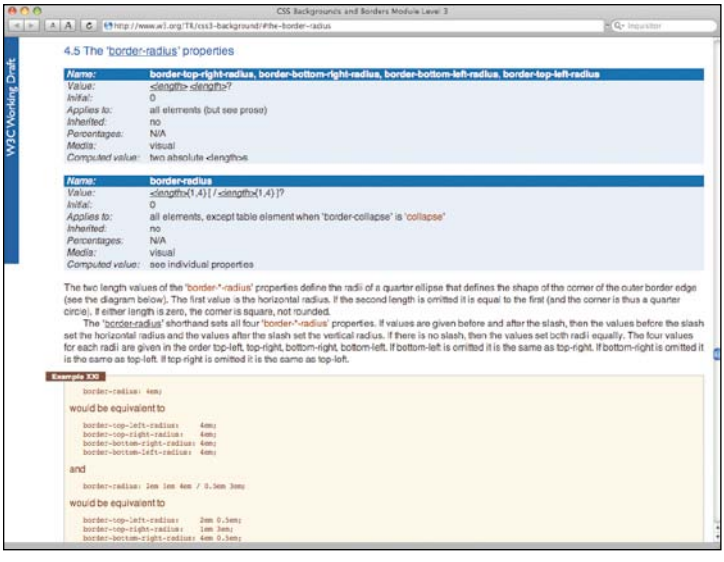



**Рис. 2.6.** Рабочий вариант border-radius на W3C

В этом объявлении мы задали для всех углов радиус закругления 8 пикселов; при этом используется цвет фона, который мы задали для блока. И все это — магия CSS. Готово.

Можно за считанные секунды задать радиус или цвет фона, изменив оформление блока с помощью нескольких простых строк.

Пока эти возможности относятся к рабочей версии CSS3 (рис. 2.6) (*http://www.w3.org/TR/css3-background/#the-border-radius*). Получается, мы сможем воспользоваться этим модерновым свойством только через лет десять?

К счастью, пользователям некоторых браузеров этого долго ждать не придется. Вообще-то есть возможность использовать свойство `border-radius` прямо сейчас. Давайте посмотрим, как это сделать.

## Расширения, специфичные для производителей

В CSS существуют расширения, специфичные для конкретных производителей. Так, производитель браузеров может создать свои собственные свойства, добавляя имя организации в качестве префикса к названию свойств.

Зачастую это отличный способ проверить новые свойства CSS, которые еще не введены в качестве официальных стандартов. Благодаря этому, браузер может протестировать и отладить эти свойства, пока спецификация находится в стадии разработки.

В таблице 2.1 приведены префиксы, существующие на момент написания этой книги (*http://reference.sitepoint.com/css/vendorspecific*).

**Таблица 2.1.** Расширения, специфичные для конкретных производителей браузеров

| Префикс | Организация |
|---|---|
| `-ms-` | Microsoft |
| `-mso-` | Microsoft Office |
| `-moz-` | Mozilla Foundation (браузеры на движке Gecko) |
| `-o-` | Opera Software |
| `-atsc-` | Advanced Television Standards Committee |
| `-wap-` | The WAP Forum |
| `-webkit-` | Safari (и другие браузеры на движке WebKit) |
| `-khtml-` | Браузер Konqueror |

Интересно, что свойство `border-radius` относится к числу тех, которые уже проходят проверку у производителей браузеров.

В настоящий момент семейство браузеров Mozilla и WebKit (к нему относятся, в частности, Firefox и Safari) обеспечили неслабую поддержку свойства `border-radius` с использованием префиксов производителей. Ура! Мы можем прямо сейчас воспользоваться этими свойствами. Сегодня же. Так давайте посмотрим, как они работают.

## Прогрессивное оформление с помощью -webkit-border-radius и -moz-border-radius

Если мы рассматриваем закругленные углы как бонус для тех браузеров, которые могут с ними работать, а не как обязательную часть оформления, видимую во всех программах просмотра веб-страниц, мы можем создать прогрессивное оформление наших страничек с помощью версий `border-radius`, специфичных для производителей, и для этого нам не придется ждать, пока CSS3 достигнет статуса «Candidate Recommendation».

Ниже показано, как в настоящий момент можно добавлять закругленные углы, используя расширения для Safari и Firefox (а также других браузеров на движках Mozilla и WebKit). Разметка останется кристально чистой: создадим один контейнер, к которому мы будем применять стили.

```
<div class="box">
   <img src="boat.jpg" alt="boat" />
</div>
```

Чтобы все четыре угла были закругленными, будем использовать такие стили (для Firefox и Safari).

```
.box {
   padding: 15px;
   background: #e2e1d4;
   -webkit-border-radius: 8px;
   -moz-border-radius: 8px;
   }
```

Если применить эти правила, вас ждет успех с браузерами на движках Mozilla и WebKit: вы увидите четыре закругленных угла с радиусом 8 пикселов (независимо от ширины блока). Иными словами, используя *CSS* для сглаживания углов, вы можете быть уверены в том, что блок будет *на самом деле* растяжимым во всех направлениях и вам не нужно будет беспокоиться о фоновых изображениях и цветах.

## ВЗГЛЯД В БУДУЩЕЕ

Можно сделать шаг вперед и добавить свойство CSS3 в подлинном виде, чтобы в будущем все браузеры его распознавали.

```
.box {
   padding: 15px;
```

**ПРИМЕЧАНИЕ**

Здесь я должен вас предостеречь. CSS-валидаторы (которые проверяют код на соответствие спецификации CSS2.1) споткнутся на свойстве border-radius, так как оно относится к еще не завершенной спецификации. Файл, в котором содержатся таблицы стилей с такими «свойствами будущего», будет расцениваться как недопустимый. Если вас устраивают изначально недопустимые таблицы стилей, оставьте так. Если нет, существует способ выделить эти свойства в отдельные таблицы стилей. Об этом мы поговорим чуть позже.

```
    background: #e2e1d4;
border-radius: 8px;
    -webkit-border-radius: 8px;
    -moz-border-radius: 8px;
    }
```

Из трех правил, добавленных в объявление, браузер выберет то, которое он распознает, и просто проигнорирует остальные.

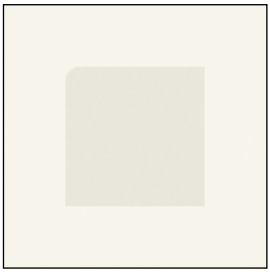


Рис. 2.7

## КАК ЗАКРУГЛИТЬ КОНКРЕТНЫЕ УГЛЫ

До сих пор мы с помощью одного правила добивались сглаживания всех четырех углов. Но можно сделать так, чтобы закругленными были только определенные углы. Для этого используется следующий синтаксис.

Чтобы закруглить только левый верхний угол (рис. 2.7), можно написать такой код.

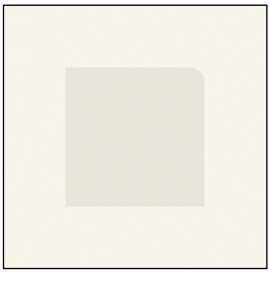


Рис. 2.8

```
.box {
    padding: 15px;
    background: #e2e1d4;
    border-top-left-radius: 8px;
    -webkit-border-top-left-radius: 8px;
    -moz-border-radius-topleft: 8px;
    }
```

Обратите внимание, что для версии `-moz-` синтаксис немного не такой, как для CSS3 и `-webkit-`: `topleft` пишется слитно и располагается в самом конце.

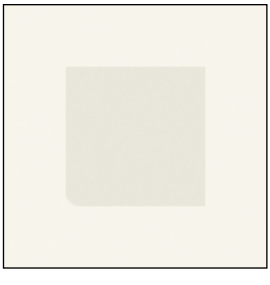


Рис. 2.9

Можно закруглить и другие углы, используя аналогичный синтаксис.

Чтобы закруглить только правый верхний угол (рис. 2.8), можно написать такой код.

```
border-top-right-radius: 8px;
-webkit-border-top-right-radius: 8px;
-moz-border-radius-topright: 8px;
```

Синтаксис для левого нижнего угла (рис. 2.9):

```
border-bottom-left-radius: 8px;
-webkit-border-bottom-left-radius: 8px;
-moz-border-radius-bottomleft: 8px;
```

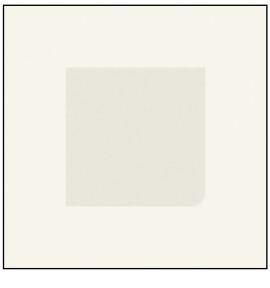


Рис. 2.10

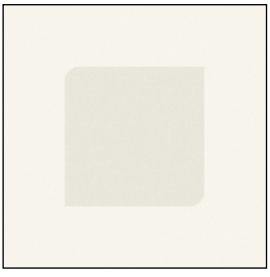


Рис. 2.11

Для правого нижнего угла (рис. 2.10):

```
border-bottom-right-radius: 8px;
-webkit-border-bottom-right-radius: 8px;
-moz-border-radius-bottomright: 8px;
```

Возможны любые комбинации: например, закругление левого верхнего и правого нижнего углов (рис. 2.11).

```
.box {
   padding: 15px;
   background: #e2e1d4;
   border-top-left-radius: 8px;
   border-bottom-right-radius: 8px;
   -webkit-border-top-left-radius: 8px;
   -webkit-border-bottom-right-radius: 8px;
   -moz-border-radius-topleft: 8px;
   -moz-border-radius-bottomright: 8px;
   }
```

В общем и целом, это *очень* простой набор правил, благодаря которому использование закругленных углов перестает быть источником головной боли. При этом разметка становится компактной и эффективной, а всю «черную» работу берет на себя CSS.

## Небольшая проблема в Firefox 2

Важно обратить внимание на то, что реализация свойства `-moz-border-radius` в Firefox 2 была немного странной: углы сглаживались плохо. Хотя браузеру можно дать поблажку, так как эта реализация была одной из первых и ее можно считать экспериментальной. К счастью, Firefox 3 избавился от таких проблем и по качеству сглаживания углов стоит в одном ряду с Safari.


**Рис. 2.12.** Сравнение реализации свойства border-radius в Safari, Firefox 2 и Firefox 3

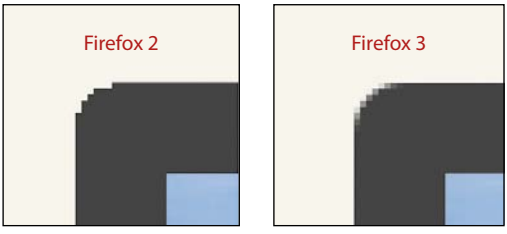



**Рис. 2.13.** Изображение угла в увеличенном масштабе, позволяющее сравнить, как свойство border-radius реализовано в Firefox 2 и Firefox 3

Посмотрев на рисунок 2.12, можно сравнить, как Firefox 2, Firefox 3 и Safari отображают один и тот же элемент с закругленным углом. Я сделал цвет фона более темным (#333), чтобы подчеркнуть контраст и чтобы вы действительно заметили разницу.

Видите, насколько неровным и ступенчатым выглядит угол в Firefox 2? При большом контрасте между закругленным элементом и фоном это становится особенно заметным и вызывает чувство разочарования. Это стоит иметь в виду.

К счастью, в Firefox 3 эта проблема полностью решена: изменение радиуса происходит куда более плавно (рис. 2.13).

## ПРИ НИЗКОМ КОНТРАСТЕ — ОЧЕНЬ ДАЖЕ НЕПЛОХО

Можно запросто принять этот вариант для Firefox 2, если контраст будет достаточно низким. Например, в нашем шаблоне Tugboat это именно так, и если мы проведем аналогичное сравнение, то обнаружим, что даже в Firefox 2 углы выглядят хорошо (рис. 2.14).

Это же превосходно! Хотя Firefox 2 был выпущен около 3 лет назад (на момент написания этой книги), он все же неплохо отображает закругленные углы при низкой контрастности цветов.


**Рис. 2.14.** Проблема с изображением закругленных углов в Firefox 2 становится почти незаметной, если контраст достаточно низкий

## Отсечение фона

Можно комбинировать фоновые *изображения* и `border-radius`. Фоновые изображения, расположенные слоями, будут также правильно обрезаны и закруглены, и при этом вы сможете добиться интересных эффектов.

Например, внизу в разделе «Coffee News» нашего шаблона Tugboat есть кнопка, открывающая архив новостей (рис. 2.15). Хотя можно запросто создать для этой кнопки картинку, почему бы не извлечь выгоду из свойства `border-radius` и фонового изображения, создав вместо этого гибкую редактируемую гиперссылку?


**Рис. 2.15**

### Простая гиперссылка

К кнопке добавлен глянцевый эффект (рис. 2.16), которого можно добиться с помощью хитро расположенного повторяющегося по горизонтали полупрозрачного фонового изображения в формате PNG. Для оформления того, что, по сути, является простой гиперссылкой, достаточно будет применить свойство `border-radius` для закругления краев кнопки и добавить отсечение повторяющегося изображения. Разметка будет достаточно простой.


**Рис. 2.16.** Кнопка в увеличенном масштабе. Здесь можно увидеть глянцевый эффект, созданный с помощью полупрозрачного фонового PNG-изображения

```
<div class="more-btn"><a href="/archives/">&larr;
     News Archives</a></div>
```

Мы заключили элемент `<a>` в `<div>`, создав класс `more-btn`. Вы также наверняка заметите, что мы использовали сущность HTML для стрелки, направленной влево, добавив `&larr;` перед текстом ссылки.

**ПРИМЕЧАНИЕ**

Элемент `<div>` необязателен, но является допустимым блоковым элементом, в который может быть заключена гиперссылка, расположенная на отдельной строке.

## СОЗДАНИЕ ФОНОВОГО PNG-ИЗОБРАЖЕНИЯ

Наш следующий шаг — это создание повторяющегося PNG-изображения, которое «наведет глянец» на синий фон. Выполните следующие действия.

В редакторе Photoshop создайте новый файл шириной 50 пикселов и высотой 100 пикселов.

Заполните новый слой белым цветом.

Увеличьте размер холста вдвое, добавив 100 пикселов снизу.

Уменьшите непрозрачность белого слоя примерно до 10%.

Сохраните изображение в формате PNG-24 (вам, возможно, предложат вариант PNG-8, но он не поддерживает альфа-каналы с данными о прозрачности).

В результате получается изображение высотой 200 пикселов, верхняя половина которого является белой при 10-процентной непрозрачности, а нижняя — полностью прозрачной (рис. 2.17).

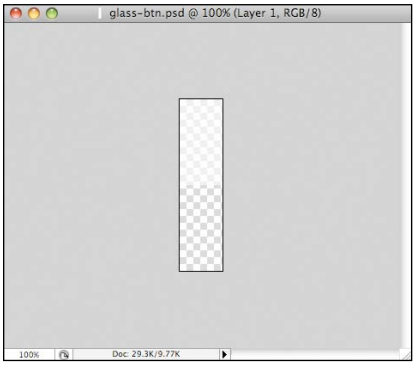



**Рис. 2.17**

**СОВЕТ**

Размеры здесь выбраны произвольно, поскольку изображение значительно больше, чем вам может понадобиться. Это — мера предосторожности на случай, если текст или отступ вокруг ссылки уменьшится или увеличится. Кроме того, такое изображение мы можем использовать для других кнопок разных размеров на нашем сайте. Как правило, я создаю такие изображения шириной (или высотой) не меньше 2 пикселов, заботясь о том, чтобы заполнение было плавным во всех браузерах (некоторым браузерам не нравится использовать для этого изображения шириной или высотой в 1 пиксел).

## СТИЛИ, КОТОРЫЕ СОЗДАЮТ КНОПКУ

Теперь мы готовы создать таблицы стилей, которые соберут из этих элементов глянцевую закругленную кнопку в форме пилюли.

Сначала добавим отступ и белый фон, а также сделаем цвет ссылки белым (рис. 2.18).

```
div.more-btn a {
   padding: 5px 14px;
   color: #fff;
   background: #3792b3;
   }
```


**Рис. 2.18**

Далее, добавим ранее созданное PNG-изображение, повторяя его по горизонтали (`repeat-x`) и центрируя по вертикальной оси (`0 50%`).


Рис. 2.19

```
div.more-btn a {
   padding: 5px 14px;
   color: #fff;
   background: #3792b3 url(img/glass-btn.png)
     repeat-x 0 50%;
   }
```

На рисунке 2.19 показано, как с помощью изображения *glass-btn.png* удалось заполнить синий фон. Так как нижняя часть изображения полностью прозрачна и мы отцентрировали его по вертикальной оси, полупрозрачная часть изображения попадает только на верхнюю половину кнопки, тем самым создавая аккуратный глянец — последний писк моды.

Теперь воспользуемся свойством `border-radius`, чтобы сделать края кнопки закругленными в браузерах на движках Mozilla и WebKit, а также в программах просмотра, которые будут поддерживать это свойство в будущем (этот способ обсуждался ранее в этой главе). Результат изображен на рисунке 2.20.


Рис. 2.20

```
div.more-btn a {
   padding: 5px 14px;
   color: #fff;
   background: #3792b3 url(img/glass-btn.png)
     repeat-x 0 50%;
   border-radius: 14px;
   -webkit-border-radius: 14px;
   -moz-border-radius: 14px;
   }
```

## КАК ПРОСТО ЗАДАТЬ СОСТОЯНИЯ :HOVER

Добавить `hover`-эффект для кнопки очень просто: нам понадобится небольшое объявление, которое всего лишь задает новое значение `background-color`. Сделаем так, чтобы при наведении курсора мыши цвет текста ссылки и изображение *glass-btn.png* оставались на месте, но цвет фона менялся на красный (рис. 2.21).


Рис. 2.21

```
div.more-btn a: hover {
   background-color: #a14141;
   }
```

Этот пример показывает, что такой способ создания кнопок и других диалоговых элементов с закруглением обладает огромными возможностями и гибкостью. Можно моментально изменять цвет, размер, шрифт и состояния `:hover`, добавляя всего несколько строк кода.

## КАК ДОБАВИТЬ ГРАНИЦУ

Можно добавить дополнительную деталь — едва заметную границу кнопки, которая будет того же цвета, что и сама кнопка. Так мы сможем немного отодвинуть глянцевый эффект от краев, создавая ощущение объемности всей кнопки.

```
div.more-btn a {
   padding: 5px 14px;
   color: #fff;
   border: 1px solid #3792b3;
   background: #3792b3 url(img/glass-btn.png)
     repeat-x 0 50%;
   border-radius: 14px;
   -webkit-border-radius: 14px;
   -moz-border-radius: 14px;
   }
```

На рисунке 2.22 можно в увеличенном масштабе посмотреть на нашу границу шириной в 1 пиксел. Она практически незаметна, но этот эффект все равно стоило обсудить. Попробуйте поэкспериментировать с границей, увеличив ее ширину.


**Рис. 2.22.** Кнопка в увеличенном масштабе. Здесь лучше видно границу шириной в 1 пиксел, которая отодвигает глянцевый эффект от края

Кстати, я как раз сейчас вспомнил: свойство `border-radius` делает закругленным не только любой фон элемента (как цвет, так и изображение), но также его границу, как в нашем примере. Давайте для примера возьмем нашу кнопку, но уберем

цвет фона и PNG-изображение (сделаем его белым). Сделаем границу толще (5 пикселов) и добавим небольшое закругление.


**Рис. 2.23**

```
div.more-btn a {
    padding: 5px 14px;
    border: 5px solid #e2e1d4;
    background: #fff;
    border-radius: 5px;
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    }
```

На рисунке 2.23 видно, что теперь свойство `border-radius` вместо фона, который мы использовали раньше, закругляет границу, заданную с помощью CSS.

### ОТСЕЧЕНИЕ ФОНА НЕ РАБОТАЕТ В FIREFOX 2

Стоит отметить, что, к сожалению, в Firefox 2 фоновые изображения обрезаются неправильно. На примере нашей конкретной кнопки этой проблемы не видно, так как полупрозрачное PNG-изображение не сильно влияет на светло-коричневый фон. А вот если сделать фон страницы более темным, вы увидите, что `-moz-border-radius` не обрезает фоновое изображение в Firefox 2 (рис. 2.24). Вы также не сможете не заметить ужасно сглаженных углов, которые выглядели бы здорово году, разве что, в 1983-м. Вот что стоит иметь в виду: реализация свойства `border-radius` в Firefox 2 работает лучше всего при низкой контрастности, а в Firefox 3 она работает правильно всегда.


**Рис. 2.24.** Кнопка в увеличенном масштабе в Firefox 2, который не поддерживает отсечку фона

## Закругление элементов формы

Пока мы применяли свойство `border-radius` для элементов `<div>` и `<a>`. Но его также можно использовать для закругления чего угодно, включая элементы формы. Используя цвет фона, границы, повторяющееся фоновое изображение и `border-radius`, вы можете легко создавать изящные и гибкие поля ввода.

Давайте быстро рассмотрим простую форму добавления комментария, которую мы вставим в наш шаблон Tugboat.

## ПРОСТАЯ РАЗМЕТКА ФОРМЫ

Сначала создадим простую форму, которая будет содержать поля ввода Name и Email, а также поле `<textarea>` для комментариев.

```
<form id="comment-form" action="/">
   <fieldset>
      <label for="name">Name</label>
      <input id="name" type="text" />
   </fieldset>
   <fieldset>
      <label for="email">Email</label>
      <input id="email" type="text" />
   </fieldset>
   <fieldset>
      <label for="comment">Comment</label>
      <textarea id="comment"></textarea>
   </fieldset>
</form>
```

Для каждого раздела формы, содержащего метку и поле ввода, будем использовать элемент `<fieldset>`.

На рисунке 2.25 показана форма, которую мы добавили к шаблону Tugboat (пока без оформления).

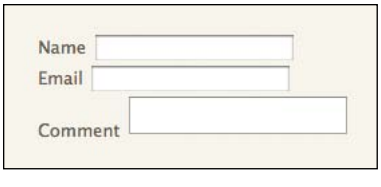



**Рис. 2.25**

Теперь можно создать таблицы стилей и добавить оформление для этого элемента.

## ПРИМЕНЕНИЕ БАЗОВЫХ СТИЛЕЙ

Сначала давайте добавим нижний отступ для каждого элемента `<fieldset>`, чтобы между ними было небольшое расстояние. Кроме того, добавим `display: block;` к элементам `<label>`, так как по умолчанию они являются строковыми. Тогда все метки и поля ввода будут располагаться на отдельных строках.


**Рис. 2.26**

А заодно добавим небольшой отступ снизу для элементов `<label>` и сделаем шрифт полужирным (рис. 2.26).

```
#comment-form fieldset {
   margin: 0 0 15px 0;
   }

#comment-form fieldset label {
   display: block;
   margin: 0 0 3px 0;
   font-weight: bold;
   }
```

## ДОБАВЛЕНИЕ ФОНА И ИЗМЕНЕНИЕ ГРАНИЦ

Теперь добавим сложное объявление, чтобы одновременно оформить элементы `<input>` и `<textarea>`. Зададим ширину (здесь я выбрал произвольное значение `400px`), добавим отступ, увеличим размер шрифта, удалим границы по умолчанию и сделаем цвет чуть более темным, чем фон страницы (рис. 2.27).

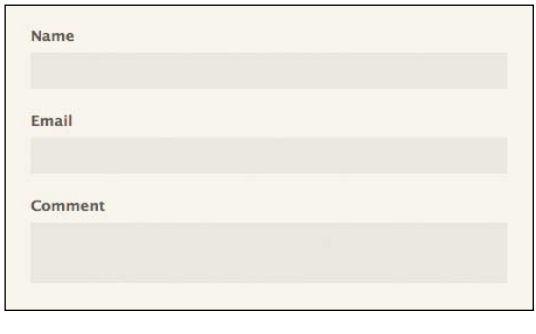



**Рис. 2.27**

```
#comment-form fieldset input,
#comment-form fieldset textarea {
   width: 400px;
   padding: 5px;
   font-size: 1.4em;
   border: none;
   background: #e2e1d7;
}
```

## СОЗДАНИЕ ЭФФЕКТА ГЛУБИНЫ

Давайте теперь добавим немного глубины, используя в качестве тени маленькое GIF-изображение, которое мы размножим по горизонтали вдоль верхнего края элементов формы.

На рисунке 2.28 показано такое GIF-изображение. Оно занимает всего несколько пикселов в высоту, а его цвет из темно-коричневого постепенно переходит в цвет фона элемента формы.

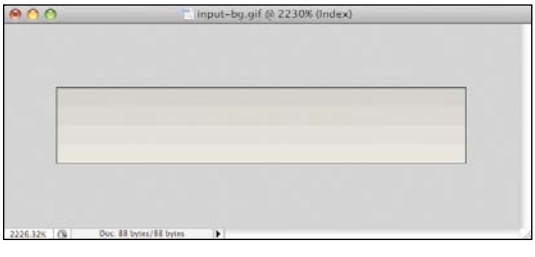



**Рис. 2.28**

Добавим это изображение в объявление, задающее стиль элементов `<input>` и `<textarea>`, повторяя его по горизонтали вдоль верхнего края.

```
#comment-form fieldset input,
#comment-form fieldset textarea {
   width: 400px;
   padding: 5px;
   font-size: 1.4em;
   border: none;
   background: #e2e1d7 url(../img/input-bg.gif)
     repeat-x top left;
}
```


**Рис. 2.29**

На рисунке 2.29 показана наша форма с эффектом глубины, полученным с помощью градиента. Так форма выглядит лучше, но ее можно сделать более изящной, если добавить границы и воспользоваться свойством `border-radius`.

## УЛУЧШЕНИЕ ОФОРМЛЕНИЯ С ПОМОЩЬЮ СВОЙСТВ BORDER И BORDER-RADIUS

Чтобы дополнить наш дизайн, можно добавить белую границу шириной в 1 пиксел вдоль нижнего и правого краев элементов формы. Так мы добьемся ощущения, что свет падает слева сверху и создает тень на внутренней части поля (рис. 2.30).

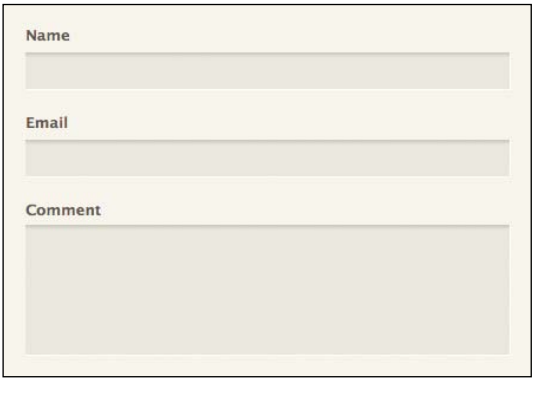



**Рис. 2.30**

```
#comment-form fieldset input,
#comment-form fieldset textarea {
   width: 400px;
```

```
    padding: 5px;
    font-size: 1.4em;
    border: none;
    border-bottom: 1px solid #fff;
    border-right: 1px solid #fff;
    background: #e2e1d7
    url(../img/input-bg.gif) repeat-x top left;
    }
```

Наконец, добавим свойства `border-radius` (которые мы уже использовали в этой главе), чтобы закруглить углы элементов `<input>` и `<textarea>` в браузерах на движках Mozilla и WebKit.


**Рис. 2.31**

```
#comment-form fieldset input,
#comment-form fieldset textarea {
    width: 400px;
    padding: 5px;
    font-size: 1.4em;
    border: none;
    border-bottom: 1px solid #fff;
    border-right: 1px solid #fff;
    background: #e2e1d7 url(../img/input-bg.gif)
      repeat-x top left;
    border-radius: 5px;
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    }
```

На рисунке 2.31 показан окончательный вариант формы (открытый в Safari). Обратите внимание, что фон и выделения границ обрезаны и закруглены, в результате чего мы получили стильное трехмерное оформление элементов формы.

### ОБЪЯВЛЕНИЕ СТИЛЯ :FOCUS

Делая еще один шаг вперед, определим, каким должен быть стиль формы, если пользователь щелкнет на поле мышью. Ниже вы видите небольшое объявление, использующее псевдоэлемент `:focus` и делающее фон полей `<input>` и `<textarea>` белым (рис. 2.32).

```
#comment-form fieldset input:focus,
#comment-form fieldset textarea:focus {
   background: #fff;
   }
```


**Рис. 2.32**

Мы переопределяем только цвет фона и изображение, а остальные стили (включая закругленные углы) наследуются от предыдущего объявления.

## Как насчет других браузеров?

Знаю, что вы теперь скажете: «Дэн, это все, конечно, хорошо, но как насчет других браузеров?»

Мы знаем, что в настоящий момент свойство `border-radius` поддерживают браузеры на движках Mozilla и WebKit (при помощи волшебных префиксов, специфичных для производителей: `-moz-` и `-webkit-`). А если к этим специальным свойствам

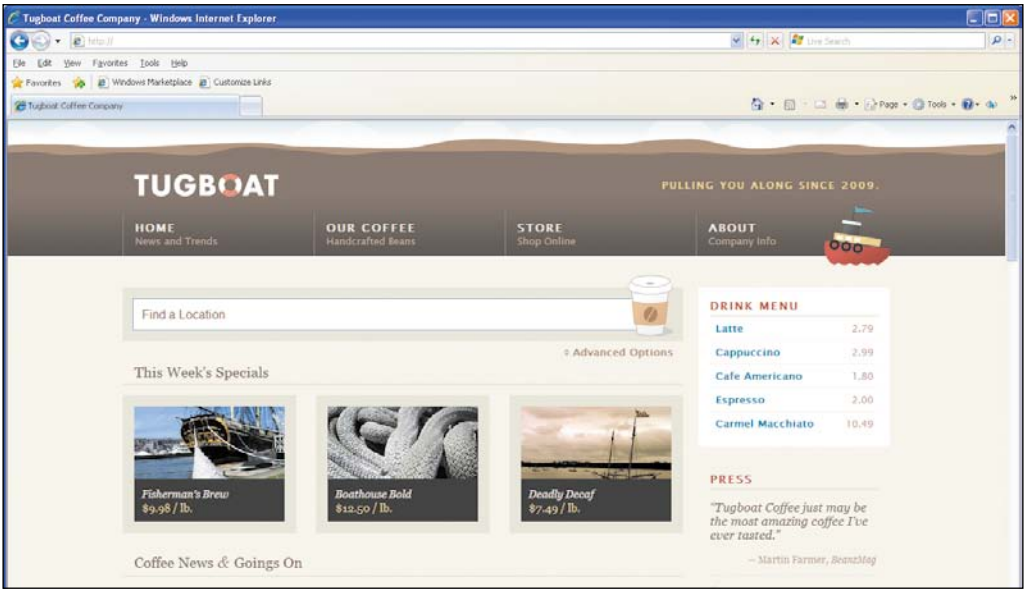



**Рис. 2.33**

добавить свойство CSS3 `border-radius` (как мы делали в этой главе), мы обеспечим возможность распознавания `border-radius` браузерами, которые в будущем будут его поддерживать.

Невозможно игнорировать поведение Internet Explorer, который на момент написания этой книги не поддерживает свойств CSS3 и не имеет расширений для CSS3, специфичных для производителей. А это значит, что в этом браузере мы будем видеть *прямоугольные* блоки, кнопки и элементы формы.

Давайте вспомним пример, который мы создавали на протяжении этой главы, и посмотрим на то, как его элементы будут отображаться в браузерах, не поддерживающих `border-radius`.

На рисунке 2.33 показан шаблон Tugboat, открытый в IE7. Налицо полное отсутствие закругленных углов. Так как IE7 не распознает `border-radius`, он благополучно игнорирует эти свойства и просто отображает прямоугольный блок. А все прочее остается без изменений.

## НИЧЕГО СТРАШНОГО

Я понимаю, что ваш босс или заказчик, возможно, и *не* отреагирует на это словами «Ничего страшного!». Но здесь важно понимать, что качество исполнения этого метода *ухудшается красиво*. Блоки, отступы и расположение элементов остаются без

изменений. Дизайн страницы не нарушается. Он сохраняет свою функциональность и четкость. Единственное отличие в том, что элементы, углы которых закруглены в браузерах более предусмотрительных производителей, здесь выглядят прямоугольными.

## ПРЯМОУГОЛЬНАЯ КНОПКА

А как выглядит в IE7 кнопка из нашего примера? На рисунке 2.34 мы видим вполне изящную, хотя и прямоугольную, кнопку. В ней тот же отступ, текст и глянцевый фон. Она сохраняет свою функциональность и пропорции. Опять же, ничего страшного.


**Рис. 2.34.** Кнопка в IE7 (как и в любом другом браузере, не поддерживающем border-radius)

## ПРЯМОУГОЛЬНЫЕ ЭЛЕМЕНТЫ

А что там с закругленными элементами `<input>` и `<textarea>`? Что ж, они тоже прямоугольные, но опять качество ухудшается красиво: сохраняется теневое фоновое изображение и цвет, а также выделенные границы (рис. 2.35). Изящная форма, даже если углы не закруглены. Разве не так?

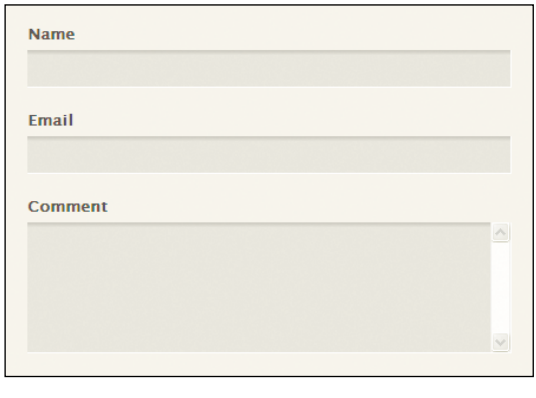



**Рис. 2.35**

И в последний раз давайте вместе произнесем эту мантру: *ничего страшного*.

## ПРОГРЕССИВНОЕ ОФОРМЛЕНИЕ

Некоторым бывает трудно смириться с тем, что дизайн выглядит по-разному в зависимости от браузера. Но, поощряя браузеры, которые *могут* работать с продвинутыми свойствами CSS, вы создаете гибкие, удобные дизайнерские решения, которые способствуют *развитию* новых стандартов.

**ПРИМЕЧАНИЕ**

Я приведу гораздо больше аргументов за прогрессивное оформление в главе 4 и постараюсь убедить вас (а также ваших заказчиков и боссов) в том, что полезно иногда выходить за привычные рамки.

## Замечательно для создания прототипа

Я хочу завершить эту главу, рассказав об одной значительной детали: независимо от того, сможете ли вы использовать `border-radius` *при создании проекта*, важно (и забавно) экспериментировать с этими прогрессивными методами. Они являются не только залогом прогресса, но и великолепным инструментом для создания прототипа.

Для меня разработка проектов для Сети — это зачастую итеративный процесс. Я могу начать с разметки и CSS-каркасов, постепенно добавляя в код какие-то мелочи, или же перенести скриншот этой страницы в Photoshop и там его причесать.

На ранней стадии дизайна свойство `border-radius` — мощный инструмент для создания прототипа. Без возможности быстро добавлять несколько правил, отвечающих за цвет и/или степень закругленности (значения параметров можно потом изменить), просто нельзя обойтись. Это ускоряет эволюцию проекта, не разрушая его. Гораздо проще переписать несколько строк кода, чтобы изменить закругленные углы шаблона, чем постоянно перерисовывать их в графическом редакторе, обрезать и снова задавать в качестве фона. Бывало и такое, что я использовал свойство `border-radius` для создания прототипа закругленных углов, только чтобы потом использовать экранные копии, снятые с реализации этого шаблона.

Так что если и есть в этой главе мораль, которую, я надеюсь, вы прочувствуете, то она состоит вот в чем: `border-radius` (и другие свойства CSS3) — это уже нечто вполне реальное; это ценный инструмент, который мы можем начать использовать при разработке и создании проектов. И стоит вам начать реализовывать закругленные углы с помощью CSS3 (как описано в этой главе), вы никогда не сможете вернуться к старым методам.

Дальше мы рассмотрим еще один прием прогрессивного оформления: использование RGBA-цвета (red, green, blue, alpha).

# Глава 3

## Гибкая работа с цветом при помощи RGBA

**Использование незатейливых форм и простых, удобных инструментов и средств позволяет достигнуть очень высокого уровня мастерства. Так, с помощью цвета и текстуры мы придаем индивидуальность каждому автомобилю.**

Ларри Эриксон, бывший дизайнер компании Ford

Как-то у меня был заказчик (я буду называть его «Стэнли»), который, ознакомившись с идеей дизайна сайта, выдал такой комментарий: «Мне совершенно не нравится этот зеленый цвет. У моей бывшей девушки было такого же гадкого цвета одеяло. Нельзя использовать этот цвет».

Стэнли (как он сам себя называет, «не дизайнер») был определенно недоволен цветом или по крайней мере этим конкретным оттенком. Ни для кого не секрет, что *цвет воздействует на эмоции*. У каждого есть любимый и/или нелюбимый цвет. И нет сомнений в том, что в мире веб-дизайна цвет является важным и мощным (и, конечно, доступным и распространенным) ресурсом.

Ларри Эриксон, бывший дизайнер автомобилей компании Ford, говорит об этом в эпиграфе. Цвета и текстуры достаточно, чтобы подчеркнуть индивидуальность, если их применить к продуманной структуре. Хотя в эпиграфе речь идет о различных моделях автомобилей, я думаю, вы согласитесь, что к веб-дизайну это, в общем-то, тоже относится.

Продолжая переоценивать старые методы и самые лучшие технические приемы, особенно интересно будет поговорить о цвете с точки зрения альфа-прозрачности, а также о гибкости и текстуре, в основе которых она лежит (рис. 3.1).

**ПРИМЕЧАНИЕ**

Цветной вариант этой главы вы найдете на сайте издательства по адресу *www.piter.com*, на странице, посвященной этой книге (ссылка «*Отрывок*»).

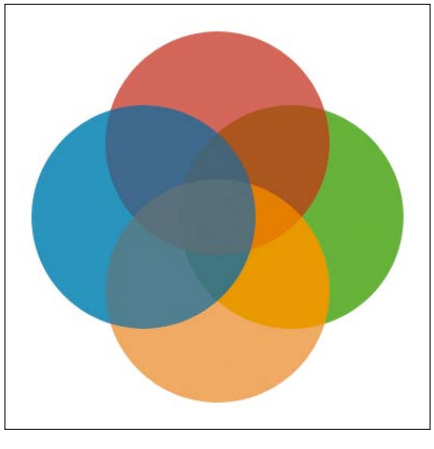


**Рис. 3.1.** Пример полупрозрачных цветов, накладывающихся друг на друга

Когда мы говорим об альфа-прозрачности, мы имеем в виду способность цвета иметь разную степень *непрозрачности*. Благодаря этой дополнительной характеристике средством выражения индивидуальности может быть такое простое свойство как цвет. Наложение полупрозрачных цветов друг на друга открывает нам целый мир визуальной неповторимости, что особенно ценится в веб-интерфейсах.

Раньше мы бы использовали для этого PNG-изображения (которые могут быть полупрозрачными) или, возможно, свойство opacity (свойство CSS3, которое неплохо поддерживается в Firefox 1.5, Safari 1.2 и более поздних версиях, а также в Opera 9 и более поздних версиях).

Сейчас существует новая интересная цветовая модель, добавленная в CSS3 Color Module, которая позволяет задавать одновременно цвет и прозрачность: RGBA. Эта модель дает нам особые преимущества, которые будут рассматриваться в этой главе. Мы также обсудим плюсы и минусы других методов. В процессе мы будем постепенно улучшать наш шаблон Tugboat с помощью RGBA, а также возьмемся за создание раздела This Week's Specials, состоящего из картинок и описаний (рис. 3.2). Но сначала я объясню, как работает RGBA.

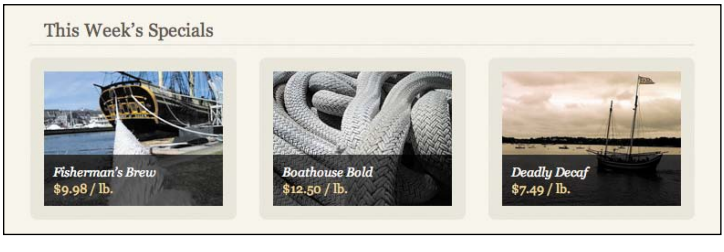



**Рис. 3.2**

## Что такое RGBA?

Сначала давайте поговорим о том, что такое RGBA. RGB (что означает Red, Green, Blue — красный, зеленый, синий) — цветовая модель, позволяющая задавать цвет с помощью численных значений трех составляющих. Комбинируя их, можно получить множество различных оттенков.

На рисунке 3.3 показана палитра цветов Photoshop. Обратите внимание, что выбранный нами голубой цвет может обозначаться несколькими способами, включая знакомую шестнадцатеричную запись, которую мы бы использовали в CSS.

```
body {
   background: #3792b3;
   }
```

Можно задать тот же цвет в RGB, используя три десятичных значения (для красного, зеленого и синего).

```
body {
   background: rgb(55,146,179);
   }
```

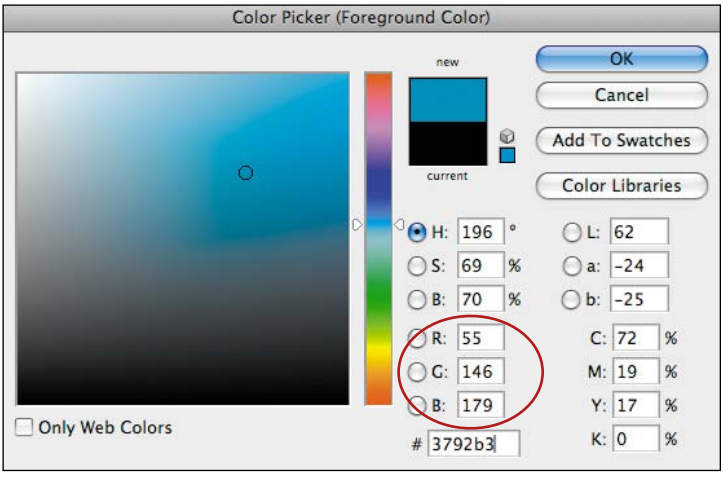



**Рис. 3.3**

В обоих случаях получается один и тот же оттенок, но используется разный синтаксис.

RGBA расшифровывается как Red Green Blue Alpha. На W3C объясняется: «В этой спецификации цветовая модель RGB расширена и включает составляющую альфа, позволяющую задать непрозрачность цвета» (*http://www.w3.org/TR/css3-color/#rgba-color*).

Это значит, что можно добавить четвертое значение (от 1 до 0), чтобы задать уровень непрозрачности данного RGB-цвета. Для полной непрозрачности используется значение 1, для полной прозрачности — 0.

Например, мы можем сделать наш голубой цвет полупрозрачным, добавив .5 в качестве четвертого значения после значений RGB.

```
body {
   background: rgba(55,146,179,.5);
   }
```

**Рис. 3.4**

На рисунке 3.4 показано, как один и тот же оттенок голубого может иметь различную степень непрозрачности, если он задан в RGBA. Ниже приведен соответствующий код; в нем выделены фрагменты, в которых задается цвет (простите, что я использую внутренний стиль, но так проще показать сразу все).

```
<div style="float: left; width: 50px; height:
     50px; background: rgba(55,146,179,1);">
     </div>
<div style="float: left; width: 50px; height:
     50px; background: rgba(55,146,179,.75);">
     </div>
<div style="float: left; width: 50px; height:
     50px; background: rgba(55,146,179,.5);">
     </div>
<div style="float: left; width: 50px; height:
     50px; background: rgba(55,146,179,.25);">
     </div>
<div style="float: left; width: 50px; height:
     50px; background: rgba(55,146,179,.1);">
     </div>
```

Для одного и того же значения RGB можно добиться разной степени непрозрачности: `1` задает непрозрачность 100%, `.75` дает тот же цвет, но с непрозрачностью 75%, `.5` означает 50% и т. д.

Возможность задавать прозрачность цвета легко и быстро прямо в таблице стилей — это прекрасно. Но как насчет собственно свойства `opacity` и чем оно отличается от RGBA?

## Свойство opacity против RGBA

В CSS3 можно добавлять прозрачность с помощью свойства `opacity`. Просто задайте значение от `1` до `0`, чтобы определить степень прозрачности любого элемента.

Например, если вы хотите добиться непрозрачности 65% для всех абзацев на странице, можно написать такой код.

```
p {
   opacity:.65;
   }
```

**ПРИМЕЧАНИЕ**

О различиях между opacity и RGBA можно узнать на http://www.css3.info/*introduction-opacity-rgba/*.

Значительная разница между `opacity` и RGBA заключается в том, что `opacity` задает прозрачность элемента *и всего, что в нем содержится*, тогда как RGBA задает прозрачность *только фона или цвета элемента*.

## КОГДА ПЕРЕСТАЕТ РАБОТАТЬ СВОЙСТВО OPACITY

Пусть, например, у нас есть полупрозрачный голубой блок с текстом, который мы хотим наложить на фон с узором (рис. 3.5).


**Рис. 3.5**

Разметка для этого маленького модуля будет выглядеть примерно так.

```
<div class="coupon">
   <p>Save 50% off all coffee, this week only!
     </p>
</div>
```

Чтобы блок выглядел так, как мы хотим, но без применения прозрачности, можно написать такие основные стили.

```
.coupon {
   padding: 1em;
   background: #3792b3;
   border-radius: 1em;
   -webkit-border-radius: 1em;
   -moz-border-radius: 1em;
   }

.coupon p {
   font-family: Helvetica, sans-serif;
   font-size: 2em;
   font-weight: bold;
   color: #a14141;
   }
```

Обратите внимание, что здесь мы используем закругленные углы, применяя свойство `border-radius`, описанное в главе 2. Вот такие мы молодцы.

### Использование свойства opacity

Если бы мы использовали свойство `opacity`, чтобы сделать блок прозрачным, то нам бы понадобилось добавить соответствующее правило в объявление `.coupon`, которое отвечает за создание голубого блока.

```
.coupon {
   padding: 1em;
   background: #3792b3;
   border-radius: 1em;
   -webkit-border-radius: 1em;
   -moz-border-radius: 1em;
   opacity:.25;
   }
```

На рисунке 3.6 показан результат. Как вы видите, непрозрачным на 25% стал не только голубой блок, но и текст внутри блока. А это уже не то, чего мы хотели.


**Рис. 3.6**

### Использование цветовой модели RGBA

Вместо свойства `opacity` давайте попробуем использовать RGBA. Так мы сможем задать цвет фона и степень прозрачности блока с помощью одного правила.

```
.coupon {
   padding: 1em;
   background: rgba(55,146,179,.25);
   border-radius: 1em;
   -webkit-border-radius: 1em;
   -moz-border-radius: 1em;
   opacity:.25;
   }
```


**Рис. 3.7**

Здесь мы изменили способ задания цвета фона: вместо шестнадцатеричной записи (`#3792b6`) мы использовали ее эквивалент в модели RGB (`55,146,179`). Затем, задав альфа-значение `.25`, мы сделали этот голубой цвет непрозрачным на 25%.

Существенное отличие здесь вот в чем: используя RGBA, мы меняем прозрачность только фона, а не всего блока и его содержимого (как это произошло со свойством `opacity`). Теперь мы выполнили нашу задачу успешно (рис. 3.7).

## Совместимость

Как и свойство `border-radius`, рассмотренное в предыдущей главе, RGBA является удивительно гибким инструментом. Но так как эту модель не поддерживает Internet Explorer, ее однозначно можно считать средством для *прогрессивного оформления*. Сейчас поддержка RGBA есть в движке WebKit (Safari, Chrome, iPhone и т. д.), Firefox 3 и Opera 10. Если вас устраивает тот факт, что эта цветовая модель будет работать только в прогрессивных браузерах, то все в порядке.

| RGBA | |
|---|---|
| WebKit (Safari) | ✓ |
| Firefox 3 | ✓ |
| Opera 10 | ✓ |

Похожим образом поддерживается свойство `opacity`, хотя, по большей части, оно было принято немного раньше, чем RGBA. Но и оно не поддерживается браузером Internet Explorer.

| Свойство opacity | |
|---|---|
| WebKit (Safari 1.2+) | ✓ |
| Firefox 1.5+ | ✓ |
| Opera 9+ | ✓ |

Более подробно об этом говорится на *http://dev.opera.com/articles/view/css-and-opacity-methods-for-creating-tr/*.

## А как насчет других браузеров?

RGBA — необычайно гибкое средство, позволяющее задавать прозрачность цвета с помощью одной простой строки кода. И здесь вы наверняка вспомните тезис из главы 2: возможность управлять элементами дизайна через таблицы стилей вместо того, чтобы редактировать изображения, не только облегчает жизнь, но и позволяет работать быстрее и эффективнее — браузер берет на себя всю тяжелую работу. И, опять же, это перспективное направление в CSS, поэтому мы можем уже сейчас начать экспериментировать с новыми возможностями в браузерах, поддерживающих продвинутые стили (такие как RGBA).

К счастью, использование синтаксиса цветовой модели RGBA безопасно для браузеров, которые ее не поддерживают. Например, Internet Explorer просто проигнорирует такое правило.

### Создание резервных правил для неполноценных браузеров

Поскольку такие браузеры, как Internet Explorer, игнорируют правила с RGBA, неплохо было бы задать «резервный» цвет до объявления цвета RGBA.

Так, если в нашем примере мы хотим добавить чистый голубой цвет для браузеров, не поддерживающих RGBA, мы должны расположить стили в таком порядке.

```
.coupon {
   padding: 1em;
   background: #c4dada; /* для IE */
   background: rgba(55,146,179,.25); /* для
     браузеров, поддерживающих RGBA */
   border-radius: 1em;
   -webkit-border-radius: 1em;
   -moz-border-radius: 1em;
   }
```

На рисунке 3.8 показано, как будет выглядеть наш блок в Internet Explorer или другом браузере, не поддерживающем RGBA (а также `border-radius`). Чистый голубой цвет,

который мы используем в качестве резервного варианта, задан первым; далее мы переопределяем его с помощью цвета RGBA с непрозрачностью 25% для браузеров, которые поддерживают эту модель. Вот это и есть настоящее прогрессивное оформление!


**Рис. 3.8.** Как может выглядеть наш пример в браузерах, не поддерживающих RGBA и border-radius

Хотя элементы вашего дизайна потеряют прозрачность в браузерах, не поддерживающих RGBA, вы все же будете полностью управлять цветами внутри таблицы стилей. Причудливые прозрачные элементы — для прогрессивных браузеров, непрозрачный запасной вариант (но функциональный и читаемый) — для остальных.

## ЗАПОЛНЕНИЕ PNG-ИЗОБРАЖЕНИЯМИ

Еще один вариант (который будет работать в большинстве браузеров) — это заполнение PNG-изображениями. Этот формат поддерживает альфа-канал, и поэтому PNG-изображение можно было бы использовать в качестве фона при создании такого полупрозрачного блока, какой мы получили с помощью RGBA.

Например, если мы создадим небольшое PNG-изображение, добавив слой нашего голубого цвета с непрозрачностью 25% (рис. 3.9), мы сможем заполнить им фон в блоке. Эффект будет таким же.

```
.coupon {
   padding: 1em;
   background: url(../img/bg.png);
   border-radius: 1em;
   -webkit-border-radius: 1em;
   -moz-border-radius: 1em;
   }
```

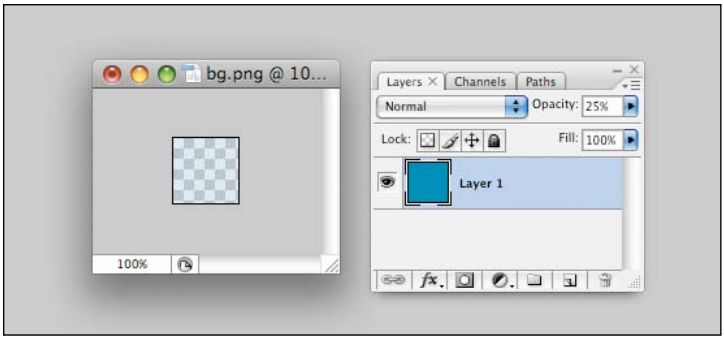



**Рис. 3.9.** Создание полупрозрачного PNG-изображения, предназначенного для заполнения фона, в Photoshop

PNG-изображения с альфа-прозрачностью отображаются правильно в Safari, Firefox и Opera (так же, как и RGBA). Но, кроме того, они работают в Internet Explorer версий 7 и 8. В версии 6 есть поддержка PNG-формата, но нет поддержки альфа-канала, так что ваши полупрозрачные PNG-изображения будут абсолютно непрозрачными в IE6.

Существуют различные приемы, позволяющие добиться того, чтобы альфа-канал поддерживался в IE6. В некоторых используется JavaScript, в некоторых — специфическое для Microsoft CSS-свойство `filter` (если PNG-изображение задано в качестве фона).

### Проблемы с цветом и степенью непрозрачности

И снова, как и в случае с закругленными углами, использование PNG-изображений для создания полупрозрачного фона связано с тем недостатком, что цвет и степень прозрачности задаются при создании изображения. А если в таких ситуациях использовать RGBA, то можно быстро изменять оттенок и степень прозрачности прямо в таблице стилей. И не нужно будет создавать новое изображение каждый раз, когда понадобится задать новый цвет или отрегулировать прозрачность. Этот факт важно учитывать, когда вы решаете, какой метод выбрать. Если вы не против того, что для браузеров, не поддерживающих RGBA, придется задавать непрозрачный цвет (или какой-либо другой запасной вариант), обязательно начинайте экспериментировать.

## Превосходный инструмент для создания прототипа

Я уже приводил этот аргумент в главе 2, когда мы говорили о свойстве `border-radius`, и сейчас, при обсуждении RGBA, он тоже уместен. Независимо от того, используете ли вы RGBA при создании проекта, эта модель может быть ценным инструментом для создания прототипов интерфейсов. То есть если при первоначальной разработке проекта вы пользуетесь Safari, Firefox или даже Opera, модель RGBA очень удобна для выполнения однотипных операций с цветом и прозрачностью. Изменения и корректировки можно вносить в код быстро и безболезненно.

А уже потом вы сможете выбрать подходящий метод окончательной реализации: возможно, это будет RGBA и резервные правила для других браузеров или же PNG-изображения, работающие также в IE7+.

И, опять же, из этой главы стоит сделать такой вывод: уже сейчас можно экспериментировать с RGBA. И можно даже использовать эту модель при создании проекта, если вы являетесь сторонником идеи прогрессивного оформления.

## Не только фон

Еще одно исключительное свойство RGBA заключается в том, что эту модель можно применять не только к фоновым изображениям. С ее помощью можно задавать цвет и прозрачность *гипертекста*. Это открывает нам массу новых возможностей, а также элегантных и гибких решений.

Я, к примеру, использовал RGBA, чтобы уменьшить непрозрачность фрагмента гиперссылки на сайте моей студии (*http://simplebits.com/*).

### Уменьшение насыщенности шрифта с помощью RGBA

На рисунке 3.10 показано несколько ссылок, и у каждой ссылки есть заголовок (написанный полужирным шрифтом) и подзаголовок (написанный обычным шрифтом). Я могу выбрать в качестве основного цвета для ссылок зеленый, а потом немного уменьшить насыщенность шрифта, задав непрозрачность для этого же цвета с помощью RGBA.

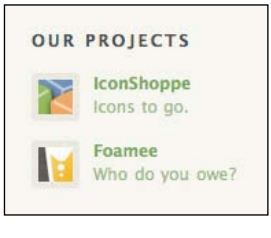



**Рис. 3.10**

Иными словами, вместо двух цветов для ссылки я выберу один и с помощью RGBA сделаю его светлее, чтобы он стал ближе к цвету фона и отличался от более насыщенного цвета.

Для этих ссылок в разметке используется неупорядоченный список. При этом в тег `<a>` заключены изображение, заголовок (выделенный в отдельный элемент `<strong>`) и подзаголовок.

```
<ul class="lst group">
   <li>
      <a href="http://iconshoppe.com/">
         <img src="/img/ishoppe-25.gif"
     alt="icon" />
         <strong>IconShoppe</strong> Icons to go.
      </a>
   </li>
   <li>
      <a href="http://foamee.com/">
         <img src="/img/foamee-25.gif" alt="icon"
         />
         <strong>Foamee</strong> Who do you owe?
      </a>
   </li>
</ul>
```

Таблицы стилей, в которых задается цвет ссылки, будут выглядеть примерно так.

```
ul. lst li a {
   color: #76a65c; /* для всех браузеров */
   color: rgba(118,166,92,.75); /* для браузеров,
     поддерживающих RGBA */
   }

ul. lst li a strong {
   display: block;
   color: #76a65c; /* переопределение для браузеров,
     поддерживающих RGBA */
   }
```

Как видите, для всей ссылки мы задаем значение RGBA с непрозрачностью 75%, а затем переопределяем его значение (в шестнадцатеричной записи) для элемента `<strong>`, используя тот же цвет, но повышая непрозрачность до 100%. Кроме того, мы добавили строку `display: block;`, чтобы заголовок располагался на отдельной строке.

Этот метод освобождает нас от необходимости подбирать разные оттенки одного и того же цвета. Остается выбрать один основной цвет, а потом с помощью RGBA менять значения его непрозрачности.

Браузеры, не поддерживающие RGBA, проигнорируют одноименное правило, и в результате для ссылок будет использоваться основной цвет, заданный правилом `color: #76a65c;` (рис. 3.11).


**Рис. 3.11**

## WILSONMINER.COM

Похожим образом RGBA использует на своем сайте Вилсон Майнер (рис. 3.12): он уменьшает непрозрачность черного текста, приближая его оттенок к цвету фона (*http://www.wilsonminer.com/*).

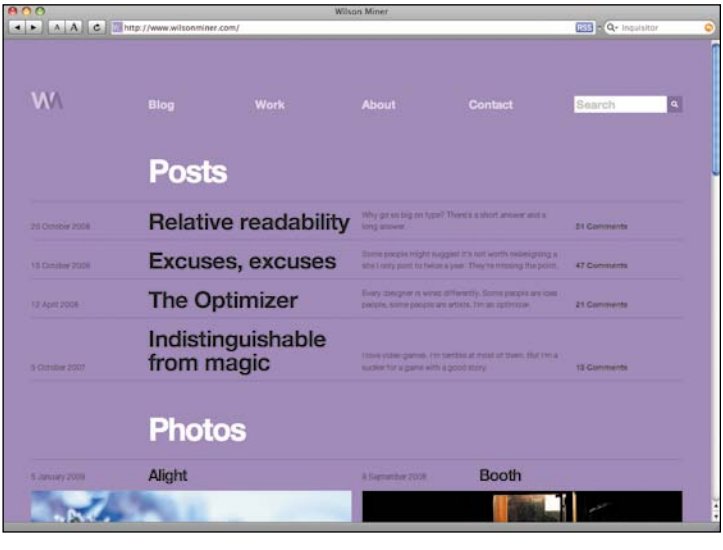



**Рис. 3.12.** Вилсон Майнер искусно применяет RGBA для задания цвета текста на своем сайте

В одной из записей блога (*http://www.wilsonminer.com/posts/2008/oct/15/excuses-excuses/*) он объясняет:

> *Я хотел немного расширить проект, но при этом добиться гибкости дизайна. Поэтому я сделал так, чтобы можно было легко изменять цвета и фоновые изображения для фрейма сайта и домашней страницы. Я начал с классического зеленого цвета, но буду время от времени обновлять оформление.*
>
> *На домашней странице, в частности, значения RGBA используются для придания прозрачности тексту. Благодаря*

*этому, к цвету текста добавляется оттенок фона, каким бы он ни был (чистым цветом или изображением). Даже «черный» текст на странице сделан полупрозрачным, и потому содержит частичку фонового цвета. Поэтому контраст становится более мягким (в полиграфии похожий эффект получается при последовательном наложении красок).*

Так что, хотя на этом сайте почти весь текст черный, Вилсон с помощью RGBA смягчает этот цвет, и текст немного сливается с фоном (независимо от того, какого цвета фон) (рис. 3.13). Можно задать другой цвет фона, не меняя цвета текста и степеней прозрачности. В полупрозрачном тексте в любом случае будет содержаться оттенок фона. Вот это и есть гибкий цвет.

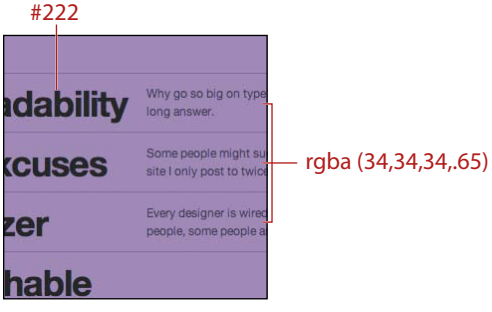



**Рис. 3.13**

## КАК ЗАПРОСТО ДОБАВИТЬ HOVER-ЭФФЕКТ

У RGBA есть еще одно преимущество: можно легко задавать изменение цвета ссылки при наведении курсора мыши. Добиться hover-эффекта можно, всего лишь уменьшая непрозрачность ссылки, и не нужно выбирать никакой другой цвет. При этом дизайн будет гибким: можно без проблем изменить цвет фона страницы или основной цвет ссылки.

Я решил реализовать эту идею в шаблоне Tugboat. Сделаем так, чтобы цвет при наведении становился полупрозрачным вариантом основного цвета ссылки (рис. 3.14).

```
a: link, a: visited {
   font-weight: bold;
   text-decoration: none;
   outline: none;
   color: #3792b3;
   }
```

```
a: hover {
   color: rgba (55,146,179,.65);
   }
```

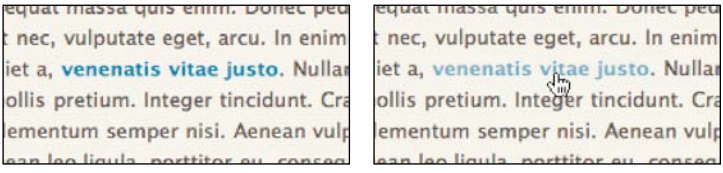



**Рис. 3.14.** Обычный цвет ссылки и цвет при наведении (в увеличенном масштабе)

## РАСКРАСКА ПО НОМЕРАМ

Такое смешивание цветов напомнило мне об одном интереснейшем интервью с Артом Лоцци, который был художником студии «Хана-Барбера» в далекие 60 годы. Джон Крисфалуси (создатель скандально известного мультипликационного сериала «Рен и Стимпи») рассуждает в своем блоге о работе Арта (пример приведен на рисунке 3.15), анализируя фоновые рисунки к мультфильму «Мишка Йоги» (полностью интервью можно прочитать на *http://johnkstuff.blogspot.com/2006/12/color-theory-art-lozzi-explains-some.html*).

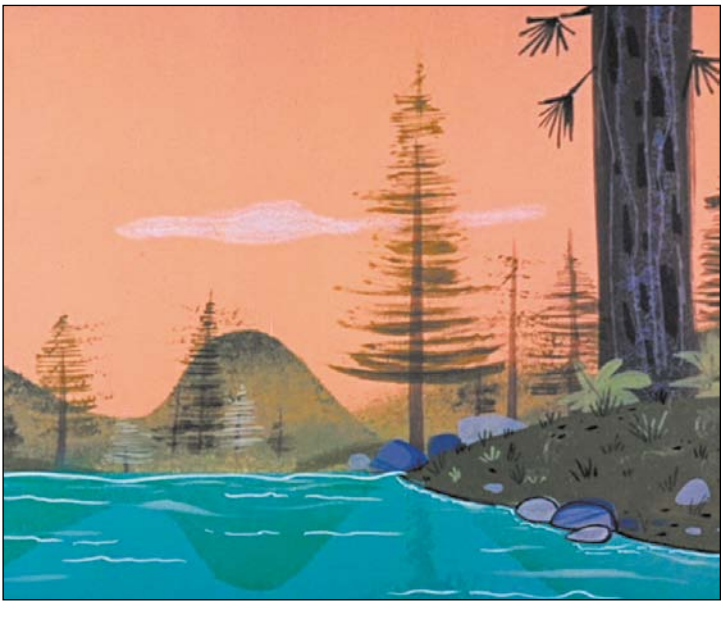


**Рис. 3.15.** Один из великолепных фоновых рисунков Арта Лоцци

Лоцци рассказывает, как они вначале раскрашивали весь фон одним цветом:

> *Этот цвет наносится на толстенный бристольский картон (не обычный картон и не холст), а уже поверх него рисуется все остальное; и если появляются какие-то «дыры» или промежутки, то цвет неба не позволяет целостности рисунка нарушиться.*

В ответ Крисфалуси излагает свой взгляд на метод Лоцци:

> *Через маленькие просветы в холмах, не до конца прорисованных губкой, проглядывает персиковое небо. Создается эффект смешивания основного фонового цвета с остальными цветами фона, который и объединяет все это в гармоничную цветовую схему.*
>
> *Точно так же цвет холма виден сквозь деревья, и так далее.*
>
> *В результате такого смешивания все элементы рисунка становятся частью цветовой системы, а не группой отдельных объектов (каждый из которых состоит из совершенно разных цветов), нарушающих целостность изображения и конкурирующих между собой.*

Смешивать цвета для создания гармоничной палитры. Это то, что Арт делал десятилетия назад с помощью краски и кистей, и то, для чего Вилсон Майнер использовал RGBA, когда он хотел добиться удачного сочетания цвета текста и фона в Сети. Мастер использует все средства!

## Создание раздела This Week's Specials

Теперь, когда мы хорошо знаем, что такое RGBA и как работать с этой моделью, а также имеем представление о других способах задания прозрачности в Сети, давайте вернемся к шаблону Tugboat и проанализируем раздел This Week's Specials. Потом мы обсудим способ его реализации, где будет использоваться RGBA для создания изображений, накладываемых на каждую из фотографий (рис. 3.16).


**Рис. 3.16**

На каждую фотографию накладывается полупрозрачное изображение, на котором расположено название и цена. Это обычное дизайнерское решение, гибкий способ добавлять метаинформацию к фотографиям, не зная заранее их цвета/яркости/контрастности. Чтобы не нарушалась удобочитаемость текста, используется полупрозрачное фоновое изображение, позволяющее создать удачный дизайн вне зависимости

от фотографии, на которую оно накладывается. Идеальное место, чтобы поиграть с RGBA.

## СОЗДАНИЕ РАЗМЕТКИ

Начнем с разметки. Будем использовать для трех наших напитков недели упорядоченный список, где для каждого элемента мы хотим, чтобы само изображение, название и цена были активными. Поэтому мы поместим каждый элемент списка в тег `<a>`, используя строковые элементы для выделения названия и цены (так как блоковые элементы нельзя располагать внутри гиперссылки).

Я также добавил несколько дополнительных элементов. Для чего они нужны, станет понятно на завершающей стадии, после создания стилей.

```
<ol class="specials">
   <li>
      <div class="special">
         <div class="special-img">
            <a href="/specials/fb">
               <img src="img/boat.jpg" alt="ship"
               />
               <span>
                  <strong>Fisherman&#8217;s
                     Brew</strong>
                  <em>$9.98 / lb.</em>
               </span>
            </a>
         </div>
      </div>
   </li>
   <li>
      <div class="special">
         <div class="special-img">
            <a href="/specials/bb">
              <img src="img/ropes.jpg" alt="ropes" />
               <span>
                  <strong>Boathouse Bold</strong>
                  <em>$12.50 / lb.</em>
               </span>
            </a>
         </div>
```

```
        </div>
    </li>
    <li>
        <div class="special">
            <div class="special-img">
                <a href="/specials/dd">
                    <img src="img/fame.jpg"
                        alt="sailboat" />
                    <span>
                        <strong>Deadly Decaf</strong>
                        <em>$7.49 / lb.</em>
                    </span>
                </a>
            </div>
        </div>
    </li>
</ol>
```

## КАК СДЕЛАТЬ СПИСОК ГОРИЗОНТАЛЬНЫМ

Первые несколько правил таблицы стилей задают ширину каждого элемента списка и его расположение вдоль левого края, чтобы превратить вертикальный список в горизонтальный.

```
ol.specials li {
    width: 210px;
    float: left;
    margin: 0 15px 1.5em 0;
    }
```

**ПРИМЕЧАНИЕ**

Как мы уже говорили во введении, применение таких стилей требует добавления таблицы стилей *reset. css*, которая обнуляет стандартные умолчания браузера.

На рисунке 3.17 показан результат добавления этого стиля: каждое изображение является ссылкой, как и расположенные ниже название и цена. Ширина каждой картинки — 210 пикселов; для каждого элемента мы также добавили небольшие поля справа и снизу.


**Рис. 3.17**

## ДОБАВЛЕНИЕ ГРАНИЦЫ С ЗАКРУГЛЕННЫМИ УГЛАМИ

Применяя на деле то, что мы узнали из главы 2, давайте добавим к каждому элементу границу шириной 15 пикселов и сделаем ее углы закругленными с помощью свойств `border-radius`, которые мы знаем и любим.

Для этого нам нужно увеличить ширину каждого элемента списка на 30 пикселов.

```
ol.specials li {
   width: 240px;
   float: left;
   margin: 0 15px 1.5em 0;
   }

ol.specials li div. special {
   position: relative;
   border: 15px solid #e2e1d4;
   border-radius: 8px;
   -webkit-border-radius: 8px;
   -moz-border-radius: 8px;
   }
```

**ПРИМЕЧАНИЕ**

Я раньше этого не говорил, но в шаблоне Tugboat используется «резиновая» верстка. Однако в этой конкретной ситуации я для простоты использую значения ширины в пикселах. Оставайтесь с нами, и в главе 6 я объясню, как эти стили можно применить в случае «резиновой» верстки.

Также к каждому элементу списка мы добавили правило `position: relative;`, чтобы в будущем можно было поместить название и цену на фотографию. На рисунке 3.18 показан наш список с добавленной закругленной границей, и теперь мы готовы приступить к наложению изображения.


**Рис. 3.18**

## ДОБАВЛЕНИЕ НАКЛАДЫВАЕМОГО ИЗОБРАЖЕНИЯ

Теперь давайте добавим накладываемое изображение с названием и ценой. Так как мы хотим, чтобы вся картинка тоже была активной, воспользуемся элементом `<span>`, который содержит цену и название. Сделаем его блоковым и поместим поверх фотографии, закрепив у нижнего края.

Следующее объявление берет на себя кучу всего, включая позиционирование и задание основных стилей шрифта для названия и цены. Сейчас мы также зададим темно-серый фон для накладываемого изображения.

```
ol.specials li div.special a span {
   display: block;
   position: absolute;
   width: 100%;
   bottom: 0;
   left: 0;
   font-family: Georgia, serif;
   font-size: 1.1em;
   font-weight: normal;
   line-height: 1.3em;
   color: #ccc;
   background: #333;
   }
```

На рисунке 3.19 показано текущее состояние дел. Вы наверняка заметите, что нужно еще добавить отступы и подправить оформление текста названия и цены, чтобы все выглядело так, как надо.


**Рис. 3.19**

## СТИЛИ НАЗВАНИЯ И ЦЕНЫ

Теперь перейдем к шрифтовому оформлению и расположению названия и цены. Для этого будем использовать два новых объявления, в которых зададим отступ вокруг текста и изменим цвет. Также добавим `display: block;` (так как по умолчанию `<strong>` и `<em>` являются строковыми элементами), после чего название и цена будут располагаться на отдельных строках.

```
ol.specials li div.special a span strong {
   display: block;
   padding: 10px 10px 0 10px;
   font-weight: normal;
   font-style: italic;
   color: #fff;
   }

ol.specials li div.special a span em {
   display: block;
   padding: 0 10px 10px 10px;
   font-style: normal;
   color: #e3c887;
   }
```

На рисунке 3.20 показан результат добавления этих двух объявлений. Ура, мы почти закончили!


**Рис. 3.20**

## ДОБАВЛЕНИЕ RGBA К НАКЛАДЫВАЕМОМУ ИЗОБРАЖЕНИЮ

На завершающем этапе добавим значение RGBA, чтобы сделать накладываемое изображение прозрачным: пусть оно будет черным с непрозрачностью 70%.

```
ol. specials li div. special a span {
   display: block;
   position: absolute;
   width: 100%;
   bottom: 0;
   left: 0;
   font-family: Georgia, serif;
   font-size: 1.1em;
   font-weight: normal;
   line-height: 1.3em;
   color: #ccc;
   background: #333;
   background: rgba(0,0,0,.7);
   }
```

Мы сохраним предыдущее значение `#333` для браузеров, не поддерживающих RGBA, и при этом текст останется разборчивым независимо от фотографии. Но мы все же переопределим этот цвет, задав значение RGBA: черный с непрозрачностью 70%. На рисунке 3.21 показан законченный список, в котором мы имеем дело с приятным гибким стилем накладываемого изображения для каждого напитка.


**Рис. 3.21**

### Внесение изменений — быстро и просто

Регулировать цвет и степень прозрачности накладываемого изображения теперь очень просто: для этого нужно исправить соответствующие правила в CSS. К примеру, если мы хотим

сделать это изображение красным, можно быстро и просто изменить значение RGB (рис. 3.22).

```
ol.specials li div.special a span {
   display: block;
   position: absolute;
   width: 100%;
   bottom: 0;
   left: 0;
   font-family: Georgia, serif;
   font-size: 1.1em;
   font-weight: normal;
   line-height: 1.3em;
   color: #ccc;
   background: #600;
   background: rgba(102,0,0,.7);
   }
```


**Рис. 3.22**

## В ЗАКЛЮЧЕНИЕ

Сейчас мы лишь поверхностно изучили возможности RGBA. Однако я надеюсь, что примеры из этой главы станут той искрой, из которой разгорится пламя, и вы будете и дальше пробовать добавлять прозрачность к элементам вашего дизайна. Возможность задавать непрозрачность одновременно с цветом — гибкий и мощный инструмент, а RGBA — всего лишь один из способов добиться этого. Но модель RGBA также позволяет сделать все быстро и легко, и за это мы ее любим. Как и в случае со свойством `border-radius`, в настоящее время ее поддерживает ограниченное число браузеров. Однако поскольку WebKit, Mozilla и Opera обязательно поведут за собой остальные браузеры, этот метод можно проверять и использовать уже сейчас, и не важно, делаете вы это для создания прототипа и итеративной разработки или же хотите внести что-то новое в создание массовых проектов.

После двух обычных глав о свойствах CSS3 и новых технических приемах, которые неодинаково отображаются в разных браузерах, пришло время остановиться и кое-что выяснить. А нужно ли вообще, чтобы веб-сайт выглядел одинаково во всех браузерах? В следующей главе мы поговорим как раз об этом, а также рассмотрим несколько забавных приемов прогрессивного оформления.

# Глава 4

## Должны ли сайты выглядеть одинаково во всех браузерах?

**Я всегда считал себя человеком, готовым выложиться на 80 процентов. Мне нравится увлеченно приниматься за какой-нибудь вид спорта или другой род деятельности и заниматься им до тех пор, пока я не буду профессионалом на 80 процентов. Чтобы выйти за эти рамки, нужна такая одержимость и такой уровень подготовки, которые меня совсем не привлекают.**

Ивон Шуинар, основатель и владелец компании Patagonia.
«Позвольте моим людям заняться серфингом»

Вам случалось когда-нибудь говорить (или работать) с человеком, который на все 100% поглощен каким-либо делом? Опасность в том, что такие люди попросту вязнут в мелочах. Докапываются до *каждой* мелочи. С другой стороны, «человек, готовый выложиться на 80 процентов» (по выражению основателя компании Patagonia Ивона Шуинара) может со временем научиться понимать, каким вещам нужно уделить особое внимание. Умение определять, *какие* мелочи наиболее важны и полезны, может быть настолько же ценным, как и знание всех этих мелочей.

Я, как и Ивон, отношу себя к числу «тех, кто готов выложиться на 80 процентов». Но это не значит, что я ленив; просто я допускаю, что можно зациклиться на таких мелочах, которые не будут *объективно* важными.

Если применить это рассуждение к веб-дизайну, можно легко помешаться на мелочах. Когда-то давно я гордился тем, что могу реализовывать дизайнерские проекты даже на уровне пикселов в любом браузере, который только удастся найти. Я был не один такой, конечно же. В далекие времена, когда для верстки использовались вложенные таблицы, а в качестве разделителей — GIF-изображения, доступность, «находимость», гибкость и прочие *-ости* в расчет не принимались. Главным считалось то, что дизайн должен выглядеть *одинаково*, и не важно, как именно.

С тех пор мы прошли долгий путь. И браузеры тоже: скорость, с которой они внедряют стандарты, стремительно возросла. В последних двух главах мы говорили о приятных возможностях CSS3, которые уже вводятся в обращение некоторыми браузерами, хотя спецификация еще не до конца разработана. Ускоренное внедрение стандартов — это замечательно: мы получаем рабочий инструментарий и можем пользоваться техническими приемами, облегчающими нашу жизнь. Однако не все браузеры следуют этому принципу, и поэтому нам нужно внимательно относиться к выбору того, что мы будем использовать.

Но важнее всего немного изменить общее представление: это нормально, если дизайн выглядит в одном браузере чуть-чуть иначе, чем в другом. Как только дизайнеры и руководители примут это и начнут *использовать*, идеи прогрессивного оформления будут задействованы в полную силу.

Итак, должны ли сайты выглядеть одинаково во всех браузерах? Давайте это выясним.

## Ответ, который я считаю правильным

Когда я зарегистрировал домен dowebsitesneedtolookexactlythesameineverybrowser.com, я решил изобразить ответ на этот вопрос просто и незатейливо (рис. 4.1). Этот забавный и крайне прмитивный сайт действительно выглядит неодинаково в разных браузерах (подробнее об этом чуть позже).


**Рис. 4.1**

Теперь, когда мы ответили на вопрос, вынесенный в название этой главы, я думаю, можно двигаться дальше.

Нет? Ах, если бы все было так просто…

## «Это бонус» против «Дизайн испорчен»

Ранее мы говорили о добавлении закругленных углов и прозрачных элементов с помощью прогрессивной технологии CSS3, и ни то, ни другое не работало в Internet Explorer. Давайте сравним, как выглядит наш шаблон Tugboat в Safari под Mac (рис. 4.2) и в Internet Explorer 8 под Windows (рис. 4.3).

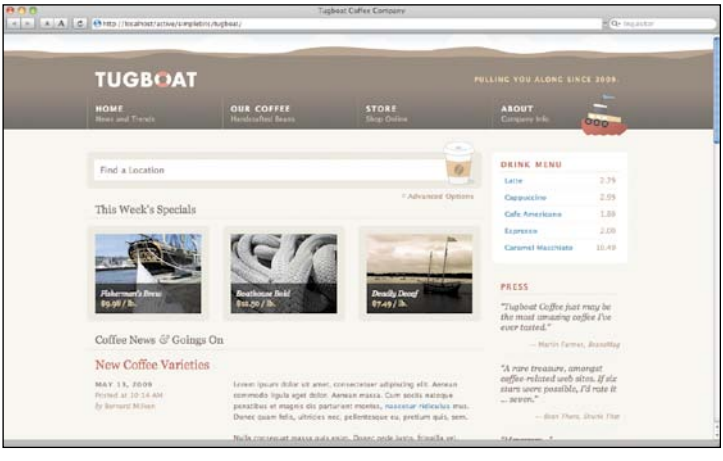



**Рис. 4.2**

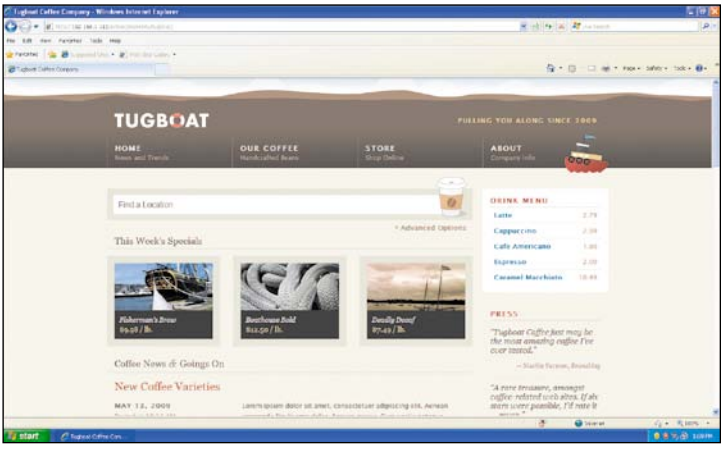



**Рис. 4.3**

Верстка, цвет, шрифтовое оформление и пропорции одинаково отображаются в этих двух браузерах. Однако углы, закругленные в Safari, выглядят прямыми в Internet Explorer 8 (IE8).

Изображения, наложенные на фотографии, в Safari являются полупрозрачными, а в IE8 — просто черными. Для дизайна, удобочитаемости и функциональности сайта Tugboat эти детали не являются принципиально важными, так что в данном случае все абсолютно нормально.

Визуальные особенности следует рассматривать как *бонусы* для браузеров, поддерживающих прогрессивный код, который их создает, а не как недостаток, присущий браузерам, не поддерживающих такой код. Это существенное изменение в понимании вопроса: дизайн не испорчен в IE8; мы не из лени отказываемся уравнять визуальное оформление во всех браузерах. Мы даем пользователям прогрессивных браузеров визуальные бонусы, а качество оформления в остальных снижается до приемлемого уровня. Это суть того, что я называю *прогрессивным оформлением*.

## Все решают руководители

Тот факт, что дизайн может выглядеть немного неодинаково в разных браузерах, устраивает не всех. Если руководитель группы не в числе сторонников такого подхода, идея прогрессивного оформления становится непопулярной. Но это не значит, что мы не должны о ней говорить.

Так, убеждать руководителей в том, что реализация закругленных углов с помощью CSS3 гибче и эффективнее, зачастую нелегко, и успех зависит от усилий всей группы.

Возьмем для примера популярное приложение для обмена сообщениями Твиттер (*http://twitter.com*), которое позволяет отправлять через Сеть короткие обновления статусов или текстовые заметки.

Если вы сравните мою страничку на Твиттере в двух современных браузерах, то увидите, что в Safari у панели навигации и основного контейнера углы закруглены (рис. 4.4), а в IE8 — нет (рис. 4.5).

Для создания закругленных углов в Твиттере используется свойство `border-radius`, которые мы обсуждали в главе 2, и руководители проекта нормально относятся к тому, что оформление немного отличается в разных браузерах.

В Твиттере имеет смысл использовать CSS3 для закругления углов, так как цвет фона и фоновое изображение (в моем случае скриншот из популярной игры Pitfall), а также цвет фона

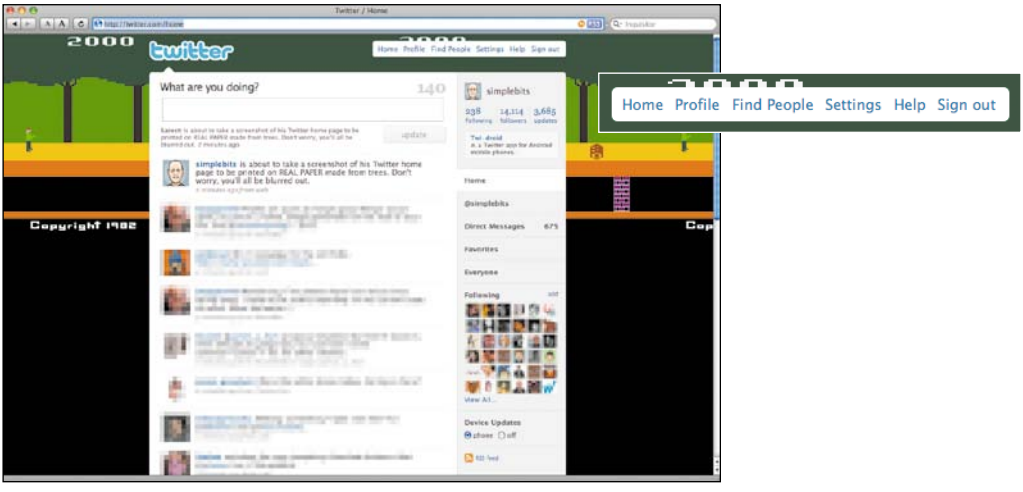



**Рис. 4.4.** Твиттер в браузере Safari: углы различных элементов страницы закруглены благодаря свойству border-radius

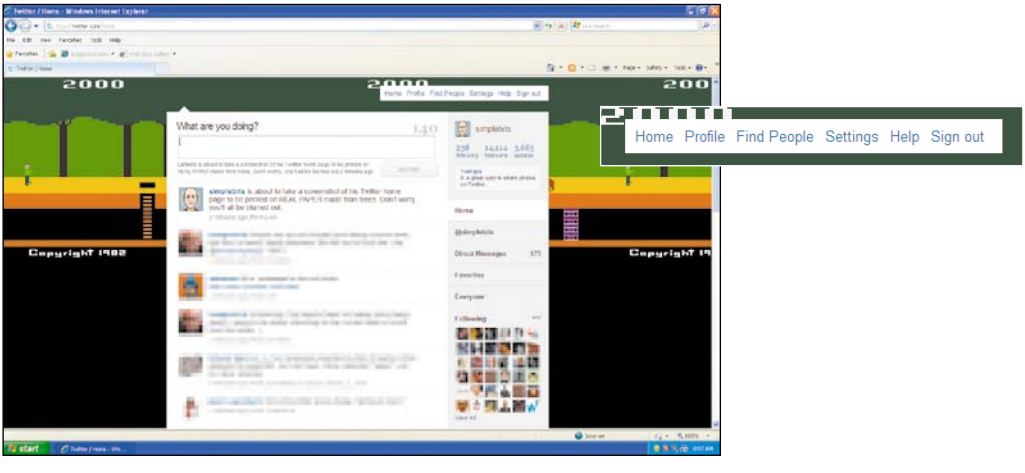



**Рис. 4.5.** Твиттер в браузере IE8: оформление немного ухудшается, но остается достаточно приятным. Углы панели навигации и основного контейнера не закруглены

боковой панели настраиваются пользователем. Именно благодаря CSS3 это стало возможным без добавления дополнительных строк в разметку и без написания каких-либо скриптов. Для Твиттера очень важны *скорость* и *пропускная способность*, а закругленные углы в браузерах на движках Mozilla и WebKit — скорее визуальный бонус, чем обязательное требование.

Все это находится в пределах допустимого, поскольку отсутствие закругленных углов не влияет на верстку, удобочитаемость и функциональность сайта.

## ВСЕ СТАНОВИТСЯ ПРОЩЕ, ЕСЛИ РУКОВОДИТЕЛЬ — ВЫ

Все эти прогрессивные решения использовать гораздо проще, если *вы* являетесь руководителем. Иными словами, при создании ваших собственных проектов или если вам посчастливилось стать тем, кто принимает решения по вопросам реализации проекта, применение принципов прогрессивного оформления становится все более заманчивым вариантом. И быстро входит в привычку.

Например, я создал бесполезный до безобразия сайт под названием Foamee (*http://foamee.com*), предназначенный для пересылки долговых расписок за пиво и кофе через Твиттер (рис. 4.6). Справа и слева от текста заголовка каждой страницы я решил поместить декоративное украшение.

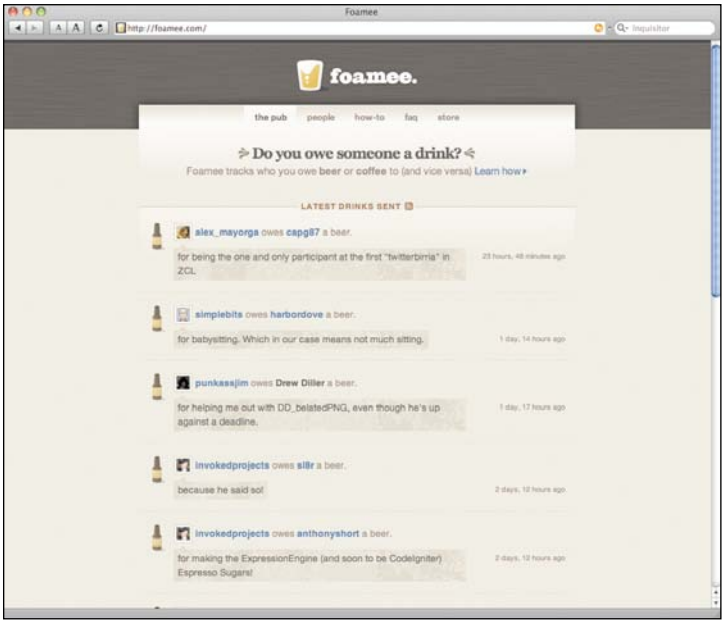



**Рис. 4.6**

Я мог бы добавить несколько строк в разметку и воспользоваться хитростями CSS, чтобы выровнять текст по центру, закрепив украшения у обоих концов текста независимо от его

длины (задача, которую вроде бы действительно просто реализовать, но которая уже не будет одинаково выполняться во всех браузерах). Вместо этого я выбрал *генерацию контента* с использованием CSS, чтобы радикально все упростить и при этом сохранить гибкость.

Я могу сделать разметку еще проще, используя всего лишь один элемент `<h1>`.

```
<h1>simplebits</h1>
```

А чтобы добавить украшения непосредственно слева и справа от текста, можно использовать псевдоэлементы `:before` и `:after`.

```
h1: before {
   content: url(../img/ornament-left.gif);
   }
h1: after {
   content: url(../img/ornament-right.gif);
   }
```

На рисунке 4.7 показано, как будет выглядеть заголовок в браузерах, поддерживающих псевдоэлементы `:before` и `:after`. Украшения расположены правильно: они плотно прилегают к элементу `<h1>`. Однако IE6 и 7 *не* поддерживают генерацию контента и, следовательно, полностью игнорируют эти объявления. Видимо, именно поэтому такое удобное решение не особенно распространено.


**Рис. 4.7**

На рисунке 4.8 показано, как будет отображен заголовок в тех случаях, когда украшения не появляются (как, например, в IE6 или IE7). Ничего страшного! И так как *я* был здесь руководителем, найти оптимальное решение было легко. Заголовок остался на месте, сохранились удобочитаемость и стилевое оформление. Единственное отличие — отсутствие украшений. И при этом мне удалось сделать разметку и стили простыми, гибкими и пуленепробиваемыми.


**Рис. 4.8**

### Буря оваций новому Internet Explorer 8

Есть еще одна хорошая новость: IE8 поддерживает псевдоэлементы `:before` и `:after`, что дает нам дополнительный аргумент в пользу таких решений. Если вы можете смириться с тем, что IE6 и IE7 выглядят немного по-другому, возможно, теперь пришло время пересмотреть отношение к использованию этих ранее не поддерживаемых селекторов.

Это относится к «переоценке старых методов», о которой я говорил во введении. Мы привыкли избегать определенных CSS-селекторов и свойств, так как они не поддерживались в браузерах, существовавших на тот момент. Но желательно время от времени проверять, не изменилась ли ситуация.

Так, например, псевдоселекторы `:before` и `:after` долгое время не поддерживались в браузерах Internet Explorer, но поскольку после выхода IE8 это уже не так, мы можем учитывать этот факт при выборе решений. Действительно ли для вашей задачи лучше использовать простые правила CSS для новейших браузеров, лишая старые версии IE элементов декора? В большинстве случаев я ответил бы «да».

### Не только принять, но и использовать недостатки браузеров

Дизайнер и писатель Энди Кларк использует радикальный и отчасти забавный подход к разграничению IE6 и других браузеров. На рисунке 4.9 показано, как его прекрасный сайт выглядит в Safari (слева) и IE6 (справа). Энди преподносит пользователям IE6 черно-белый вариант своего сайта, призывая обновить браузер, чтобы видеть полноценный цветной вариант.

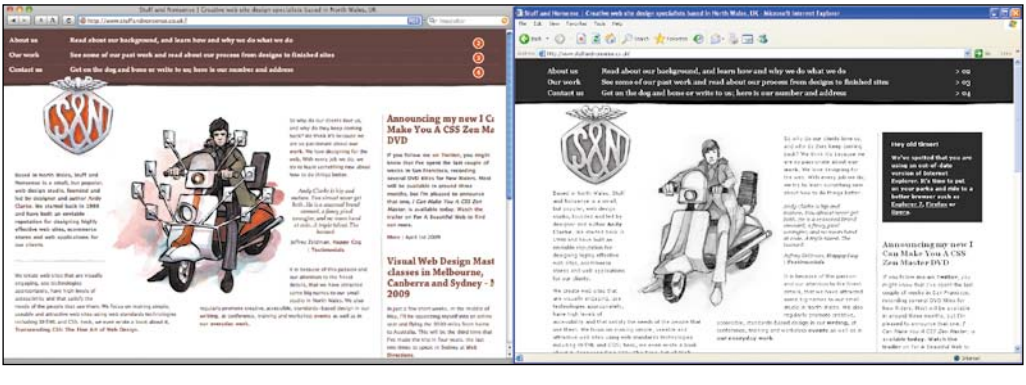


**Рис. 4.9**

Так Энди Кларк не только отвечает на вопрос «Должны ли сайты выглядеть одинаково во всех браузерах?», но и сознательно использует это различие. Будет ли это работать в любой ситуации? Нет. Но кое-что поможет вам понять, как далеко можно зайти: статистика.

## Все сводится к статистике

Какой фактор является самым важным, когда вы решаете, стоит ли использовать свойства CSS, которые не поддерживаются всеми браузерами? Естественно, статистика. Вы могли бы задать такие вопросы: «Когда можно использовать CSS3 для создания закругленных углов? Когда можно полагаться на RGBA при работе с альфа-прозрачностью цвета? Когда можно перестать думать о том, как элементы дизайна будут выглядеть в IE6?» Все зависит от конкретного сайта, над которым вы работаете.

Эрик Мейер, кое-что понимающий в CSS, оставил комментарий в моем блоге более пяти лет назад (*http://simplebits.com/notebook/2004/12/17/ie5/#comment53*) в ответ на вопрос «Когда можно убрать CSS для IE/Mac?» Этот комментарий до сих пор со мной, и его можно применять ко всему, что касается браузерной поддержки.

> *Ответ прост: когда журнал посещений (лог) вашего сайта (или сайтов, если у вас их много) скажет вам, что можно. Не раньше. Статистика пользователей других сайтов и данные глобальных исследований не только бесполезна, но и вредна.*

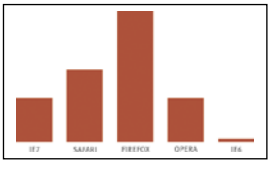


**Рис. 4.10.** Если ваши данные выглядят так, смело расширяйте возможности!

Вот о чем говорил Эрик: не важно, каковы *глобальные* статистические данные по определенному браузеру; наиболее важны данные по сайту, над которым вы работаете.

Большинство гостей вашего сайта все еще работают в IE6? Ну что ж, тогда дважды подумайте о том, стоит ли широко использовать приемы прогрессивного оформления. Но если гостей, работающих в IE6, еле-еле набирается один процент — обязательно попытайтесь расширить возможности.

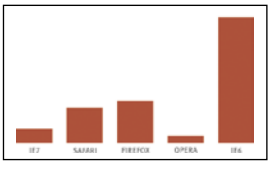


**Рис. 4.11.** Если ваши данные выглядят так, вы будете не столь часто использовать элементы прогрессивного оформления (конечно же, сильно огорчившись)

Статистические данные по *вашему* сайту могут помочь убедить руководителя в необходимости использовать продвинутые стили. Возможен, правда, и обратный эффект.

## Еще несколько новых свойств CSS, которые работают уже сейчас

Мы попытались найти аргументы в пользу того, что веб-сайты не обязаны выглядеть одинаково во всех браузерах. Теперь давайте рассмотрим еще несколько новых возможностей CSS и свойств CSS3, специфичных для производителей, с которыми мы можем поэкспериментировать, чтобы добавить еще немного прогрессивного оформления в шаблон Tugboat.

## СВОЙСТВО TEXT-SHADOW

| | |
|---|---|
| Safari 1+ | ✓ |
| Firefox 3.5 | ✓ |
| Konqueror | ✓ |
| iCab | ✓ |
| Opera 9.5+ | ✓ |

Свойство `text-shadow`, первоначально введенное в CSS2, позволяет добавлять теневые эффекты к тексту. Safari был первым и единственным браузером, который реализовал `text-shadow` уже в первой версии. Сейчас это свойство также поддерживается в Opera 9.5, Konqueror и iCab; кроме того, оно было добавлено в Firefox 3.5 (*http://www.css3.info/preview/text-shadow*).

Я использовал свойство `text-shadow` для оформления надписи «No!» на сайте dowebsitesneedtolookexactlythesameineverybrowser.com, о котором я говорил в начале главы (рис. 4.12). Этот примитивный сайт подтверждает свои утверждения на практике: в IE6 и старых версиях Firefox, к примеру, тени не видно.

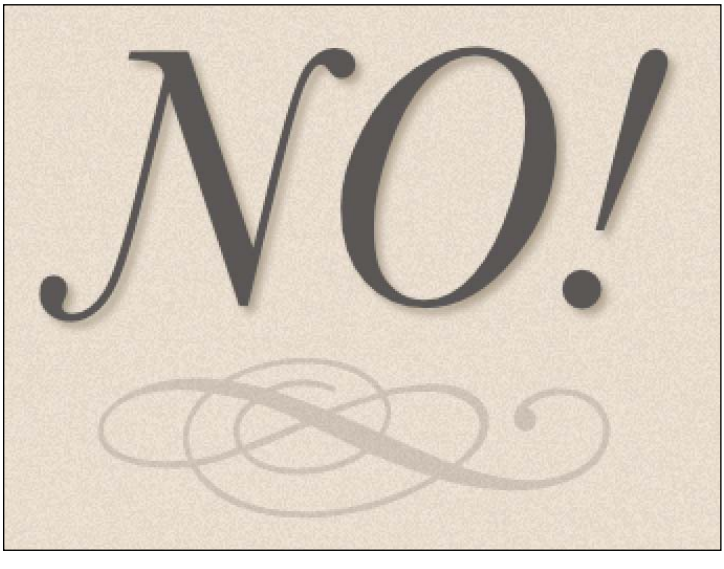



**Рис. 4.12.** Текст, к которому применено свойство text-shadow, в Safari (в увеличенном масштабе)

Давайте посмотрим на другой пример. Добавим тень к тексту в панели навигации шаблона Tugboat. Разметка панели навигации представляет собой неупорядоченный список, и мы будем использовать для каждой ссылки свойство `text-shadow`.

```
#nav ul li a {
   text-shadow: 2px 2px 4px #000;
   }
```

Фактически такой синтаксис говорит следующее: сместить тень на 2 пиксела вправо и на 2 пиксела вниз от текста (чтобы

задать противоположное направление, используются отрицательные значения). Третье значение факультативно и задает радиус размытия тени (в данном случае 4px), а последнее задает цвет тени.

На рисунке 4.13 показан результат в браузере Safari: к тексту в панели навигации добавилась тень. Четыре значения в этом правиле (смещение по горизонтали, смещение по вертикали, радиус размытия и цвет) позволяют добиться множества различных эффектов.


**Рис. 4.13**

**ПРИМЕЧАНИЕ**

Стоит еще раз напомнить, что если вы будете использовать эти свойства, то ваша таблица стилей, скорее всего, не будет соответствовать спецификации CSS2.1. Возможно, вы захотите объединить такие правила в отдельную таблицу стилей (если вам так необходимы эти зеленые сообщения об успешном выполнении). Или же спокойно относитесь к тому, что ваша основная таблица стилей недопустима (я считаю это вполне нормальным). Не бойтесь *запланированных* ошибок валидации.

### Использование RGBA вместе с text-shadow

До этого мы использовали для тени чистый черный цвет. Но что если мы хотим немного смягчить оттенок, применяя новые знания, полученные из главы 3? Есть способ выбрать чуть более темный оттенок коричневого по сравнению с фоном. Однако более простой и гибкий вариант — использовать значение RGBA. Мы снова будем использовать черный цвет, но немного уменьшим его непрозрачность, чтобы тень сливалась с коричневым фоном.

```
#nav ul li a {
   text-shadow: 2px 2px 4px rgba(0,0,0,.7);
   }
```

Так вместо сплошного черного цвета мы задали черный цвет с непрозрачностью 70%. На рисунке 4.14 показана та же тень, которую мы создали вначале, но теперь у нее более изящный

темно-коричневый цвет, полученный смешиванием черного и коричневого.


**Рис. 4.14**

Использование RGBA освободило нас от необходимости выбирать цвет тени, который подошел бы к коричневому фону. Тем самым мы улучшили гибкость дизайна: мы можем изменить цвет фона, не меняя тени.

**ПРИМЕЧАНИЕ**

Хочу напомнить, что RGBA работает еще не во всех браузерах. Более подробно об этом рассказывается в главе 3.

### Эффекты объемности с помощью text-shadow и RGBA

Иногда свойство `text-shadow` используется для создания ощущения объемности текста. Узкая полоска тени, еще незаметнее подчеркивающая фон, — отличный кандидат для RGBA.

Если, используя отрицательное значение, поместить тень шириной 1 пиксел над текстом и задать для нее слегка прозрачный черный цвет без размытия, мы создадим иллюзию того, что текст «вдавлен» в коричневый фон, а источник света находится прямо над текстом (рис. 4.15).

```
#nav ul li a {
   text-shadow: 0-1px 0 rgba (0,0,0,.8);
   }
```

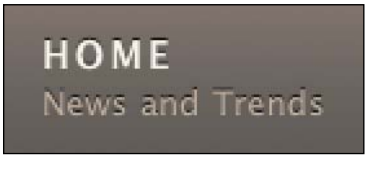



**Рис. 4.15.** Применение свойства text-shadow к тексту панели навигации для создания ощущения объемности (в увеличенном масштабе)

Кроме того, для получения похожего эффекта можно создать более светлую тень *под* текстом. Возьмем для примера кнопку из главы 2 и, задав ее цвет, добавим к тексту белую тень шириной `1px` без размытия с прозрачностью 80%.

```
div.more-btn a {
   padding: 6px 14px;
   text-shadow: 0 1px 0 rgba(255,255,255,.8);
   color: #777;
   border: 1px solid #ccc;
   background: #ccc url(img/glass-btn.png)
     repeat-x 0 50%;
   border-radius: 14px;
   -webkit-border-radius: 14px;
   -moz-border-radius: 14px;
   }
```

На рисунке 4.16 показан результат: кнопка в стиле Mac OS (световой блик под текстом).


**Рис. 4.16.** Кнопка, для которой с помощью text-shadow был создан эффект «вдавленности» (в увеличенном масштабе)

Свойство `text-shadow` в настоящее время тоже имеет ограниченную поддержку, но эти эффекты можно считать визуальным бонусом для браузеров, которые его поддерживают. В остальных браузерах качество оформления также снижается до приемлемого уровня, так как это всего лишь эффект, добавленный к и так читабельному гипертексту.

## СВОЙСТВО BOX-SHADOW

Как и `text-shadow`, свойство `box-shadow` позволяет добавлять эффект тени к любому *элементу*. Синтаксис, в общем, такой же: задается смещение по горизонтали и по вертикали, а затем радиус размытия и значение цвета.

Как правило, создавать по-настоящему гибкие решения с тенями с помощью изображений — неприятная рутинная работа, так как для этого приходится выделять место для тени

фиксированной ширины в сетке макета. Для контейнера, меняющего свой размер, потребуется создать несколько изображений. Но если переложить эти проблемы на плечи CSS, тени будут просто дополнением, которое можно добавить в любой момент.

| | |
|---|---|
| Safari 1+ | ✓ |
| Firefox 3.5 | ✓ |

В настоящее время поддержку `box-shadow`, конечно, нельзя назвать полноценной. Но так как Firefox 3.5 присоединился к числу поддерживающих это свойство, я уверен, мы скоро увидим, что все больше сайтов будут использовать его для создания теней (так же, как и для добавления закругленных углов теперь все чаще применяется CSS3).

## Применение свойства box-shadow к шаблону Tugboat

К примеру, давайте добавим небольшую тень к каждому блоку This Week's Specials в шаблоне Tugboat.

Как вы, наверное, помните из главы 3, напитки недели организованы в виде упорядоченного списка, а `<div class="special">` содержит изображение и информацию по каждому напитку. Также в таблицу стилей для элемента `<div>` включены правила `border-radius`, с которыми мы познакомились в главе 2 (рис. 4.17).

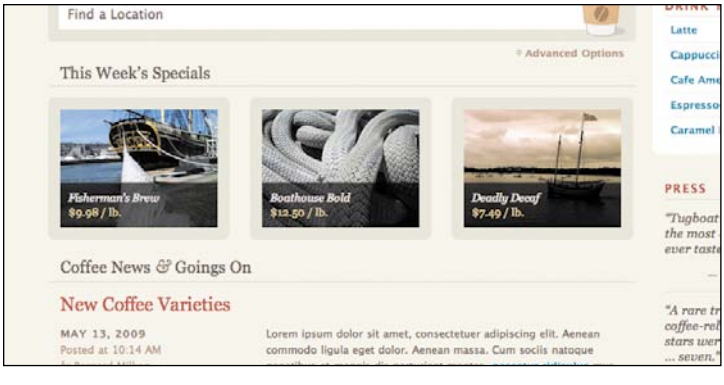



**Рис. 4.17**

```
ol.specials li div.special {
   border: 15px solid #e2e1d4;
   background: #e2e1d4;
   border-radius: 8px;
   -webkit-border-radius: 8px;
   -moz-border-radius: 8px;
   }
```

Добавим в это объявление свойства box-shadow так же, как мы добавляли border-radius в главе 2: будем использовать свойства, специфичные для Safari (-webkit-box-shadow, которое уже поддерживается) и для Firefox (-moz-box-shadow, которое будет поддерживаться в новой версии 3.1), а также стандартное правило CSS3 box-shadow для любого браузера, в который будет добавлена их поддержка.

```
ol.specials li div.special {
   border: 15px solid #e2e1d4;
   background: #e2e1d4;
   border-radius: 8px;
   -webkit-border-radius: 8px;
   -moz-border-radius: 8px;
   box-shadow: 2px 2px 4px #bdbcb0;
   -webkit-box-shadow: 2px 2px 4px #bdbcb0;
   -moz-box-shadow: 2px 2px 4px #bdbcb0;
   }
```

Таким образом, мы создали тень со смещением 2 пиксела по горизонтали и по вертикали и радиусом размытия 4 пиксела; мы выбрали оттенок коричневого цвета, более темный, чем цвет границы. На рисунке 4.18 показан результат. Видно, что если одновременно использовать свойство border-radius (которое уже задано для каждого блока), тень будет учитывать закругленные углы и правильно их обрисовывать (рис. 4.19).


**Рис. 4.18**

**Рис. 4.19.** Один из закругленных углов в увеличенном масштабе: тень правильно обрисовывает изгиб

Всего несколько строк кода, и мы создали гибкий редактируемый блок с тенью и закругленными углами, используя только CSS3. Теперь вспомним, что не все браузеры поддерживают свойство box-shadow (привет браузеру Internet Explorer), так

что в них пользователь не увидит тени (как и закругленных углов). Но я снова скажу (не подумайте, что у меня заело пластинку): ничего страшного. Пусть некоторые гости вашего сайта довольствуются прямыми углами без тени.

### Использование RGBA вместе с box-shadow

Как и в случае с `text-shadow`, цвет здесь может быть задан с помощью RGBA. Если добавить непрозрачность, тень слегка сольется с фоном (в браузерах, поддерживающих RGBA).

Вместо темно-коричневого оттенка, который мы использовали до этого (`#bdbcb0`) будем использовать обычный черный цвет, но уменьшим его непрозрачность. Так, с помощью RGBA мы создадим универсальную тень, которая подойдет к любому фону.

```
ol.specials li div.special {
   border: 15px solid #e2e1d4;
   background: #e2e1d4;
   border-radius: 8px;
   -webkit-border-radius: 8px;
   -moz-border-radius: 8px;
   box-shadow: 2px 2px 4px rgba(0,0,0,.3);
   -webkit-box-shadow: 2px 2px 4px rgba(0,0,0,.3);
   -moz-box-shadow: 2px 2px 4px rgba(0,0,0,.3);
   }
```


Рис. 4.20

Результат (рис. 4.20) получился почти такой же, как и при задании цвета с помощью шестнадцатеричной записи. Но использование черного цвета с непрозрачностью 30% означает, что если мы захотим сделать фон, скажем, красным, синим или зеленым, то тандем `box-shadow`/RGBA будет все равно хорошо работать. При этом не придется вносить никаких поправок (рис. 4.21), так как на фон будет по-прежнему накладываться черная тень с непрозрачностью 30%.


Рис. 4.21

Поскольку Safari и Firefox — единственные два браузера, поддерживающие `box-shadow` (на момент написания этой книги), и они также поддерживают RGBA, использовать эту цветовую модель для тени — вполне логичная идея. В главе 3 мы добавляли резервные правила, дополнительно задавая цвет в шестнадцатеричной записи для браузеров, не поддерживающих RGBA; однако здесь мы можем этого не делать.

## СВОЙСТВО -WEBKIT-TRANSITION

Теперь давайте посмотрим на «нечто симпатичное, вызывающее восторг». WebKit решил поэкспериментировать с анимацией в CSS: будучи прекрасным кандидатом для воплощения идеи прогрессивного оформления, свойство `-webkit-transition` позволяет добавлять по-настоящему *эффектные* вещи, используя самые обычные таблицы стилей. В настоящее время эти поразительные трюки работают *только* в Safari, однако отсутствие таких анимационных эффектов в других браузерах безвредно и не является серьезным недостатком.


**Рис. 4.22**

Сейчас, как никогда, я жалею о том, что в книгу нельзя вставить видео, ведь мы говорим о визуальных эффектах и писать о них кажется просто несправедливым. Но я буду очень стараться.

Обязательно оцените CSS-анимацию в блоге Surfin' Safari (*http://webkit.org/blog/138/css-animation/*), где еще в 2007 году были продемонстрированы некоторые удивительные возможности (рис. 4.22).

Поворачивать элементы HTML, повышать/понижать яркость при наведении или щелчке мышью, увеличивать/уменьшать ширину границы и отступы — о некоторых из этих эффектов вы раньше и не слышали, а сейчас их можно создать с помощью нескольких простых CSS-правил (специфичных для производителей).

### Изменение яркости при наведении в шаблоне Tugboat

Чтобы показать, как легко применить малозаметный, но полезный эффект, скажем, к навигационным ссылкам сайта, попробуем с помощью свойства `-webkit-transition` добиться изменения яркости фона ссылки при наведении на нее курсора мыши. Наш небольшой пример не показывает всего, что можно делать с помощью CSS-переходов, но для начала и этого достаточно.

Панель навигации в шаблоне Tugboat (рис. 4.23) представляет собой неупорядоченный список, заключенный в тег `<div id="nav">`; каждая ссылка является отдельным элементом списка — на сегодняшний день это стандартная структура. В упрощенном виде разметка выглядит примерно так.

**СОВЕТ**

Элементы `<strong>` и `<em>` позволяют по-разному задать стили двух частей ссылки.

```
<div id="nav">
   <ul>
      <li><a href="/"><strong>Home <em>News and
     Trends</em></strong></a></li>
      …
      </li>
   </ul>
</div>
```


**Рис. 4.23**

В качестве простого hover-эффекта мы можем поменять цвет фона позади текста ссылки. А поскольку в панели навигации используется градиентный фон (рис. 4.24), почему бы не использовать RGBA? Если задать слегка прозрачный цвет, эффект градиента сохранится и при наведении.

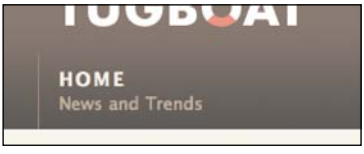



**Рис. 4.24.** Градиентный фон панели навигации в увеличенном масштабе: цвет переходит от темно-коричневого (внизу) к светло-коричневому (вверху)

```
#nav li a: hover {
   background: #000;
   background: rgba (0,0,0,.15);
   }
```

Чтобы не искать подходящий оттенок коричневого (более темный, чем фон панели навигации), мы снова будем использовать черный цвет, но уменьшим его непрозрачность до 15% с помощью RGBA. Также зададим резервное значение цвета в шестнадцатеричной записи для браузеров, не поддерживающих RGBA.

На рисунке 4.25 показано, как меняется панель навигации при наведении курсора мыши: так как мы использовали значение RGBA с непрозрачностью 15%, эффект градиента сохраняется, благодаря чему цвет фона за текстом ссылки не так резко выделяется.

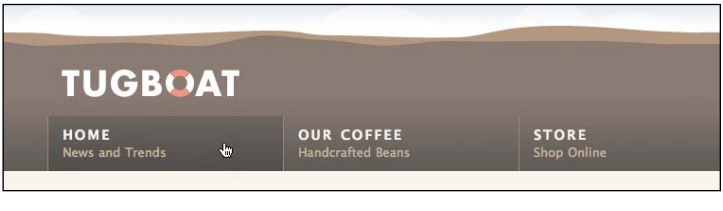



**Рис. 4.25**

Все станет гораздо интереснее, когда мы воспользуемся свойством `-webkit-transition`. Если добавить переход для цвета фона и задать время перехода, при наведении курсора на элемент панели навигации цвет фона будет плавно меняться.

```
#nav li a: hover {
   background: #000;
   background: rgba(0,0,0,.15);
   }

#nav li a {
   -webkit-transition-property: background-color;
   -webkit-transition-duration:.4s;
   -webkit-transition-timing-function: linear;
   }
```

К каждой ссылке в панели навигации мы добавили три свойства, управляющих переходом:

1. — `webkit-transition-property` задает свойство, для которого должен произойти переход (в нашем случае цвет фона);
2. — `webkit-transition-duration` задает время, за которое должен произойти переход (в нашем случае четыре десятых секунды);
3. — `webkit-transition-timing-function` определяет то, как изменяется скорость за время перехода. Здесь мы выбрали `linear`, но есть и другие варианты, которые напоминают мне жуткие домашние задания по математическому анализу. (Посмотрите полное описание спецификации на *http://www.w3.org/TR/css3-transitions/*.)

В качестве альтернативы можно использовать краткую запись, объединяя все эти свойства в одно.

```
#nav li a {
   -webkit-transition: background-color;
   -webkit-transition-duration:.4s;
   -webkit-transition-timing-function: linear;
   -webkit-transition: background-color.4s
     linear;
   }
```

Жаль, что я не могу показать результат здесь (рис. 4.26 — лишь жалкая попытка это сделать). Это такой эффект, который раньше можно было увидеть только у flash-элементов, а теперь его можно легко создать с помощью CSS. Этот эффект едва уловим, и не каждый его даже заметит (и более того, в настоящее время его могут увидеть только пользователи Safari). Но его невероятно просто реализовать, и он безвреден для браузеров,

не распознающих такой код. Мастерство — это мелкие, иногда крайне неочевидные детали.

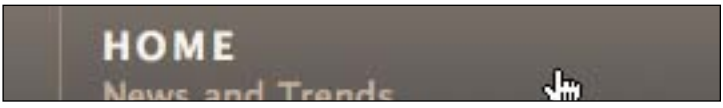



**Рис. 4.26**

### Применение переходов к цвету текста гиперссылок

Мы можем создать эффект перехода для *всех* текстовых гиперссылок на странице, добавив похожее плавное изменение цвета ссылки (а не фона, как в предыдущем примере).

Вставив `-webkit-transition` в основное объявление стиля всех ссылок страницы и выбрав в качестве свойства, которое должно меняться при переходе, `color` (а не `background-color`), мы создадим эффект изменения цвета ссылок при наведении.

Вот как выглядят наши два объявления *без* добавления эффекта перехода. Как вы помните из главы 3, мы ловко используем для `hover`-эффекта значения RGBA, сохраняя обычный цвет ссылки, но уменьшая его непрозрачность до 65%.

```
a: link, a: visited {
   font-weight: bold;
   text-decoration: none;
   outline: none;
   color: #3792b3;
   }

a: hover {
   color: rgba(55,146,179,.65);
   }
```

А теперь добавим правило перехода в нормальное состояние ссылки. Я уменьшил время перехода до `.2s`: для обычных гиперссылок лучше, если скорость реакции выше.

```
a: link, a: visited {
   font-weight: bold;
   text-decoration: none;
   outline: none;
   color: #3792b3;
   -webkit-transition: color.2s linear;
   }
```

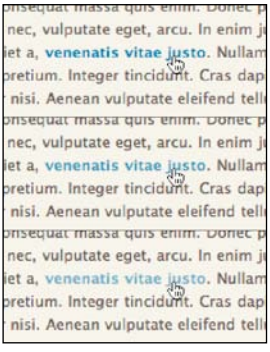



**Рис. 4.27**

```
a: hover {
   color: rgba(55,146,179,.65);
   }
```

В результате мы получили малозаметное изменение цвета гиперссылок, которое, несомненно, вызовет восторг у ваших друзей-поклонников Safari (рис. 4.27 — еще одна жалкая попытка показать вам эффект анимации; и вас, наверное, мучает вопрос, почему же я не сделал специально для этого флипбук). И опять ничего не произойдет в других браузерах: они проигнорируют то единственное правило, которое создает весь этот эффект.

## В ЗАКЛЮЧЕНИЕ

Должны ли сайты выглядеть одинаково во всех браузерах? Если ваш ответ — «нет», то вы сразу развязываете себе руки, чтобы пользоваться теми замечательными возможностями CSS3, которые мы до сих пор обсуждали в этой книге. Если считать элементы прогрессивного оформления *визуальными бонусами*, а не обязательными составляющими дизайна, в вашем арсенале неожиданно появятся простые и гибкие инструменты, с которыми гораздо легче работать. При этом стоит надеяться, что экспериментируя с этими свойствами уже *сейчас* или даже используя их при создании сайтов, вы будете иметь преимущество, так как все большее количество браузеров поддерживает эти развивающиеся стандарты.

Когда углы блока выглядят закругленными в одном браузере и прямыми в другом, это нормально? Безусловно, если сохраняется удобочитаемость и функциональность (а это и есть самое главное).

Давайте будем «теми, кто готов выложиться на 80 процентов». Создание сложных решений, касающихся визуального оформления, требует практичного кода и браузеров, которые позаботились о его поддержке. Поэтому мы будем обращать внимание только на важные детали.

# Глава 5

## Управление плавающими блоками

**Главное различие между мастером и экспертом заключается в том, что происходит после достижения достаточного уровня знаний. Эксперт сделает все, чтобы остаться верным одной-единственной области, и только в ней он будет стремиться узнавать что-то новое, применять знания на практике и создавать проекты. Мастер же обладает достаточным мужеством и скромностью, чтобы отбросить свои профессиональные навыки и овладеть новой технологией или освоить новую область.**

Дэйв Гувер (из статьи, опубликованной на StickyMinds.com)

Один из тезисов, к которым я хочу привлечь внимание в этой книге, состоит в том, насколько важно *переоценивать старые методы и лучшие технические приемы*. Этот вопрос является ключевым в веб-дизайне, так как браузеры постоянно развиваются, внедряют новые стандарты и двигаются вперед со стремительно возрастающей скоростью. Приемы, которые были лучшими несколько лет назад, в современном контексте могут оказаться устаревшими. Хитрый трюк или «заплатка», которые применялись раньше, чтобы обойти проблему в определенном браузере, теперь могут быть не нужны. Свойство CSS, которое какое-то время назад не имело широкой браузерной поддержки, могло в итоге дойти до поворотного пункта, открыв более простые пути для выполнения дизайнерских задач.

Конечно, я тоже когда-то *избегал* такой переоценки. Я продолжал придумывать обходные пути в ситуациях, где лучше было бы использовать решения, для которых, как я тогда думал, «сейчас не самое лучшее время». Я отказывался выходить за рамки привычного и использовать передовые технические решения, планируя уделить им внимание тогда, когда они станут широко распространены.

Но с тех пор мое отношение к этому изменилось, и, надеюсь, оно найдет свое отражение на страницах этой книги. Один из компонентов переоценки — это умение «не обращать внимания». Существенную часть предыдущей главы мы посвятили разговору о том, что если дизайн выглядит неодинаково в разных браузерах, то это не страшно. И здесь мы как раз имеем дело с настоящей переоценкой; если спокойно относиться к этим вариациям, мы будем иметь в своем распоряжении широкий арсенал средств CSS3, которые в настоящее время внедряются передовыми браузерами.

Еще один компонент переоценки заключается в том, чтобы возвращаться к лучшим техническим приемам и проверять, они по-прежнему актуальны или уже нет. Есть ли более удобный и простой способ сделать то, что мне нужно? Не стал ли этот конкретный метод стереотипным решением, которое когда-то было стандартным, но теперь может быть усовершенствовано?

Настоящий мастер будет задаваться такими вопросами и переоценивать методы, если это поможет улучшить конечный результат. На это указывает Дэйв Гувер в эпиграфе. Он сравнивает эксперта с мастером, говоря о мужестве, которое потребуется для того, чтобы выйти за рамки привычного и научиться чему-то новому. Мастер не побоится переоценить методы и лучшие приемы и не станет придерживаться узкого одностороннего взгляда на решение конкретной задачи при реализации дизайнерского проекта.

Итак, в этой главе мы хотим переоценить понятие «самоочищающихся плавающих элементов», хотя с первого взгляда эта тема кажется скучной, унылой и не заслуживающей того, чтобы посвящать ей целую главу. Но использование плавающих элементов, увы, до сих пор является лучшим способом обеспечить гибкость верстки с помощью CSS, поэтому сохранение изящных и эффективных приемов работы с ними — превосходный материал для переоценки.

## Еще раз о плавающих элементах

Я уверен, вы уже знакомы с проблемами, которые могут вызывать плавающие элементы. Хотя на сегодняшний день плавающие элементы — лучший способ создания большинства схем расположения с помощью CSS, проблема *размещения* их в контейнере всегда была основной причиной для разочарования.

Возьмем для примера белый блок, в котором содержится картинка и элемент `<div>` с текстом, прижатый к правому краю (рис. 5.1). В упрощенном виде разметка может выглядеть примерно так.

```
<div class="box">
   <img src="/img/latte.jpg" alt="Latte at
     Stumptown" />
   <p class="description">Lorem ipsum dolor …</p>
</div>
```

А стили будут выглядеть примерно так.

```
div.box {
   background: #fff;
   }
div.box img {
   width: 25%;
   }
```

```
div.box p.description {
   float: right;
   width: 70%;
   }
```

**ПРИМЕЧАНИЕ**

Я специально не включил некоторые детали, чтобы показать только самое важное. Кстати, вы, наверное, захотите спросить: «А почему мы задаем ширину изображения в процентах?» Подождите немного, и в следующей главе Итан Маркотт покажет нам все возможные варианты.


**Рис. 5.1**

На рисунке 5.2 показана обычная ситуация: если высота плавающего элемента (в данном случае абзац текста) больше высоты обычного элемента (картинка с кофе латте), то плавающий элемент выйдет за границы контейнера (грустное нисходящее глиссандо тромбона).


**Рис. 5.2.** Ситуация, в которой использование плавающих элементов в контейнере не приводит к желаемому результату

В идеальном мире контейнер знал бы, что в нем содержится, и расширил бы свои границы, чтобы все это поместилось. Увы, с плавающими элементами дело обстоит не так. Но это не их вина — плавающие элементы не созданы для работы со сложными (и даже не очень сложными) схемами расположения, для которых мы их используем.

Вместо этого мы нашли несколько оригинальных способов заставить контейнер «очистить» все плавающие элементы, которые в нем содержатся. Но я не собираюсь тратить бумагу и подробно их здесь описывать. Я уже сделал это в книге «Пуленепробиваемый Web-дизайн» (Bulletproof Web Design, New Riders, 2007).

Лучше я остановлюсь на одном решении, в надежности которого я уверен: этот метод «очистки» плавающих блоков работает в наибольшем числе различных ситуаций.

### МЕТОД EASY CLEARING

**СОВЕТ**

Вы, возможно, помните из главы 4, что IE7 и более ранние версии не поддерживают `:before` и `:after`.

В статье «How to Clear Floats Without Structural Markup» (*http://positioniseverything.net/easyclearing.html*), впервые опубликованной еще в 2004 году на поистине незаменимом сайте Position Is Everything, описан метод автоматической «очистки» всех плавающих блоков для данного контейнера: используя в таблице стилей псевдоэлемент `:after`, нужно добавить точку после контейнера, «очистить» плавающие блоки, а затем сделать эту точку невидимой. Это разумное решение, но оно требует особых ухищрений для разных версий IE.

Я снова скажу, что существуют и другие способы добиться этой цели, но я придерживаюсь этого, так как он оказался самым пуленепробиваемым.

**СОВЕТ**

Чтобы ознакомиться с другими методами «самоочистки» плавающих блоков в контейнере, посмотрите статью на SitePoint, в которой рассматриваются несколько распространенных решений (*http://www.sitepoint.com/blogs/2005/02/26/simple-clearing-of-floats/*).

## Использование класса .clearfix

Чтобы показать, как работает этот метод, давайте применим его к предыдущему примеру, в котором слишком длинный абзац

вылезал из блока, в котором он располагался. Следуя методу Easy Clearing, создадим класс `.clearfix` с такими правилами.

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

Такой класс можно применить к любому контейнеру, в котором вы хотите автоматически «очистить» плавающие элементы. Этот метод будет хорошо работать в браузерах, поддерживающих псевдоэлементы `:before` и `:after`, но Internet Explorer опять «в пролете». Для IE версий 6 и 7 нужно будет добавить два дополнительных правила, позволяющих эффективно размещать плавающие блоки в контейнере за счет особенностей этих браузеров.

## ДОПОЛНИТЕЛЬНЫЕ ПРАВИЛА ДЛЯ IE6 И 7

Для IE6 можно просто создать правило, которое задает для контейнера произвольную высоту.

```
/* для IE6 */
* html.clearfix {
   height: 1%;
   }
```

Для IE7 проблема решается добавлением `min-height: 1px`, и здесь снова используется этот дурацкий трюк с селектором, чтобы показать, что код предназначен для IE7.

```
/* для IE7 */
*:first-child+html.clearfix {
   min-height: 1px;
   }
```

Наконец, для IE8 мы будем использовать совсем уж безумный синтаксис, чтобы показать, что «очистка» плавающих блоков предназначена для последней версии браузера Microsoft.

```
* /_** (.)/*+:@//\*+html! ~/rofl**/ {
   clear: please !reallyreallyimportantomg;
   }
```

Шутка! Просто проверка на внимательность.

Нет, на самом деле я сообщу вам хорошую новость: IE8 поддерживает псевдоэлементы `:before` и `:after`, так что никаких дополнительных объявлений не требуется. IE8 распознает исходное правило, где используется `:after`.

**СОВЕТ**

Вместо трюков с селекторами (выделяющих код, предназначенный для определенной версии IE) можно использовать условные комментарии, помещая эти объявления в отдельные таблицы стилей для каждой версии IE.

Добавив эти правила, мы получим реализацию «самоочистки» блоков контейнера, которая работает во всех браузерах.

### ДОБАВЛЕНИЕ КЛАССА .CLEARFIX В РАЗМЕТКУ

Возвращаясь к разметке нашего примера, давайте применим класс `.clearfix` к элементу `<div>`, который содержит плавающий текст и картинку.

```
<div class="box clearfix">
   <img src="/img/latte.jpg" alt="Latte at
     Stumptown" />
   <p class="description">Lorem ipsum dolor …</p>
</div>
```

После добавления класса `.clearfix` в контейнер три заданных ранее объявления автоматически «очистят» все плавающие элементы внутри него. Благодаря этому, блок с текстом и картинки будут одним самостоятельным модулем, не зависящим от окружения.

## Семантическая дилемма

Именно здесь в игру вступает переоценка. Мы знаем, что этот метод был придуман более 5 лет назад, и поэтому нельзя винить его авторов в проблеме, которую я здесь вижу: меня беспокоит термин «clearfix», использующийся в названии класса.

Создавать такие классы заранее удобно потому, что `.clearfix` можно применить к любому контейнеру, в котором используются плавающие элементы. Проблема в том, что такие, как я, применяют этот класс к большому числу элементов на странице. Ведь использование плавающих блоков — лучший способ

добиться гибкого дизайна, и поэтому правильная «очистка» этих блоков принципиально важна при создании пуленепробиваемых интерфейсов. Так, отдельные части страницы можно сделать независимыми, чтобы их было легко передвигать, не нарушая расположения остальных элементов макета.

Таким образом, недостаток использования слова «clearfix» состоит в том, что ваш шаблон очень скоро будет «захламлен» этим асемантическим термином. А это внесет большую путаницу.

Представьте, как ваш заказчик или босс просматривает разметку и на каждом шагу натыкается на слово «clearfix». Что же это за проблема, которую требуется так часто решать?

## ОДНО ИЗ ВОЗМОЖНЫХ РЕШЕНИЙ: БОЛЬШОЙ И ДЛИННЫЙ СПИСОК

Вместо того чтобы всюду добавлять имя класса, можно создать отдельное объявление, где будут перечислены все элементы, которые вы хотите «очистить». В результате класс будет вынесен из разметки, но вам придется внимательно следить за этим списком.

Так, в нашем примере вместо использования класса `.cleafix` мы можем задать список элементов, которые нужно «очистить», с помощью сложного объявления.

```
div.box: after,
#header: after,
#nav: after {
   content: ".";
   display: block;
   height: 0;
   clear: both;
   visibility: hidden;
   }
```

Этот способ можно использовать при создании маленьких сайтов с ограниченным числом элементов на странице. Однако если сайт большой, этот список может вскоре стать громоздким, и с ним будет неудобно работать. Если вы решите исправить или удалить какой-нибудь класс, нужно будет сделать соответствующие изменения и в этом списке. Более того, обновления придется делать в трех местах: в этом объявлении с псевдоэлементом `:after` и в двух дополнительных объявлениях для IE6 и IE7.

### Возможные трудности при использовании списка

Я на себе испытал все эти трудности, когда создавал таблицы стилей для сайта MTV.com (рис. 5.3). Автоматическую «очистку» всех модулей сайта мы решили реализовать с помощью такого списка. При этом сайт был сложным: в нем использовалось много разных схем расположения и плавающего контента.


**Рис. 5.3.** Сайт MTV.com весной 2009

Ниже я привел копию списка для сайта MTV. Как видите, он разросся до размеров настоящего *монстра*, состоящего из компонентов, внутри которых нужно «очистить» плавающие блоки. Имена классов с течением времени приходилось изменять или удалять, и каждый раз обновлять этот список (и такие же списки для IE) было почти таким же увлекательным занятием, как сортировать мусор в алфавитном порядке.

```
#header:after,
#memberbar:after,
#nav:after,
#wrap:after,
#wrap-inner:after,
div.main-feature:after,
div.marquee-half:after,
ol.slat li:after,
```

```
ol.promo:after,
ol.threeby:after,
ol.fourby:after,
ol.pop:after,
div.vidclips:after,
ul.vid-toggle:after,
div.full ol li:after,
div.full ol li a:after,
div.main-artist:after,
p.alpha-list:after,
div.twocol:after,
div.twocol-row:after,
div.mtv2-twocol:after,
div.threecol:after,
div.threecol ul:after,

#wrap ol.gallery:after,
.results:after,
.results-nav:after,
div.main-graphic:after,
div.flipbook-thumbs:after,
ul.rateit:after,
div.tabnav ul:after,
ol.cmnts li div.cmnt:after,
div.lyrics-wrap:after,
div.lyricsearch-main:after {
   content: ".";
   display: block;
   height: 0;
   clear: both;
   visibility: hidden;
   }
```

Таким образом, хотя при создании списка элементов, которые вы хотите «очистить», не нужно добавлять специальный класс в разметку, это довольно опасный путь, если вы работаете над большим или средним по размеру сайтом.

### ВЫБОР БОЛЕЕ ПОДХОДЯЩЕГО ИМЕНИ КЛАССА

Итак, проведем переоценку. Пять лет назад использование имени класса `.clearfix` казалось безобидным, но ведь мы стали умнее и хитрее, не так ли?

Не так давно я стал использовать для «самоочистки» имя класса `.group`. С точки зрения семантики это более понятный термин. Мы подразумеваем, что контейнер — это группа элементов, часть из которых является плавающими. Слово «группа», на мой взгляд, хорошо подходит.

Вот так выглядит разметка для предыдущего примера, где `.group` используется вместо `.clearfix`.

```
<div class="box group">
   <img src="/img/latte.jpg" alt="Latte at
     Stumptown" />
   <p class="description">Lorem ipsum dolor …</p>
</div>
```

Разве не замечательно? Ну, по крайней мере, уж точно стало лучше.

Я понимаю, что здесь мы только с натяжкой можем говорить о семантике. Но, используя имя класса `.group`, мы хотя бы *пытаемся* отразить суть того, что при этом происходит с контейнером; используя же `.clearfix`, мы лишь привлекаем внимание к проблеме. Тем, кто не может (или не должен) понимать всех тонкостей верстки веб-страниц с помощью плавающих блоков, не покажется подозрительным применение класса `.group` к большому количеству элементов в разметке.

## Создание класса .group в таблице стилей

Теперь, когда мы выбрали для класса более подходящее имя, самое время создать объявления, выполняющие «очистку» и позволяющие использовать класс `.group` там, где это необходимо. Я, пожалуй, создам для правил, относящихся к IE, свою таблицу стилей (обычное название для такой таблицы — *ie.css*). Тогда чистые и не использующие особых трюков CSS будут храниться отдельно.

На рисунке 5.4 показаны три объявления, выполняющие «очистку» для всех браузеров: в главном файле *master. css* мы видим реализацию с помощью псевдоэлемента `:after`, а в файле *ie.css* содержатся два дополнительных правила для IE6 и 7.

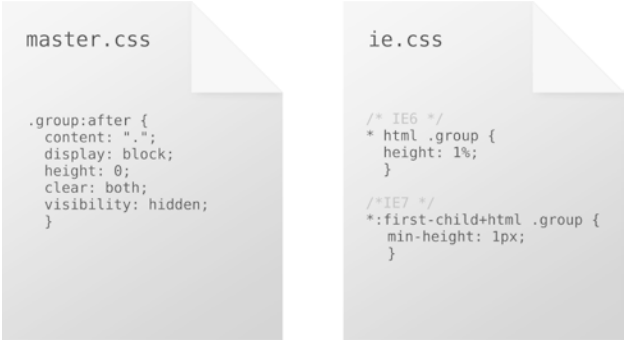



**Рис. 5.4.** Выделение специальных правил, относящихся к IE, в отдельную таблицу стилей

Как видите, я использую селекторы `* html` и `*:first-child+html`, чтобы показать, что код предназначен, соответственно, для IE6 и IE7. Но вместо этого можно создать для каждого из них собственную таблицу стилей (рис. 5.5) и добавить в тег `<head>` документа условные комментарии, чтобы установить соответствие между браузером и названием CSS-файла.

```
<head>
<link href="/css/master.css" media="screen,
     projection"
rel="stylesheet" type="text/css" />
<!-- [if IE 6]>
     <link href="/css/ie6.css" media="screen,
     projection"
rel="stylesheet" type="text/css" />
<! [endif]-->
<!-- [if IE 7]>
     <link href="/css/ie7.css" media="screen,
     projection"
rel="stylesheet" type="text/css" />
<! [endif]-->
</head>
```

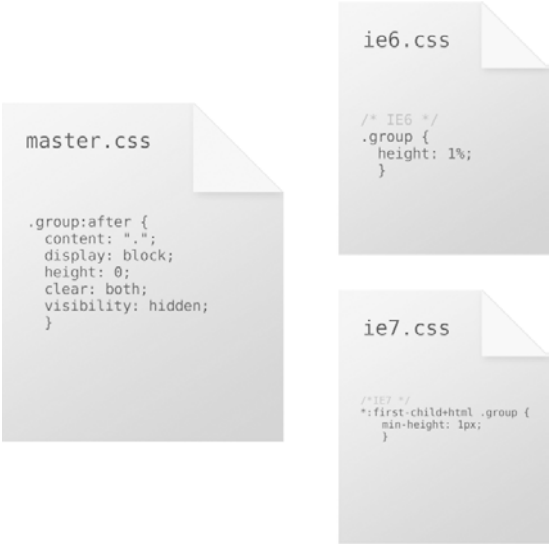



**Рис. 5.5.** Создание отдельных таблиц стилей для разных версий IE, для добавления которых потребуются условные комментарии

## СОЗДАЙТЕ СТИЛЬ И ЗАБУДЬТЕ ОБ ЭТОМ

Когда мы реализовали «очистку», используя понятное имя класса, и добавили дополнительные правила для всех браузеров, у нас появилось замечательное преимущество: теперь мы знаем, что можно применить `.group` к любому элементу на странице. Если перед вами контейнер, в котором нужно «очистить» плавающие блоки, просто примените к нему этот класс и спите спокойно, зная, что ваш контейнер будет независимым и гибким, а смена его расположения не приведет к нарушению верстки всей страницы.

Сейчас я редко об этом задумываюсь, но это надежный инструмент, и он в нашем распоряжении. Работая над дизайном проекта, я всегда знаю, что можно применить класс `.group` к любому элементу, и все плавающие блоки будут удобно в нем размещаться. И мне не придется беспокоиться о добавлении контейнера в списки, а разметка не будет «захламлена» классом со странным названием.

Это первый шаг к созданию более сложной системы настроек, своего рода фреймворка. Идея состоит в том, чтобы писать часто повторяющийся код один раз, а затем использовать его в различных проектах. Поэтому далее мы немного поговорим о таких фреймворках.

## Фреймворки для мастеров

*— Вы используете CSS-фреймворк?*

Я часто слышу такой вопрос. Идея использования обобщенной структуры для таблиц стилей и разметки стала немного популярнее в последние годы (на рис. 5.6 показаны два классических примера). Хотя стандартные структуры могут быть полезны при обучении, я предпочитаю создавать свои собственные.



**Рис. 5.6.** Два классических CSS-фреймворка: Blueprint (*http://blueprintcss.org/*) и 960 Grid System (*http://960.gs/*)

CSS-фреймворк — это заранее подготовленный набор HTML и CSS-файлов, достаточно универсальных для того, чтобы их можно было брать за основу при создании различных проектов. Существует несколько широко распространенных вариантов, и каждый из них имеет свое назначение. Идея в том, чтобы при создании каждого нового проекта использовать этот готовый набор файлов, в которых для вас уже создан макет, задано шрифтовое оформление и написан весь основной код, общий для большинства сайтов.

Как правило, я рекомендую использовать CSS-фреймворки только при создании прототипов и каркасов на скорую руку или же в процессе обучения. Я предпочитаю начинать с нуля каждый раз, когда разрабатываю новый сайт. В большинстве случаев мне пришлось бы удалить существенную часть фреймворка, чтобы правильно реализовать все дизайнерские решения. Иначе говоря, если вы используете готовый фреймворк, то существует опасность того, что в вашем коде будет много ненужного хлама.

Тем не менее кое в чем фреймворки *могут* нам пригодиться. В них, конечно же, есть фрагменты кода, которые я использую в различных проектах, причем независимо от визуального оформления. Если идентифицировать эти часто используемые фрагменты, они будут полезны при *создании* CSS-фреймворка. Вашего собственного.

## СОЗДАЙТЕ СВОЙ ФРЕЙМВОРК

Так что мой ответ — «да». Я использую CSS-фреймворк, но только он очень простой; я сам его создал и применяю всякий раз, когда приступаю к реализации нового проекта (рис. 5.7). И так поступают многие, не только я. Настоящий мастер всегда *создает* свои собственные инструменты, выделяя фиксированный набор правил, которые используются в любых проектах независимо от их дизайна — чистый лист, который ждет, чтобы его заполнили на начальном этапе проекта.

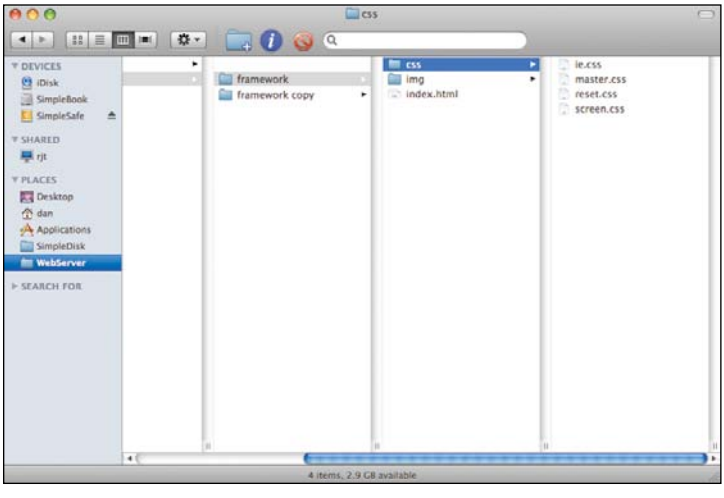



**Рис. 5.7.** Базовая структура файлов предельно упрощенного пустого фреймворка, который я использую на начальной стадии каждого проекта

Код для метода «очистки», который мы обсуждали в этой главе, является частью этого фреймворка. Объявления класса `.group` включены в стандартный набор таблиц стилей, которые я добавляю в каждый дизайнерский проект (в файлы *master.css* и *ie.css*). Я написал их один раз, зная, что класс `.group` сделает свое дело независимо от того, в каком месте разметки мне придется его использовать.

Давайте посмотрим, как устроены таблицы стилей моего фреймворка. Я покажу вам код каждого CSS-файла, сопровождая его интересными комментариями.

## INDEX.HTML

Сначала поговорим об HTML-файле, который служит шаблоном по умолчанию для любого моего проекта. Я разумно

называю его *index.html*, но вы можете легко предложить другой, более интересный вариант.

Внутри файла *index.html* вы найдете простую незаполненную структуру с «рыбами» для тех элементов, которые я планирую использовать в любом проекте. И здесь стоит упомянуть о некоторых особенностях.

### Выберите тип документа, но не слишком старайтесь

Последние несколько лет я использую тип XHTML 1.0 Transitional. Изменится ли это когда-нибудь? По всей вероятности, нет. В этой книге мы говорили в основном о нововведениях, касающихся CSS, хотя вообще-то такое же развитие наблюдается и в разметке. HTML5 и XHTML 2 двигаются вперед, претендуя на ведущую роль в формировании будущего HTML в Сети. Пока все говорит о том, что лидирует HTML5, но наверняка это можно будет сказать только по прошествии времени. Версия HTML5 является пока что лишь экспериментальной, учитывая количество браузеров, существующих на сегодняшний день.

Так что пока я продолжаю использовать XHTML 1.0, потому что это удобно, и мне нравятся правила, накладываемые на разметку в соответствии со спецификацией (для тегов обязательно использовать строчные буквы, все элементы должны быть закрыты и т. д.). Но со временем становится все более очевидным, что выбор HTML 4.01 Strict, XHTML 1.0 Transitional, XHTML 1.0 Strict или другого типа не имеет особого значения для конечного пользователя.

### Как скрыть все стили от IE6

Восемь лет назад Джеффри Зельдман в своей статье «К черту плохие браузеры» («To Hell with Bad Browsers», *http://www.alistapart.com/articles/tohell*) возвестил о начале «века CSS». Он рассказал, что если использовать свойство `@import` для добавления таблиц стилей, Netscape 4 будет их игнорировать. Это стало важным аргументом, призывающим обратить внимание на проблемы обновления старых браузеров. Прошло *восемь лет*.

Полностью игнорировать браузер средствами CSS — это невероятно удобно. Конечно, этот прием нельзя использовать на всех сайтах. Важно помнить, что статистика сайта определяет тот уровень поддержки, который вы должны обеспечить. Об этом мы уже говорили в предыдущей главе.

Позднее возникли проблемы с IE5/Mac. Я использовал такую же политику игнорирования, какая применялась по отношению к Netscape 4: трюк с комментариями и обратными слэшами, о котором я расскажу чуть позже.

```
/* import stylesheets and hide from IE/Mac \*/
@import url("screen.css");
/* end import and hide */
```

**СОВЕТ**

На всякий случай: Стив Саудерз провел масштабное исследование, пытаясь определить, как использование свойства `@import` в различных конфигурациях для добавления таблиц стилей влияет на время загрузки и производительность страницы. Вы можете посмотреть, что он выяснил, на *http://www.stevesouders.com/blog/2009/04/09/dont-use-import/*.

Сейчас причиной наших страхов и проблем является IE6. Поэтому я однозначно хочу добиться того, чтобы некоторые сайты выглядели в этом браузере неоформленными, «голыми». PNG-изображения с альфа-каналом, свойства `min-width` и `max-width` — все время что-то не работает. На многих сайтах, которые я разрабатывал и обслуживаю, процент пользователей IE6 достаточно мал, чтобы можно было не обращать на них внимания и двигаться дальше. Так что сейчас самое время!

Так какое решение будет самым простым? Я открыл Google и нашел статью Саймона Клэйсона (*http://www.simonclayson.co.uk/reportage/ie_6_text_only/*), где автор с умом использует условные комментарии, чтобы скрыть элемент `<link>`, добавляющий таблицы стилей. Я написал об этом в своем блоге на SimpleBits (*http://simplebits.com/notebook/2009/02/13/iegone/*), а после обсуждения придумал более простой вариант, который я использую и по сей день в моем фреймворке, а также в нескольких действующих проектах.

```
<!-- [if gte IE 7]><!-- >
   <link rel="stylesheet" type="text/css"
     media="screen, projection"
     href="css/screen.css" />
<!-- <![endif]-->
```

Такой код скрывает все стили (если, конечно, они содержатся в файле *screen.css*) от IE6 и более ранних версий IE, но успешно импортирует их в остальные браузеры. Гости, которым

посчастливилось зайти на сайт в IE6 или более ранней версии, не обнаружат почти никакого оформления, так же как и пользователи Netscape 4 почти десять лет назад.

В оригинальном варианте метода Саймона предлагалось также создавать крайне упрощенный CSS-файл специально для IE6, хотя я считаю это излишней щедростью. Еще один реальный пример использования этого метода на практике — сайт The Rissington Podcast (*http://therissingtonpodcast.co.uk/*), в котором для IE6 создана специальная таблица стилей, а для текста используется шрифт Comic Sans, чтобы еще больше подразнить пользователей этого браузера (рис. 5.8).

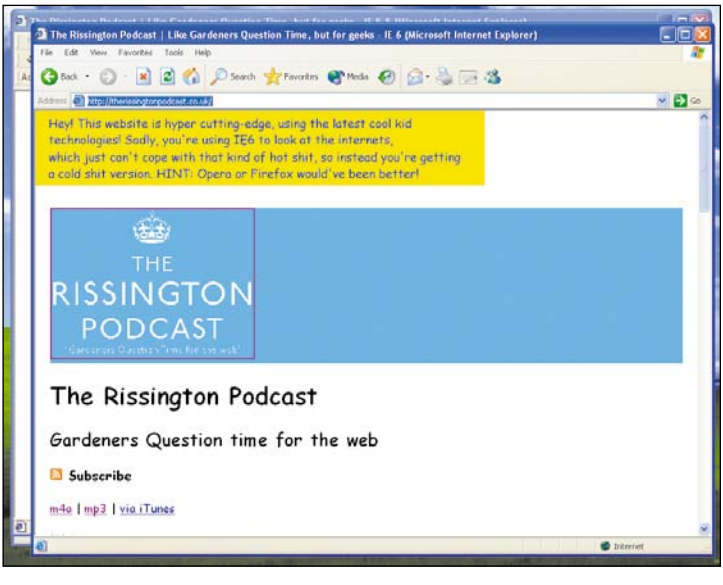



**Рис. 5.8.** Сайт The Rissington Podcast в IE6: оформление почти отсутствует, а для текста используется шрифт Comic Sans

Независимо от того, создаете ли вы дополнительные стили или скрываете их все, у этого подхода есть существенное преимущество: так как не все ваши таблицы стилей импортируются в IE6, вам не нужно беспокоиться о том, чтобы их переопределить. Для IE6 можно создать минимальные стили, начав с нуля. Или вообще не создавать их.

Похоже на какой-то трюк? Да. Но в *определенных ситуациях* возможность не беспокоиться о IE6 этого стоит. И, безусловно, сейчас уже пришло время начать экспериментировать с методами, позволяющими скрыть стили от IE6 (хотя даже просто говорить об этом невероятно интересно).

Этот метод стоит, по крайней мере, добавить в свой арсенал, чтобы его можно было использовать в будущем.

### Странный комментарий с ASCII-графикой

Прямо перед элементом `</body>` в файле *index.html*, использующемся по умолчанию в моем фреймворке, добавлена любопытная строка кода в виде четырех закомментированных символов.

```
<!-- c (~) -->
```

Я получил несколько писем с вопросами об этом маленьком фрагменте кода от тех, кто заметил его в нижней части исходных HTML-файлов всех (или почти во всех) сайтов, которые я разрабатывал. Он находится там потому, что я когда-то добавил его в фреймворк, и он (а также другие элементы разметки) составляет основу любого проекта, над которым я работаю.

Так что же это такое? Считайте, что я добавил своего рода «клеймо изготовителя», подобно тому, как гончар наносит свои инициалы на дно глиняного горшка.

Это изображение кружки пива в кодировке ASCII (ручка слева, внутри плещется пенный напиток). По-моему, закончить длинный шаблон HTML-кода освежающим глотком — неплохая идея, не так ли?

Это важная деталь? Конечно же, нет. Хотя для этой книги мне, наверное, следовало бы создать изображение кружки кофе или даже кофейного зерна.

```
<!-- (∫) -->
```

Можно так увлечься этой идеей, что с помощью комментариев создать в таблице стилей целый логотип, своего рода «пасхальное яйцо» (рис. 5.9). Это ж каким сумасшедшим нужно быть, чтобы написать столько лишнего кода!

(Кхе-кхе)

**СОВЕТ**

Не подумайте, что я потратил целую неделю на то, чтобы аккуратно создать логотип вручную. На самом деле для этого существует удобный бесплатный сервис на Photo2Text.com (*http://photo2text.com/*). Просто загрузите изображение, и сайт чудесным образом создаст для вас его ASCII-версию. И пусть ваши друзья думают, что вы сами сделали этот логотип.

Ладно, пойдем дальше и займемся чем-нибудь более существенным. Вот тот HTML-файл, который является основой

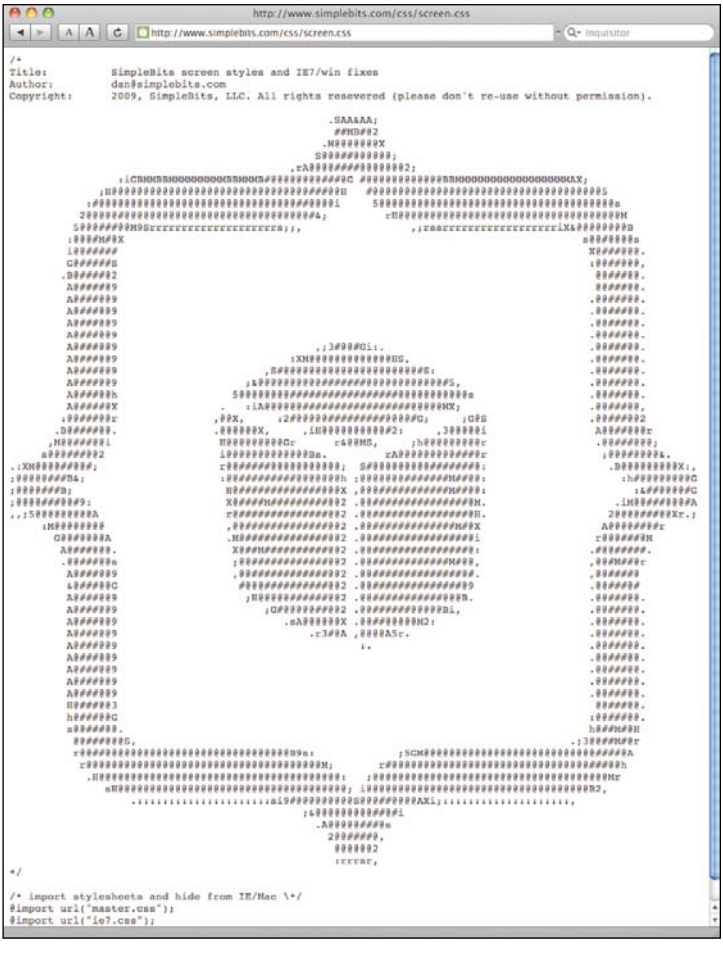



**Рис. 5.9.** Исходный файл screen.css: большое изображение логотипа в ASCII-графике, добавленное в качестве комментария

моего CSS-фреймворка. Я приведу его целиком, так что смотрите и наслаждайтесь.

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

```
<title>Untitled</title>
<!--[if gte IE 7]><!-->
<link rel="stylesheet" type="text/css"
      media="screen, projection" href="css/screen.
      css" />
<!--<! [endif]-->
</head>

<body>

<div id="wrap">

<div id="header" class="group">
   <div id="logo">
      logo
   </div>

   <ul id="nav">
      <li><a href="#">Link</a></li>
      <li><a href="#">Link</a></li>
      <li><a href="#">Link</a></li>
      <li><a href="#">Link</a></li>
   </ul>
</div> <!-- /header -->

<hr />

<div class="group">
   <div id="main">
      main
   </div> <!-- /main -->

   <hr />

   <div id="secondary">
      secondary
   </div> <!-- /secondary -->
</div> <!-- /.group -->

<hr />
```

```
<div id="footer">
   footer
</div> <!-- /footer -->

</div> <!-- /wrap -->

<!-- c(~) -->
</body>
</html>
```

Далее мы посмотрим на каждый из четырех CSS-файлов, которые я использую для создания структуры стилей на начальном этапе каждого проекта.

## SCREEN.CSS

Это основная таблица стилей, которая импортируется в исходном HTML-файле. По сути, она представляет собой всего-навсего список других таблиц стилей, которые будут использоваться. Порядок импортирования важен, и мы поговорим об этом буквально через минуту.

```
/*
Title: Screen styles and IE/Win patches
Author: dan@simplebits.com
*/

/* import stylesheets and hide from IE/Mac \*/
@import url ("reset.css");
@import url ("master.css");
@import url ("ie.css");
/* end import/hide */
```

Обязательно обратите внимание на обратный слеш в конце этой строки.

```
/* import stylesheets and hide from IE/Mac \*/
```

Этот маленький безобидный обратный слеш заставляет IE/Mac игнорировать весь последующий CSS-код до тех пор, пока не встретится другой комментарий: в нашем случае — `/* end import/hide */` (если вы вообще хотите, чтобы IE/Mac видел хоть какой-то CSS-код).

В настоящее время IE/Mac — «мертвый» браузер, брошенный на произвол судьбы ребятами из Редмонда. Так что, скорее всего, уже нет необходимости использовать этот фокус Гудини

в моих таблицах стилей. Но если все же кто-то откроет созданный мной сайт в IE/Mac, то перед ним будет просто сайт без оформления, а не дизайн, который выглядит продуманным, но почему-то не работает. Я больше не забочусь о создании оформления для IE/Mac, поэтому продолжаю использовать этот небольшой трюк.

## RESET.CSS

Первая таблица стилей, которая импортируется в файле `screen.css`, — это таблица стилей `reset`. Идея сброса стандартных стилей, которые браузеры по умолчанию применяют к HTML-элементам, принадлежит Эрику Мейеру. Так как эти стандартные стили зачастую отличаются в разных браузерах, их лучше обнулить до написания ваших собственных таблиц стилей. Это поможет вам заложить прочный фундамент для дальнейшей работы.

Еще один плюс состоит в том, что в некоторых случаях вам не придется дублировать те правила, которые должны встретиться в таблице стилей несколько раз. Так, например, работая над CSS для MTV.com, я *не* использовал *reset.css*, и напрасно. Я подсчитал, сколько раз мне пришлось использовать в основной таблице стилей правила `margin: 0;` и `padding: 0;` для сброса стандартных настроек, и результат был просто шокирующим.

```
margin: 0;     66
padding: 0;    89
```

Это означает, что в общей сложности 155 правил в основной таблице стилей не делают ничего, кроме обнуления значений отступов и полей, использующихся в браузерах по умолчанию. Слишком много повторов!

Я немного отстал в вопросе использования таблицы стилей `reset`. Но теперь я переоценил свои методы (видите, к чему я клоню?) и понял, что это совершенно необходимо. И эта таблица стилей всегда добавляется *первой*, чтобы остальные элементы оформления накладывались сверху на этот прочный фундамент.

```
/*
Title: Reset default browser styles
Author: dan@simplebits.com, based on Eric Meyer's
     Reset
```

```
CSS: http://meyerweb.com/eric/tools/css/reset
*/

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, font, 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 {
   margin: 0;
   padding: 0;
   font-size: 100%;
   vertical-align: baseline;
   border: 0;
   outline: 0;
   background: transparent;
   }

ol, ul {
   list-style: none;
   }

blockquote, q {
   quotes: none;
   }

:focus {
   outline: 0;
   }

table {
   border-collapse: collapse;
   border-spacing: 0;
   }
```

**СОВЕТ**

Файл *reset.css*, который я использую при создании всех проектов, во многом позаимствован у Эрика Мейера (*http://meyerweb.com/eric/tools/css/reset/*).

## MASTER.CSS

После сброса стандартных настроек браузера мы добавляем стили основного оформления. Основной файл, *master.css*, содержит все стили, специфические для дизайна данного сайта. Чтобы ускорить процесс создания оформления, я добавил во фреймворк кое-какие базовые элементы.

Помимо стиля и размера шрифта в этом css-файле я создал «рыбы» для основных элементов, которые предполагается добавить в разметку. У большинства сайтов есть заголовок, футер (подвал), основная область и боковая панель. С помощью комментариев мы выделяем для них место в таблице стилей, и это помогает сэкономить немного времени. Заранее заполняя комментарии, я задаю стандартный способ их оформления, а также отделяю стили друг от друга, располагая их по порядку.

Но наиболее важно то, что объявление класса `.group`, выполняющего «очистку», уже добавлено (вы найдете его внизу). Оно готово к использованию настолько, что мне не нужно даже задумываться о CSS-коде, который в нем используется; я лишь должен знать, что добавление этого класса к контейнеру «очистит» все плавающие элементы внутри него.

```
/*
Title: Master styles for screen media
Author: dan@simplebits.com
*/

body {
   font-family: "Helvetica Neue", Helvetica,
     Arial, sans-serif;
   color: #444;
   font-size: 62.5%;
   background: #fff;
   }

/* links */

a:link, a:visited {
   color: #369;
   outline: none;
   }
```

```
a:hover {
   color: #39c;
   }

/* page structure
---------------------------------- */

#wrap {

   }

#main {

   }

#secondary {

   }

#footer {

   }

/* header
---------------------------------- */

#logo {

   }

/* nav */

#nav {

   }

/* main styles
---------------------------------- */

/* secondary styles
---------------------------------- */
```

```
/* footer
---------------------------------- */

#footer {

   }

/* misc.
---------------------------------- */

hr, .hide {
   display: none;
   }
a img {
   border: none;
   }

/* self-clear floats */

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

## IE.CSS

Наконец, мы импортируем файл *ie.css*, в котором содержится весь дополнительный код для Internet Explorer. Я предпочитаю выделять эти правила в отдельную таблицу стилей, чтобы *master.css* оставался чистым и лишенным того безобразия, которое происходит из-за необходимости учитывать странности IE.

В самом фреймворке такой файл содержит не очень много правил — разве что трюк со свойством `filter`, который может (иногда) заставить IE6 правильно отображать PNG-изображения с альфа-каналом, и (что более важно) правила, касающиеся «самоочистки» плавающих элементов в IE6 и IE7.

Используя этот фреймворк в каждом проекте, я могу быть уверен в правильной работе класса `.group` во всех браузерах. Мы создали стиль и забыли об этом.

```
/*
Title: IE patches
Author: dan@simplebits.com
*/

/* PNG fix */

* html #selector {/* for IE<6 */
   filter: progid:
     DXImageTransform.Microsoft.AlphaImageLoader
     (enabled=true, sizingMethod=scale
     src='img/image.png'); background-image:
     none; background-repeat: no-repeat;
     background-color: transparent;
   }

/* self-clear floats */

* html .group {/* IE6 */
   height: 1%;
   }
*:first-child+html.group {/* IE7 */
   min-height: 1px;
   }
```

**ПРИМЕЧАНИЕ**

Помните, что IE8 поддерживает генерацию контента (`:before` и `:after`), так что для него дополнительные действия не требуются. Как и другие браузеры, IE8 будет использовать код для «очистки», содержащийся в файле *master.css*.

## ВАШ ФРЕЙМВОРК МОЖЕТ ОТЛИЧАТЬСЯ

Я только что подробно описал структуру моего небольшого фреймворка. Но это не значит, что я считаю его самым лучшим или самым продуманным; и я не утверждаю, что он подойдет любому дизайнеру в любой ситуации. Скорее я надеюсь пробудить в вас интерес в создании своего собственного фреймворка (если его у вас еще нет).

Мой фреймворк достаточно простой, и я легко могу его усовершенствовать. Но в нем есть несколько действительно важных моментов.

- Таблицу стилей `reset` нужно импортировать *первой*, чтобы создать прочный фундамент для проекта и избежать повторяющихся правил.
- Следует добавить объявления класса `.group`, выполняющего «очистку», в *master.css* и *ie.css*, чтобы при необходимости можно было быстро разместить плавающие блоки в контейнере.
- Дополнительный код для Internet Explorer следует добавить в отдельную таблицу стилей.

## Использование класса .group в шаблоне Tugboat

Итак, у нас есть фреймворк. Теперь можно поговорить и о том, где в шаблоне Tugboat используется класс `.group` для «очистки» плавающих элементов. Так мы сможем посмотреть на этот метод в действии.


**Рис. 5.10**

На рисунке 5.10 я постарался наглядно показать, где я добавил класс `.group` в уже существующую разметку. Я имею в виду,

что для «очистки» мне не потребовалось создавать *дополнительную* разметку, я просто добавил класс `.group` к элементам-контейнерам, в которых есть плавающие блоки. Я старался по возможности сохранить семантику элементов, атрибутов `id` и названий классов, зная, что `.group` позаботится о размещении плавающих блоков.

Этот пример также показывает, как добавить класс `.group` к элементу, у которого уже есть класс, задающий его оформление. Так, упорядоченный список «This Week's Specials» уже содержит класс (`<ol class="specials" >`), но я просто добавил `.group` в конец, чтобы «очистить» плавающие блоки.

## ПЕРЕМЕЩЕНИЕ МОДУЛЕЙ

Когда мы уверены в том, что плавающие элементы правильно размещены в… *контейнере*, мы неожиданно понимаем, что имеем дело с *модульной* верстkой. При необходимости эти отдельные части можно перемещать в пределах страницы и не беспокоиться о том, как это повлияет на другие области макета (рис. 5.11). Контейнеры останутся *независимыми модулями*, а плавающие элементы, использующиеся в них для гибкости, останутся на месте и не выйдут за заданные границы. Это хорошая основа для пуленепробиваемого дизайна.

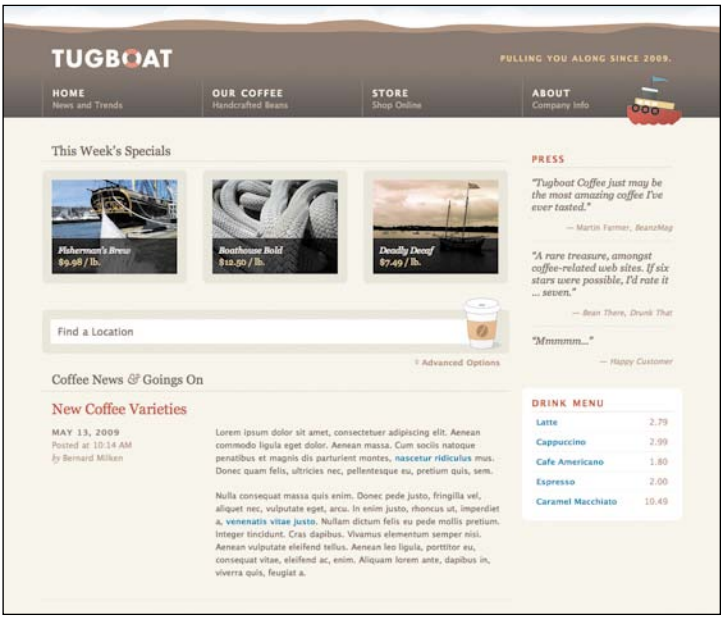



**Рис. 5.11.** Шаблон Tugboat, в котором мы кое-что поменяли местами

## В заключение

Я рассказал вам свою собственную историю о переоценке методов работы с плавающими блоками. Но я надеюсь, что вы вынесете из этой главы самое важное: переоценивать можно любые методы, которые вы используете при создании вебсайтов.

Если считать, что настоящий мастер все время задается разными вопросами, то мы правильно начали первую главу с вопроса «А что, если?..». А здесь, в этой главе, мы спросили себя: «Можно ли улучшить методы, которые я использую для создания ____?».

В эпиграфе Дэйв Гувер утверждает, что эксперты не осмелятся спрашивать себя об этом, в то время как мастерам это, наоборот, поможет пересмотреть и доработать привычные методы. Я полностью согласен и изо всех сил стремлюсь выискивать более эффективные способы достижения одной и той же цели, и я не боюсь переосмысливать и корректировать старые привычки.

# Глава 6

## «Резиновая» сетка

Итан Маркотт

Дэн Седерхольм попросил знаменитого веб-дизайнера и разработчика Итана Маркота написать для книги эту проникновенную главу.

**Где есть порядок, есть и имена.**
**Как только появляются имена,**
**Надобно знать, где в знании остановиться.**
**Кто знает, где остановиться в знании,**
**Сможет избежать большой беды.**
**Великий Путь для мира**
**Все равно что полноводная река и море для ручья.**

Дао дэ Цзин. Перевод В. Малявинова

В последние годы возродился интерес к использованию сетки в дизайне и, в частности, в веб-дизайне. Оказалось, что эта идея совсем не нова: она пришла к нам из мира печатной типографики, где использовалась еще в начале XX века. В то время дизайнеры-модернисты считали, что печатная типографика должна вернуться к оформлению, ориентированному на текст, без бессмысленных рисунков и прочих сомнительных излишеств. Они утверждали, что такие украшения просто, если можно так выразиться, стоят поперек дороги — все эти красивые картинки мешают дизайну эффективно взаимодействовать с пользователем.

Развитие этих идей достигло кульминации, когда Йозеф Мюллер-Брокман написал книгу «Модульные системы» («Rastersysteme für die visuelle Gestaltung»). В ней он попытался показать, что основой любого макета страницы должна быть жесткая модульная система. Преимущества этой идеи были вполне логичны: такая структура из строк и столбцов берет на себя часть работы по исключению субъективизма со стороны дизайнера, и результат становится гораздо более понятным. Хорошим примером такого дизайна, использующего геометрические фигуры и жирные шрифты, можно считать концертные афиши цюрихского оркестра Тонхалле, относящиеся к числу наиболее известных работ самого Мюллера-Брокмана (рис. 6.1).


**Рис. 6.1**

Перемотав пленку времени на несколько десятилетий вперед, мы обнаружим молодую область веб-дизайна в такой же ситуации. В первые годы существования этой индустрии из-за относительной новизны технических средств (а также неадекватности браузеров, под которые создавались проекты) многие из этих основополагающих принципов были роскошью, которую мы, дизайнеры, не могли себе позволить. Половина времени уходила на настройку соединения через телефонную линию, вторая половина — на поиск четырех способов написания кода для одного веб-сайта так, чтобы результат хорошо отображался во всех тогдашних браузерах. Если это кажется забавным, то отчасти так оно и было, но еще в этом было много сумбура и мазохизма.

Так как браузеры совершенствовались и улучшалась поддержка веб-стандартов, безумие уступило место поискам и экспериментам. На передний план вышли такие сайты, как «Сад CSS-Дзен» (CSS Zen Garden), и армия скучающих по стандартам дизайнеров ринулась в бой, подняв на знамена фильтры Photoshop.

Затем в конце 2004 года Кхои Винху (рис. 6.2) и Марк Балтон (рис. 6.3) стали говорить о том, что дизайнерам стоит вернуться к основам, а именно к использованию типографской сетки — объекта из мира печати, почти совершенно чуждого веб-дизайну.

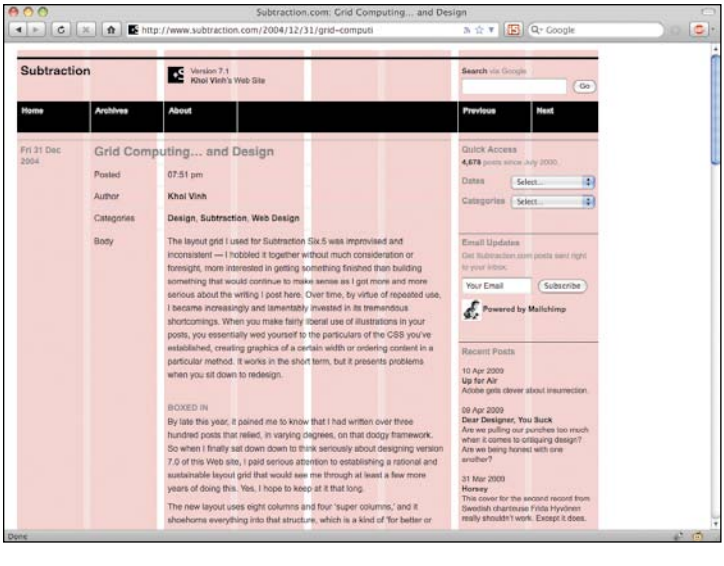



**Рис. 6.2**

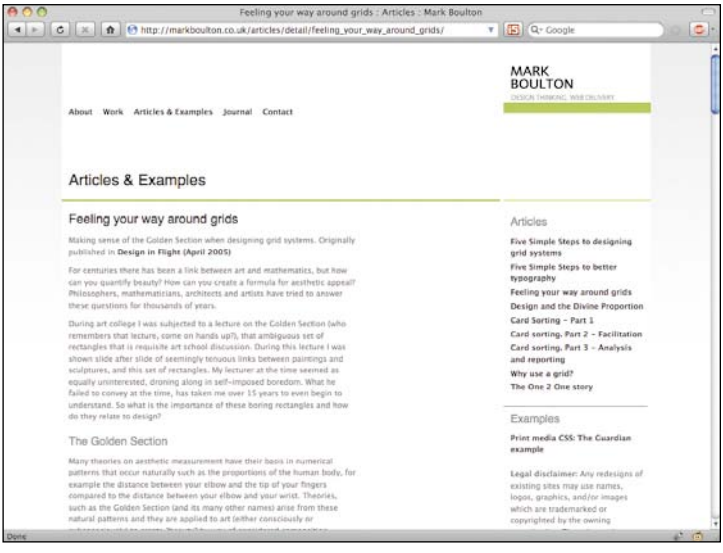



**Рис. 6.3**

Сообщество веб-дизайнеров воспользовалось этой идеей, и сетка вскоре стала пользоваться колоссальным успехом. Ресурсы вроде The Grid System (*http://www.thegridsystem.org/*) возникли буквально за считанные дни. Было создано множество CSS-фреймворков, таких как Blueprint (*http://www.blueprintcss.org/*) и YUI Grids (*http://developer.yahoo.com/yui/grids/*), делающих работу с сетками гораздо более удобной. Похоже, что теперь в Сети господствует рациональный подход, благодаря чему в выигрыше оказываются и дизайнеры, *и* пользователи. Не так ли?

Только есть одна небольшая проблема. Большинство таких сеток имеют фиксированную пиксельную ширину.

## Крупный заказ

Около двух лет назад в компанию, где я работал, обратился Консорциум Всемирной паутины (*http://www.w3.org/*) и попросил разработать новый дизайн для своего сайта. Это был долгий и сложный проект, отчасти из-за огромного объема контента, который нужно было разместить на сайте и далее обслуживать.

Требования заказчика к дизайну были, в общем-то, минимальными. Поскольку веб-сайт W3C содержит очень много информации, нас попросили в первую очередь обратить внимание на удобочитаемость. Так что мы решили не захламлять

интерфейс лишними элементами дизайна, а приложить все усилия для того, чтобы оформление наглядно отражало иерархическую структуру информации. И после нескольких месяцев работы и активных обсуждений с заказчиком был создан новый минималистичный дизайн (рис. 6.4).

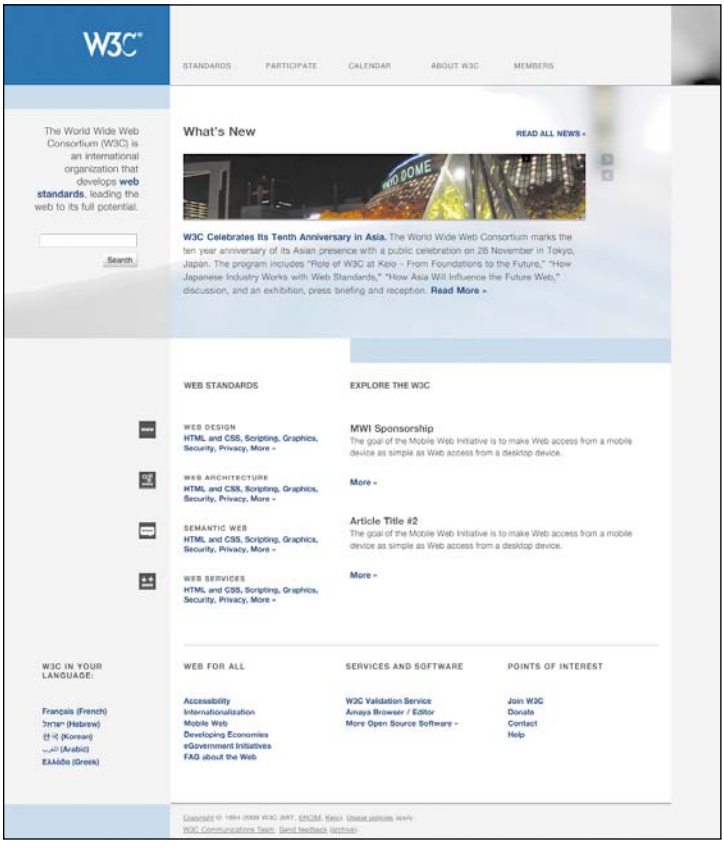



**Рис. 6.4**

Как видите, сетка, на которой мы в конечном итоге остановили свой выбор (рис. 6.5), четко разбивает страницу на восемь столбцов с ровными, вымеренными интервалами между ними. Для дизайнера, как и во времена Мюллера-Брокмана, сетка является бесценным инструментом: столбцы образуют структуру, которая позволяет расположить различные элементы на странице и показать их взаимосвязь друг с другом. Для пользователей сетка облегчает чтение, формируя систему естественных линий, и помогает лучше «переварить» то огромное количество информации, которое содержится на сайте заказчика.


**Рис. 6.5**

Второе требование к сайту состояло в том, чтобы дизайн имел переменную, а не фиксированную ширину. Это означало, что при уменьшении или увеличении размера окна масштаб должен автоматически меняться соответствующим образом. И тогда, как это обычно и происходило, возможность использовать «резиновую» верстку заставила мое сердце затрепетать — я помешан на дизайнах нефиксированной ширины с тех самых пор, как начал работать в Сети.

Почему я такой фанат «резиновой» верстки? Что ж, если хотите, я расскажу вам короткую историю.

## ПРОБЛЕМА С ФИКСИРОВАННОЙ ШИРИНОЙ

Одна моя знакомая работает в крупном научно-исследовательском учреждении. Она веб-разработчик; среди тех, кого я когда-либо встречал, вряд ли найдется много людей, разбирающегося в вопросах IT лучше нее. Она знает консоль UNIX как свои пять пальцев, помогает ученым работать с ПО, анализирующим данные со спутников (нет, я не выдумываю!), и знает мельчайшие тонкости работы с Perl-модулями. В свободное время она разгадывает кроссворды, на кухне — слушает подкасты, а также любит читать блоги, Твиттер и газеты с экрана ноутбука.

Вот в этом-то все и дело: ее ноутбуку уже четыре или пять лет — модель, снятая с производства. Он, безусловно, хорошо работает, особенно после увеличения объема памяти и смены жесткого диска. Но кое-что заменить невозможно: экран

и видеокарту. В итоге от ее монитора нельзя добиться разрешения выше, чем 1034×768.

Это ее совсем не беспокоит, хотя я всегда считал ее помешанной на таких вещах. Но когда я впервые увидел, как она читает одну из своих любимых интернет-газет, я заметил кое-что интересное (рис. 6.6).


**Рис. 6.6**

Видите, она оставляет слева немного места для часто используемых программ, а также небольшую полоску справа, чтобы был виден рабочий стол (на случай, если ей понадобится какой-нибудь файл). Поэтому, хоть экран и настроен на определенное разрешение, но размеры окна браузера с ним не совпадают.

В результате сайты, рассчитанные на ширину экрана около 1024 пикселов, не полностью помещаются в ее окне. Правый столбец обычно оказывается обрезанным, и часто в поле зрения не попадает все самое интересное, особенно на страницах с большим количеством контента. У нее даже есть забавная привычка: она прокручивает страницу чуть-чуть вниз, потом вправо, потом опять влево, а потом опять вниз. Она зигзагом (точнее, буквой L) перемещается по странице, и это происходит не потому, что какая-то часть контента не видна, а потому, что она *может* быть не видна. Но узнать, что конкретно не попадает в поле зрения, можно, только передвинув ползунок полосы прокрутки.

Проблема интерфейсов фиксированной ширины состоит вот в чем: такой подход предполагает, что пользователь должен адаптироваться к дизайну, а не наоборот. Моей знакомой (как и многим другим) приходится менять свои привычки, чтобы приспособиться к сайту; в противном случае часть контента останется вне поля зрения.

Верстка фиксированной ширины доставляет неудобства не только пользователям: пострадать может и владелец сайта (рис. 6.7).


**Рис. 6.7**

Крайняя справа ссылка, кусок которой виден на этом скриншоте (*Register Now*), видимо, очень важна для коммерческого успеха газеты: новые пользователи, больше подписчиков на различные услуги, больше щелчков мыши на рекламных объявлениях. Но ее расположение на странице довольно нестабильно — если разрешение экрана пользователя будет чуть-чуть ниже минимального разрешения для данного сайта, такой призыв к действию не будет услышан.

Часть проблемы, кстати, заключается в самом словосочетании «минимальное разрешение экрана». Дизайн фиксированной ширины базируется на этих трех словах, которые задают для пользователей воображаемый нижний предел. И мы создаем веб-сайты фиксированной ширины, а потом раз в несколько лет переделываем их, поднимая планку на один или два размера экрана. Если возможности наших пользователей и окна их браузеров соответствуют этой планке, то все хорошо; но если нет, придется использовать полосу прокрутки.

## БОЛЬШЕ РЕШЕНИЙ, МЕНЬШЕ ПЕНЫ!

Как известно, слова сами по себе почти ничего не стоят. И когда я работал над новым дизайном для W3C, я не мог позволить себе писать длинные тирады на тему враждебного отношения пользователей к интерфейсам фиксированной ширины. Вместо этого я выяснил, почему большинство интерфейсов, использующих

сетку, имеет фиксированную ширину. Оказывается, дизайнеры убеждены, что сверстать макет с большим числом столбцов очень просто, если явно задать ширину в пикселах. Поэтому мне предстояло понять, можно ли соединить жесткость дизайнов, использующих сетку, с гибкостью, присущей веб-дизайну в целом. Возможны ли вообще «резиновые сетки»?

Оказывается, возможны. «Резиновые» сетки — всего лишь вопрос *контекста*.

## Гибкость за счет... размера шрифта?

В конце концов, решение пришло ко мне, когда я работал над другой составляющей гибкого дизайна: шрифтовым оформлением. Чтобы лучше понять, что такое гибкая верстка, возьмем несколько элементов дизайна Дэна (из шаблона Tugboat) и немного поупражняемся с размерами шрифтов. К примеру, представим, что Дэн попросил нас написать код для простого фрагмента, в котором нет ничего, кроме названия раздела, а также заголовка и одного абзаца текста из блога (рис. 6.8).


**Рис. 6.8**

Перед нами наитипичнейший дизайн: простой, четкий, *изящный*. А поскольку я не очень-то уверен в значении слова «наитипичнейший», я бы назвал разметку для этого раздела достаточно прямолинейной. Для основной области контента используется тег `<div>`, как и для фрагмента блога, а для остального текста

разумно используются теги <h2>, <h3> и <p>. Такая разметка лучше всего отражает смысловое содержание контента.

```
<div class="main">
   <h2>Coffee News <span class="amp">&amp;
     </span> Goings On</h2>

   <div class="entry">
      <h3>New Study Says Coffee "Good"</h3>

      <div class="entry-body">
         <p>Lorem ipsum dolor sit amet,
            consectetuer…</p>
      </div><!-- /entry-body -->
   </div><!-- /entry -->
</div><!-- /main -->
```

Корректный, семантически богатый и хорошо структурированный код — что ж, хорошее начало. Теперь, когда фундамент заложен, добавим немного CSS к нашим безликим угловым скобкам.

```
body {
   font-size: 100%;
   font-family: "Lucida Grande", "Lucida Sans
      Unicode", "Lucida Sans", Helvetica, Arial,
      sans-serif;
   font-weight: normal;
   background: #F3F2E8;
   color: #51463D;
}
```

**СОВЕТ**

Здесь для задания шрифта используется три отдельных свойства (`font-size`, `font-family` и `font-weight`). Но я хочу напомнить, что существует универсальное свойство `font`, позволяющее вместо трех строк написать одну: `font: normal 100% "Lucida Grande", "Lucida Sans Unicode", "Lucida Sans", Helvetica, Arial, sans-serif;`

С помощью свойства `font-family` мы указываем, что браузер должен по умолчанию использовать шрифт Lucida Grande, если он доступен; если нет, браузер будет использовать одно из дополнительных значений, которые мы тоже задали. Кроме того, мы применили к элементу `body` свойство `background`

со значением `#F3F2E8` и задали цвет текста по умолчанию (`#51463D`).

Наконец, мы задали для свойства `font-size` значение `100%`. Почему `100%`? Так мы просто задаем тот размер шрифта, который используется в браузере по умолчанию (обычно это 16 пикселов). Поэтому если пользователь решит в настройках браузера сделать размер шрифта чуть больше (рис. 6.9), это значение тоже увеличится. Но для наших упражнений по заданию шрифтов это полезно: таким образом, мы получаем надежное базовое значение, относительно которого будем вычислять размер шрифта для остальных элементов.

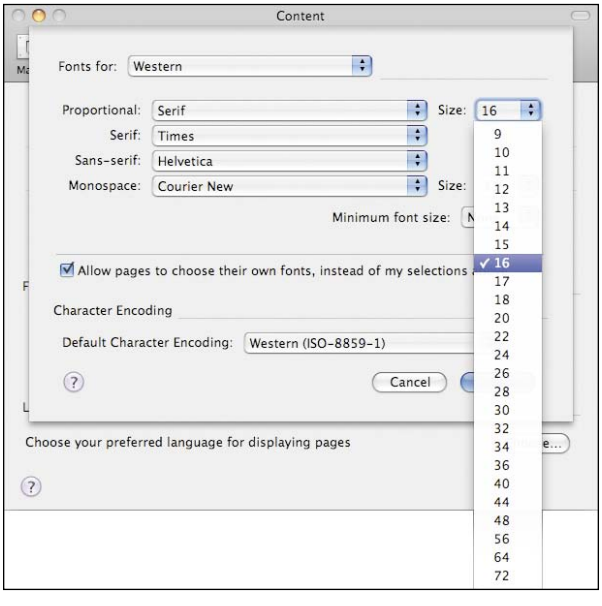



**Рис. 6.9**

Как видите, с помощью одного этого правила мы добились неплохих результатов (рис. 6.10).

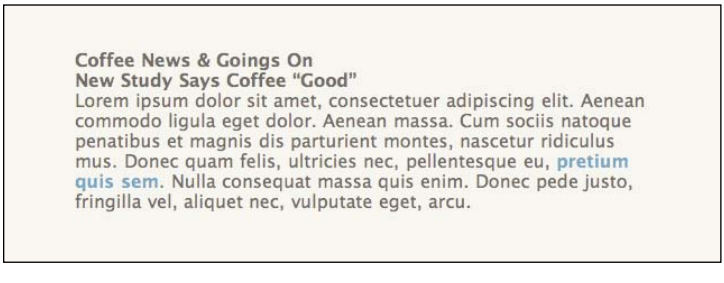



**Рис. 6.10**

Как говорил Дэн во введении к этой книге, файл *reset.css* нужен нам для того, чтобы обнулить внутренние таблицы стилей браузера и отменить правила оформления, которые используются в нем для элементов HTML. Вот почему наши заголовки выглядят, если можно так выразиться, «незаголовочно». Пока что для текста во всем нашем образце используется одно и то же значение `font-size` (а именно `100%`), которое наследуется от элемента `body`. Так как в большинстве случаев это базовое значение равно 16 пикселам, мы можем использовать его в качестве отправной точки, когда будем работать с относительными размерами шрифтов.

## ВОПРОС КОНТЕКСТА

Давайте повнимательнее посмотрим на наш образец и поймем, какие начертания и размеры шрифтов следует использовать для нашего текста (рис. 6.11).

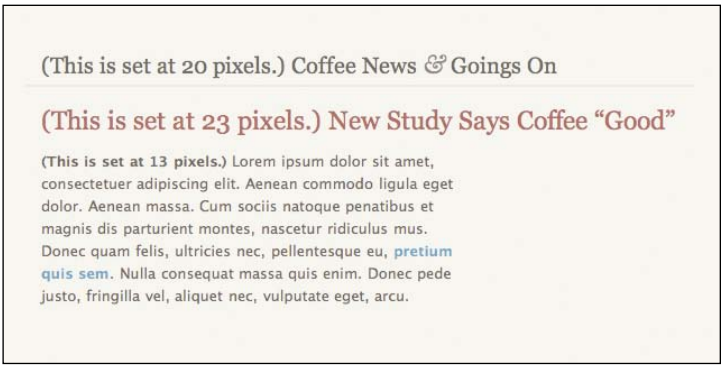



**Рис. 6.11**

С точки зрения типографики наш результат достаточно прямолинеен. Самый верхний заголовок и название блога написаны шрифтом Georgia, размер — 20 и 23 пиксела соответственно. А для основного текста по умолчанию используется размер 13 пикселов. Довольно просто, не так ли?

**СОВЕТ**

Я никогда не умел так ловко обращаться с числами — во всем виновата ученая степень по литературе. Для всех вычислений здесь и далее я просто использовал встроенную в OS X программу Калькулятор. Не стесняйтесь использовать любые похожие программы. Да что там, подойдут даже счеты!

Как раз здесь в игру вступают относительные единицы `em`. А это значит, что речь идет о *контексте*. Иными словами, фактический размер единицы `em` элемента вычисляется относительно значения `font-size` родителя этого элемента. Таким образом, чтобы посчитать искомое значение размера шрифта, мы должны разделить нужное значение в пикселах на размер шрифта контейнера (то есть его контекст). В результате мы получим тот размер шрифта, который мы хотели задать, только в относительных, масштабируемых, удивительных единицах `em`.

нужное значение ÷ контекст = результат

Имея в распоряжении такую формулу, вернемся к нашему заголовку. Мы знаем, что нам нужен шрифт 20px Georgia. Задать размер в пикселах было бы довольно просто.

```
.main h2 {
   font: normal 20px Georgia, serif;
}
```

Однако `20px` — это только *нужное* значение. Если предположить, что для элемента `body` (то есть *контекста*) используется размер шрифта `16px`, то мы можем подставить эти два числа в формулу.

20 ÷ 16 = 1.25

Вот что мы получили: шрифт заголовка в 1,25 раз больше, чем шрифт, использующийся по умолчанию для элемента `body`. То есть его размер — `1.25em`, и это значение можно подставить прямо в правило таблицы стилей.

```
.main h2 {
   font: normal 1.25em Georgia, serif;
   /* 20px / 16px = 1.25em */
}
```

Теперь шрифтовое оформление заголовка приятно совпадает с тем, что изначально планировалось; и к тому же оно может быть изменено пользователем (рис. 6.12). Чем дальше, тем лучше.


**Рис. 6.12**

Но и это еще не все. О заголовке блога и его основном тексте тоже нужно позаботиться, используя единицы `em`. К счастью, здесь мы тоже можем прибегнуть к формуле «нужное значение ÷ контекст = результат». Так как мы планируем задать для заголовка шрифт 23px Georgia, просто подставим новое нужное значение в нашу формулу.

$$23 \div 16 = 1.4375$$

А для нашего одинокого абзаца с размером шрифта 13px результат мы вычислим так.

$$13 \div 16 = 0.8125$$

Да, многовато знаков после запятой. Тем не менее именно эти значения мы хотели получить: заголовок блога в 1,4375 раз больше, чем шрифт в элементе `body`; основной текст блога меньше этого шрифта, и коэффициент равен 0,8125. Эти относительные значения хорошо впишутся в нашу таблицу стилей.

```
div. entry h3 {
   font: normal 1.4375em/1.3 Georgia, serif;
   /* 23px / 16px = 1.4375em */
}

div. entry-body {
   font-size: 0.8125em;
   /* 13px / 16px = 0.8125 */
   line-height: 1.6;
}
```

Еще раз напомню, что для этих элементов размер шрифта задается относительно размера шрифта их контейнера (в данном случае элемента `body`, для которого мы определили свойство `font: 100%`). И теперь, когда мы это сделали, шрифт всех трех элементов наконец-то имеет нужный размер (рис. 6.13).


**Рис. 6.13**

Стоп: амперсанд выглядит как-то не так. В макете Дэна (рис. 6.14) он, пожалуй, гораздо красивее.

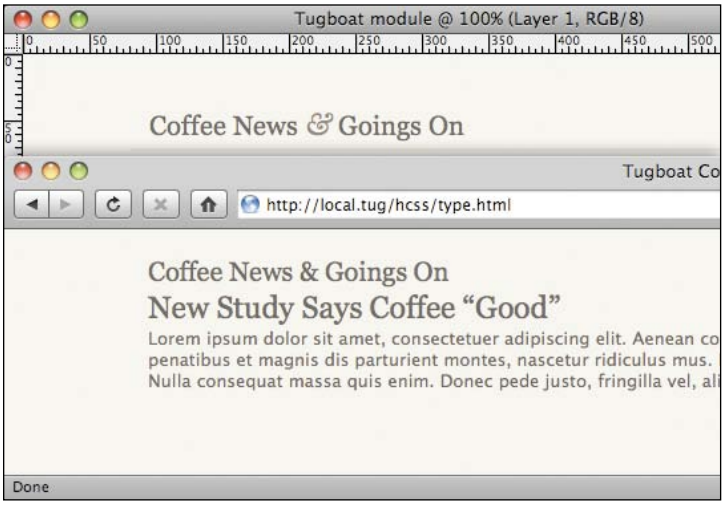



**Рис. 6.14**

Дэн использовал курсив роскошного шрифта Baskerville (размер 24px), так что этот знак элегантно выделяется на фоне остальной части заголовка. А наш амперсанд просто наследует шрифт 23px Georgia от элемента `h1`. Получается, над заданием шрифтового оформления нам стоит еще поработать.

## СМЕНА КОНТЕКСТА

Итак, Baskerville размером 24 пиксела? Что ж, если бы мы просто переносили в код пиксельные значения из нашего образца, мы бы написали примерно такую таблицу стилей.

```
span.amp {
   font: normal 24px Baskerville, "Goudy Old
     Style", "Palatino", "Book Antiqua", serif;
   font-style: italic;
   color: #766557;
}
```

Но, как и в остальных случаях, мы хотим задать размер шрифта в относительных единицах, не используя фиксированных пиксельных значений. Так что вспомним снова нашу проверенную формулу «нужное значение ÷ контекст = результат».

Но есть одна важная деталь, которая отличает данную ситуацию от предыдущей. Раньше контекстом для всех наших элементов было значение 16px (размер шрифта элемента `body`).

Теперь же амперсанд, для которого мы хотим создать стиль, находится *внутри* другого элемента.

```
<h2>Coffee News <span class="amp">&amp;</span>
     Goings On</h2>
```

Если бы `<h2>` не содержал явного значения `font-size`, мы могли бы подставить в формулу 16px. Но так как мы задали размер шрифта для заголовка, контекст изменился, и поэтому размер шрифта для всех дочерних элементов должен вычисляться относительно значения `1.25em`, то есть `20px`.

Так что в нашу формулу в качестве нужного значения мы подставим `24px`, а в качестве контекста — `20px`.

$$24 \div 20 = 1.2$$

И вот что мы получили: амперсанд должен быть в 1,2 раз больше, чем шрифт содержащего его заголовка (то есть `1.2em`). Это значение можно добавить прямо в правило `span.amp`.

```
span.amp {
   font: normal 1.2em Baskerville, "Goudy Old
     Style", "Palatino", "Book Antiqua", serif;
   /* 24px / 20px = 1.2em */
   font-style: italic;
   color: #766557;
}
```

Теперь, наконец, все наши элементы выглядят так, как мы хотели: шрифт имеет правильный размер и начертание (рис. 6.15).


**Рис. 6.15**

Итак, мы добавили последнее украшение, и теперь перед нами текст правильного размера и начертания. Немного наведем порядок (где-то добавим чуть-чуть цвета, где-то зададим межстрочный интервал), и результат будет наконец-то совпадать с замыслом Дэна (рис. 6.16).


**Рис. 6.16**

Таким образом, наш текстовый блок наконец-то готов. Шрифт задан превосходно: использование относительных единиц делает его чувствительным к изменению размера в настройках браузера.

## ПИЛИТЕ, ШУРА, ПИЛИТЕ

Вы, наверное, недоумеваете, где ваш друг Седерхольм откопал того идиота, который пишет эту главу. Ведь предполагалось, что разговор пойдет о *макетах страниц*. Так какого черта он болтает про относительный размер шрифта? (И вообще, похоже, ему не стоит доверять: он наверняка украл пару баксов из денег, предназначенных на мелкие расходы. Позовите охрану.)

Но именно в тот момент, когда я работал над шрифтовым оформлением с использованием единиц `em` для нового сайта W3C, меня осенило: каждая составляющая решетки, а также всех элементов

внутри нее, может быть представлена в виде пропорции относительно соответствующего контейнера. Как и размер шрифта, размер любого элемента нужно рассматривать только с точки зрения *соотношения этого размера с размером контейнера элемента*. Поэтому мы не будем задавать ширину столбцов в жестких единицах — пикселах. Вместо этого на основе пиксельных измерений мы вычислим процентные значения, и тогда сетка будет сохранять свои пропорции при изменении размера.

Иными словами, мы получим «резиновую» сетку. Ну что ж, попробуем.

## Больше «резины», больше сеток, больше радости

Итак, мы хотим создать гибкую, нефиксированную сетку. Начнем с еще одного упрощенного фрагмента шаблона Tugboat (рис. 6.17). Здесь мы всего лишь чуть-чуть увеличили объем работ: добавили боковую панель справа и «подвал» внизу страницы; слева под заголовком блога мы добавили немного метаинформации, а сам текст блога удобно разместили правее.


**Рис. 6.17**

К счастью, разметку не придется сильно менять. Мы можем воспользоваться HTML-формой из предыдущего примера (в котором мы занимались шрифтами), добавив в нее неупорядоченный список с метаинформацией о записи блога (`<ul class=`

”entry-meta”>). Дополнительно придется создать «содержащие» блоки для боковой панели ( ) и подвала ( ). И все это мы заключим в тег .

```
<div id="wrap" class="group">

   <div class="main">
      <h2>Coffee News <span class="amp">&amp;
     </span> Goings On</h2>
      <div class="entry last group">
         <h3>New Study Says Coffee "Good"</h3>
         <ul class="entry-meta">
            <li><h4>January 10, 2009</h4></li>
            <li>Posted at 10:14 <abbr>AM</abbr>
               </li>
            <li><span class="by">by</span> Dunder
               Shellfish</li>
         </ul>
         <div class="entry-body">
            <p>Lorem ipsum dolor sit amet,
              consectetur…</p>
         </div><!-- /entry-body -->
      </div><!-- /entry -->
   </div><!-- /main -->
   <div class="secondary">
      <div class="mod alt">
         <h3>Press</h3>
         <ul class="press">
            <li>
               <blockquote>
                  <p>"Tugboat Coffee just may be
                    the most amazing coffee I've
                    ever tasted."
                    </p>
                  <p class="author">— Martin
                    Farmer, <cite>BeanzMag</
                    cite></p>
               </blockquote>
            </li>
```

```
            <li>…</li>
          </ul>
        </div>
    </div> <!-- /secondary -->
    <div id="footer">
        Copyright © 2009. Tugboat Coffee is a
         fictitious demonstration, handcrafted by
         <a href="http://simplebits.com/">SimpleBits
         </a>.
    </div> <!-- /footer -->
</div> <!-- /wrap -->
```

Пока мы выполняем это небольшое упражнение по дизайну, давайте представим, что все трудные проблемы со шрифтами уже решены. В нескольких последующих разделах мы будем считать, что все шрифты находятся на своих местах, все линии аккуратно проведены, и единственное, с чем нам осталось разобраться, — это верстка. В результате наш дизайн выглядит очень красивым по своему шрифтовому оформлению, но все же чересчур линейным (рис. 6.18).

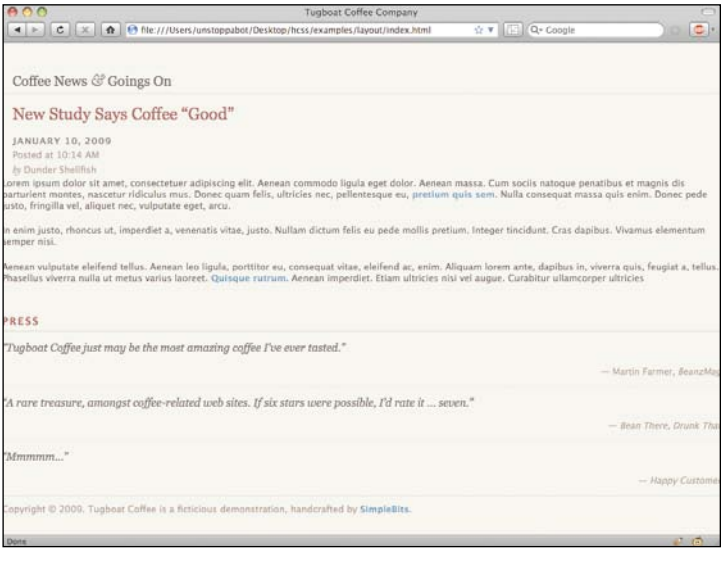



**Рис. 6.18**

Чтобы создать нашу первую «резиновую» сетку, давайте повнимательнее посмотрим на дизайн.

## ОТ МАКЕТА К РАЗМЕТКЕ

В общем-то, в качестве основы своего дизайна Дэн использует очень простую сетку. В соответствии с макетом страница имеет ширину 1000 пикселов и поделена на четыре равных столбца по 250 пикселов (рис. 6.19). Необычность этой сетки состоит лишь в том, что между столбцами нет промежутков, однако это не мешает дизайну выглядеть *хорошо*.


**Рис. 6.19**

Продолжая анализ визуального оформления, заметим, что большие блоки контента (которые в разметке заключены в теги `<div class="main">` и `<div class="secondary">`) удобно распределяются по этим четырем столбцам. Боковая панель закрепляется справа и занимает один столбец шириной 250 пикселов. А так как в центре внимания должен быть элемент `<div class="main">`, в сетке Дэн выделил для него больше всего места. Он занимает три столбца, за вычетом поля шириной 20 пикселов, которое Дэн разумно добавил между ним и боковой панелью. В итоге под основную область контента отводится 730 пикселов (рис. 6.20).


**Рис. 6.20**

Если бы мы хотели создать очередной сайт фиксированной ширины, мы могли бы очень быстро расставить все по местам с помощью трех простых правил.

```
#wrap {
   margin: 0 auto;
   padding: 40px 0 0 0;
   width: 1000px;
}

.main {
   float: left;
   width: 730px;
}

.secondary {
   float: right;
   width: 250px;
}
```

Для блока-контейнера `#wrap` мы задали ширину `1000px`, а также выровняли его по центру с помощью правила `margin: 0 auto;`. Боковую панель `<div class="secondary">` шириной `250px` мы расположили внутри контейнера у левого края. Для блока `.main`, в соответствии с требованиями макета, мы задали ширину `730px`, расположив его у левого края.

Так что если бы нас интересовали пиксельные значения, верстка была бы уже почти готова. Но фиксированные значения ширины — это не то, чего мы хотели. Даже если размер окна — 1024×768, стоит чуть-чуть его уменьшить, и появится полоса прокрутки (рис. 6.21). Поэтому мы будем задавать эти значения в *процентах*, а не в пикселах. Но как это сделать?

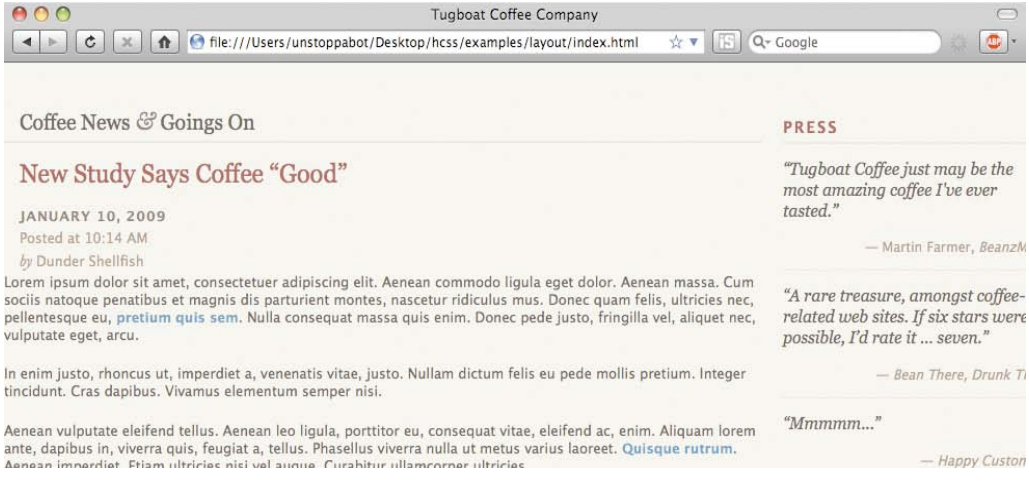



**Рис. 6.21**

## ДЕЖА ВЮ: ЗНАК ДЕЛЕНИЯ

Именно! Вернемся к нашему старому другу — формуле «нужное значение ÷ контекст = результат». Но использовать ее мы будем не для вычисления размеров шрифта, а для того, чтобы перевести пиксельные значения ширины столбцов в процентные (и за счет этого добиться *гибкости*). Так мы получим дизайн, использующий сетку, но основанный на пропорциях, а не пикселах. И эти пропорции будут менять масштаб вместе с окнами пользователей.

### Переделываем wrap

Начнем с нашего элемента-контейнера `#wrap`, название которого говорит само за себя.

```
#wrap {
   margin: 0 auto;
   padding: 40px 0 0 0;
   width: 1000px;
}
```

Вместо того чтобы задавать ширину элемента `#wrap` в явном виде, будем использовать свойство `max-width`. На мой взгляд, `max-width` — отличное решение для дизайнера с гибким мышлением: если размер окна не соответствует «минимальному разрешению экрана», страница *уменьшается*; при этом дизайнер (то есть вы) сам задает «оптимальное» значение ширины, чтобы строки текста не оказались слишком длинными.

Теперь мы можем легко задать значение свойства `max-width` в пикселах.

```
#wrap {
   margin: 0 auto;
   padding: 40px 0 0 0;
   max-width: 1000px;
}
```

Однако на больших дисплеях такое решение будет выглядеть странно. Просматривая такой дизайн на очень большом мониторе, мы обнаружим вокруг него много свободного, пустого места. И хотя, возможно, это не играет *колоссальной* роли в вопросе удобства, существует другой, более совершенный способ: вместо пикселов воспользоваться помощью нашего доброго друга, единицы `em`.

**ПРИМЕЧАНИЕ**

Некоторые дизайнеры, уделяющие особое внимание общедоступности, считают свойство `max-width` спорным. Они утверждают, что не существует «идеальной» длины строки, и поэтому дизайнер не должен никак ограничивать пользователя (*см. http://projectcerbera.com/web/articles/line-lengths и http://accessifyforum.com/viewtopic.php?p=65693#65693*). Право выбора я оставляю за читателем, но хочу уточнить: наш дизайн будет прекрасно работать без `max-width` — метод «резиновой» сетки вообще не использует это свойство.

Так же как и в нашем примере со шрифтами, мы можем выразить ширину не в пикселах (с помощью свойства `max-width` со значением `1000px`), а в относительных единицах `em`. Как и раньше, нам нужно всего лишь посмотреть на контекст, в котором мы работаем. Помните `font-size` для элемента `body`?

```
body {
   font-size: 100%;
   font-family: "Lucida Grande", "Lucida Sans
      Unicode", "Lucida Sans",    Helvetica,
      Arial, sans-serif;
   font-weight: normal;
   background: #F3F2E8;
   color: #51463D;
}
```

**ПРИМЕЧАНИЕ**

Многие современные браузеры (такие как Firefox 3, Opera и Safari) позволяют изменять масштаб всей страницы (эта возможность впервые появилась в Opera). Вместо того чтобы увеличивать только размер шрифтов, эти браузеры меняют масштаб всего дизайна — изображений, мультимедиа и всего остального — естественно, вместе с текстом. Эта новая возможность есть даже в Internet Explorer 8! Так что, хоть свойство max-width с относительным значением уже и не является необходимым для самых новых и самых лучших браузеров, в распоряжении пользователей все же оказывается дополнительный способ контролировать масштаб страницы, что особенно важно в случае чуть более старых браузеров.

Как вы уже знаете, `font-size: 100%;` в большинстве браузеров примерно соответствует значению ширины 16 пикселов. Поэтому возьмем в качестве *нужного значения* для элемента `#wrap` значение `1000px`, разделим его на *контекст*

(16 пикселов), и тогда мы получим *результат* в относительных единицах em.

$$1000 \div 16 = 62.5$$

Ну вот: нам удалось преобразовать значение 1000px свойства max-width в 62.5em. Последнее можно аккуратно добавить в правило #wrap.

```
#wrap {
   margin: 0 auto;
   padding: 40px 0 0 0;
   max-width: 62.5em;
}
```

Итак, мы добавили свойство max-width в относительных единицах em. И теперь пользователь может увеличить размер страницы, просто выбрав больший шрифт в настройках браузера, а при высоком разрешении экрана это очень удобно. Конечно же, так как мы не задали явного значения ширины для контейнера #wrap, страница автоматически изменит свой размер при просмотре на маленьком дисплее. Ура «резиновым» контейнерам!

### Извне вовнутрь

Теперь, когда мы избавились от пиксельной ширины элемента #wrap, займемся двумя контейнерами, расположенными на один уровень ниже: .main и .secondary.

```
.main {
   float: left;
   width: 730px;
}

.secondary {
   float: right;
   width: 250px;
}
```

Хотя мы уже задали значение свойства max-width для элемента #wrap в гибких единицах em, мы все еще можем опираться на данные из макета Дэна. Мы знаем, что ширина блока .main — 730px, а содержащего его контейнера — 1000px. Если подставить эти значения в формулу «нужное значение ÷ контекст = результат», то мы получим гибкое значение ширины.

730 ÷ 1000 =.73

Итак, `.73` (или `73%`) — ширина блока `.main`. Сделаем то же самое для блока `.secondary`?

250 ÷ 1000 =.25

Вуаля! Если передвинуть запятую на пару знаков, мы получим значение ширины `25%`. Теперь давайте внесем соответствующие изменения в таблицы стилей.

```
.main {
   float: left;
   width: 73%;
}

.secondary {
   float: right;
   width: 25%;
}
```

Теперь у нас есть все, что нужно для настоящей «резиновой» сетки (рис. 6.22). Так как мы задали ширину наших двух столбцов в процентах, пропорции всегда будут сохраняться, даже когда дизайн будет подстраиваться под размер окна браузера.

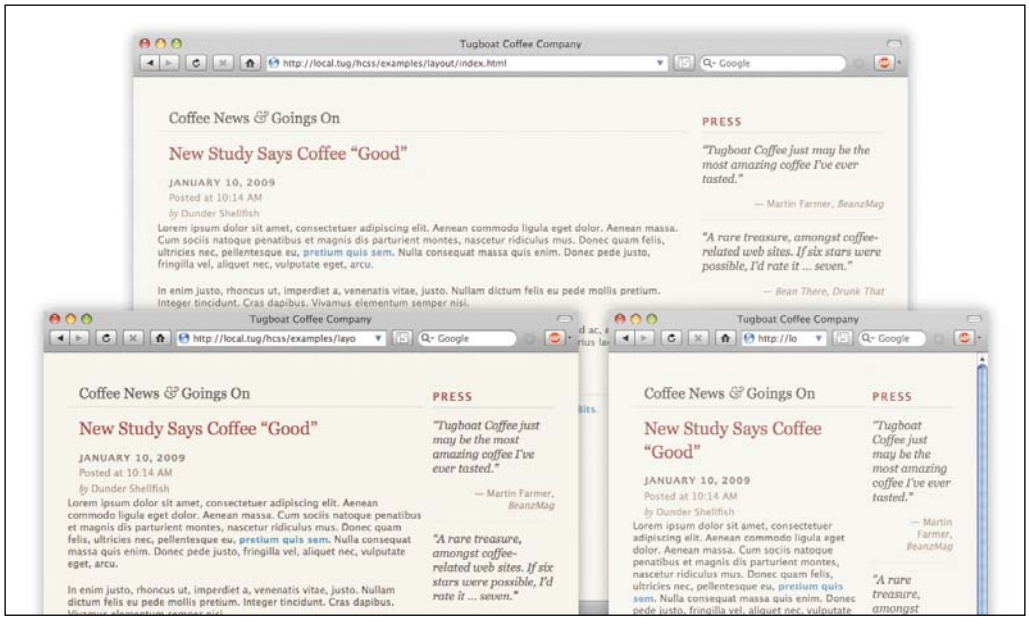



**Рис. 6.22**

## СТОЛБЦЫ, КОНТЕКСТ И ИЗМЕНЕНИЯ — ПОДУМАТЬ ТОЛЬКО!

На самом верхнем уровне наша «резиновая» сетка выглядит отлично: для контейнера `#wrap` значение `max-width` задано с помощью относительных единиц `em`; внутри него располагаются два столбца, размер которых зависит от размеров контейнера. Чтобы закончить дизайн, обратим внимание на фрагмент блога и список метаданных, который был добавлен слева.

Как и раньше, давайте повнимательнее посмотрим на макет (рис. 6.23) и попытаемся понять, что необходимо сделать.

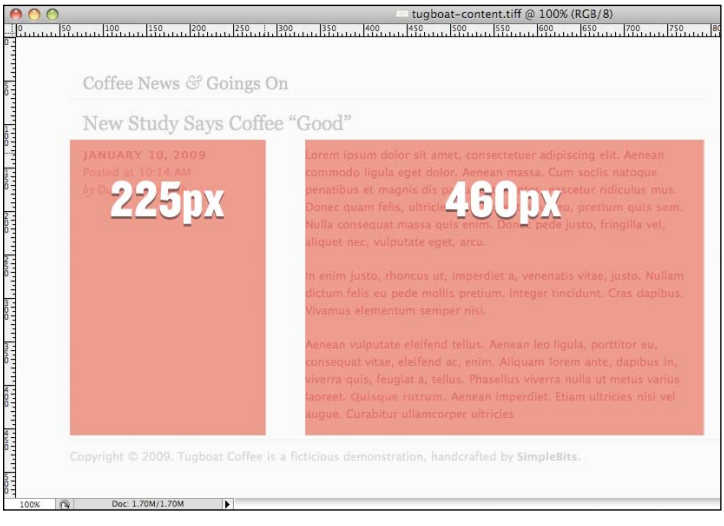



**Рис. 6.23**

В нашем образце метаданные (`<ul class="entry-meta">`) располагаются в левом столбце и занимают 225 пикселов в ширину. Под основной текст (`<div class="entry-body">`) отводятся следующие два столбца, ширина которых в сумме составляет 460 пикселов.

Опять же, если бы мы хотели ограничиться пиксельными значениями (надеюсь, вы услышали в моих словах презрительные нотки), два простых правила легко расставят все по местам.

```
div.entry ul.entry-meta {
   float: left;
   width: 225px;
}
```

```
div.entry-body {
   float: right;
   width: 460px;
   font-size: 1em;
   line-height: 1.6;
}
```

Но вы уже прочитали пару десятков страниц этой главы и должны были хорошо меня узнать. Итак, нас не интересуют, как выражаются дети, *эти вонючие пикселы*.

Однако прежде чем обратиться к нашей старой формуле с пропорцией, мы должны разобраться, что в нашем случае является контекстом. Так же как мы задавали размер амперсанда относительно содержащего его заголовка, сейчас мы уже не привязаны к блоку `#wrap` шириной `1000px`.

Вычислять значения ширины для `ul.entry-meta` и `div.entry-body` мы будем относительно их контейнера `.main` (рис. 6.24).

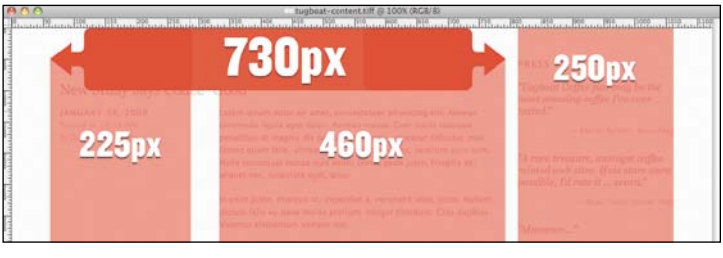



**Рис. 6.24**

Поскольку в макете размер элемента `.main` равен 730 пикселам, мы, таким образом, знаем значение контекста. А так как мы хотим выразить ширину элемента `ul.entry-meta` (`225px`) в процентах относительно `730px`, мы опять возвращаемся к старой формуле.

$$225 \div 730 = 0.30822$$

Получаем значение 0,30822 (или 30,822%). А как насчет элемента `div.entry-body`, который в макете имеет ширину `460px`?

$$460 \div 730 = 0.63014$$

Ну вот: ширина элемента `div.entry-body` составляет 63,014% от ширины его контейнера `div.main`. Без особых церемоний (и без дополнительного деления) добавим эти значения в таблицы стилей.

```
div. entry ul.entry-meta {
    float: left;
    width: 30.822%;
}

div. entry-body {
    float: right;
    width: 63.014%;
    font-size: 1em;
    line-height: 1.6;
}
```

Наконец-то наша «резиновая» сетка готова (рис. 6.25). Дайте «пять»! В итоге нам удалось расколоть этот «бородатый» орешек: жестко фиксированная ширина больше не имеет отношения к макетам страниц. Теперь *с помощью сетки* мы создаем сложные макеты с большим количеством столбцов, и такие страницы могут менять масштаб в соответствии с предпочтениями пользователя.

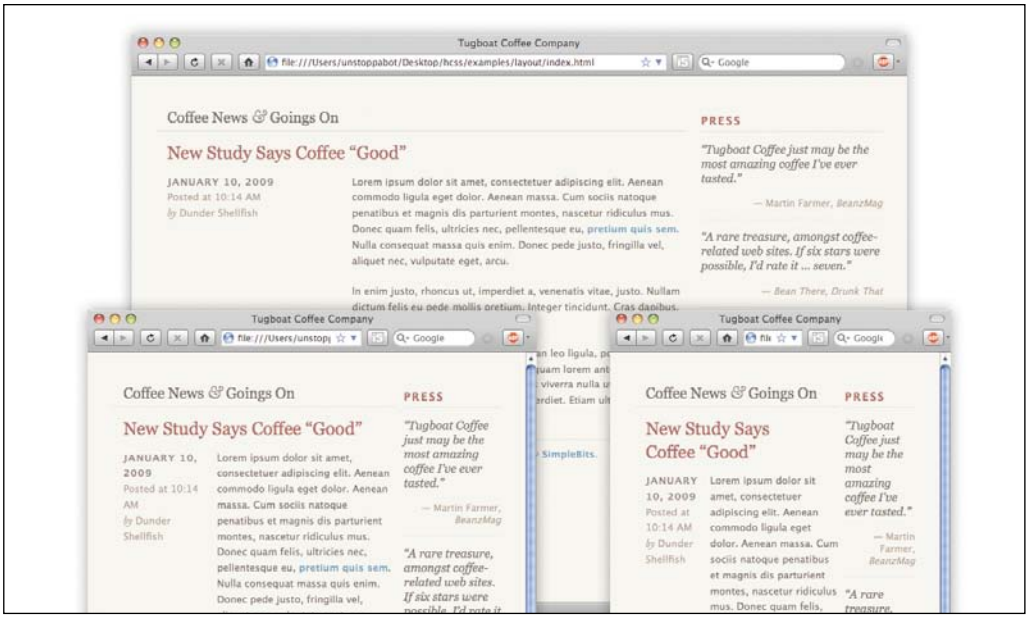



**Рис. 6.25**

Но не стоит раньше времени откупоривать шампанское. Да, действительно, мы создали масштабируемую модульную структуру. Но на самом деле макеты, над которыми мы работали, немного подыгрывают нам. До настоящего момента мы

были попросту избалованы полным отсутствием элементов с фиксированной шириной. Так и есть: нашему дизайну не хватает чего-то, что *не является* текстом.

Как вы думаете, работа у дизайнера «резиновых» интерфейсов *когда-нибудь* закончится?

## «Резиновые» элементы

К счастью, она почти закончилась. Несмотря на тот факт, что элементы фиксированной ширины действительно сильно затрудняют работу с гибким дизайном, это все же вполне решаемая задача, особенно в сравнении с тем, что мы делали до сих пор в этой книге. Все, что нужно — это аккуратные вычисления, немного процентных значений ширины и (как вы уже догадались) наша удобная формула с пропорцией.

### Это вам не просто тег IMG

Предположим, мы хотим вставить в запись блога фотографию. Этот изумительный, лицензированный на условиях Creative Commons кадр с изображением аппетитного капуччино (рис. 6.26) я нашел на сайте Flickr.

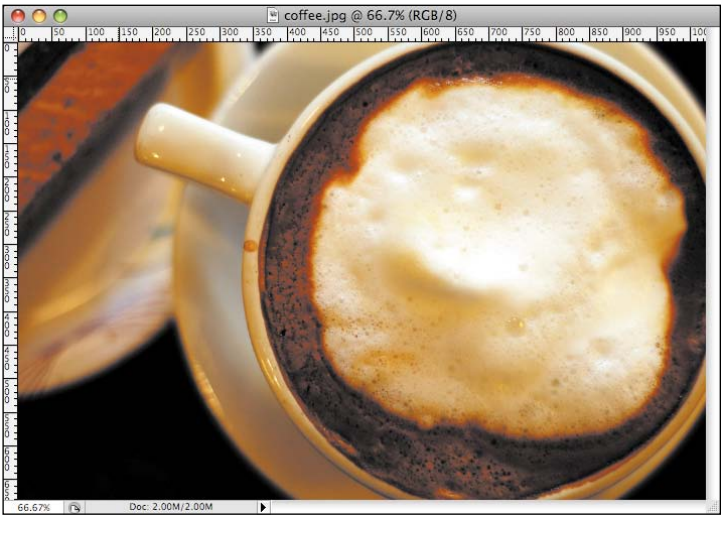



**Рис. 6.26**

Ну что ж, имея наготове великолепную фотографию, давайте освободим для нее место в разметке.

```
<div class="entry">
   <h3>New Study Says Coffee "Good"</h3>

   <ul class="entry-meta">
      <li><h4>January 10, 2009</h4></li>
      <li>Posted at 10:14 <abbr>AM</abbr></li>
      <li><span class="by">by</span> Dunder
        Shellfish</li>
   </ul>

   <div class="entry-body">
      <p class="photo"><img src="coffee.jpg"
        alt="" /></p>
      <p>Lorem ipsum dolor sit amet, consectetuer…
        </p>
   </div><!-- /entry-body -->
</div><!-- /entry -->
```

Ничего особенного: мы поместили элемент `img` для изображения с говорящим названием `coffee.jpg` внутрь параграфа вверху основного текста блога и дали классу имя `photo`.

Далее, как вы могли заметить, фотография Александра, которую я загрузил, занимает 1024 пиксела в ширину и 681 пиксел в высоту. По меркам веб-дизайна это слегка многовато. Но перед тем как мы откроем редактор изображений, чтобы уменьшить размер фотографии, давайте посмотрим на то, что у нас получилось на настоящий момент (рис. 6.27).

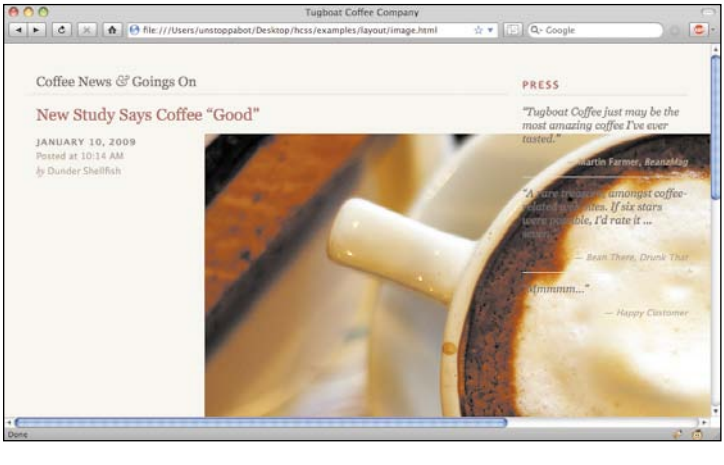



**Рис. 6.27**

Да… *это* просто ужас. Поскольку сейчас ничто не ограничивает его необъятные размеры, изображение выходит за границы содержащего его параграфа. В таком виде это никуда не годится.

Но что если как-то *ограничить* размер `coffee.jpg`? В самом деле, что, если написать CSS-правило, запрещающее изображениям превышать контейнер по ширине?

Добавьте это правило в таблицы стилей, и все проблемы будут окончательно решены.

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

Используя правило `max-width: 100%;` для всех изображений в нашем документе, мы накладываем на них очень абстрактное ограничение. Они будут отображаться в исходном размере только тогда, когда их ширина не превышает ширину контейнера. Благодаря этому единственному правилу изображения будут менять масштаб вместе с контейнерами, и наша верстка снова в порядке (рис. 6.28).

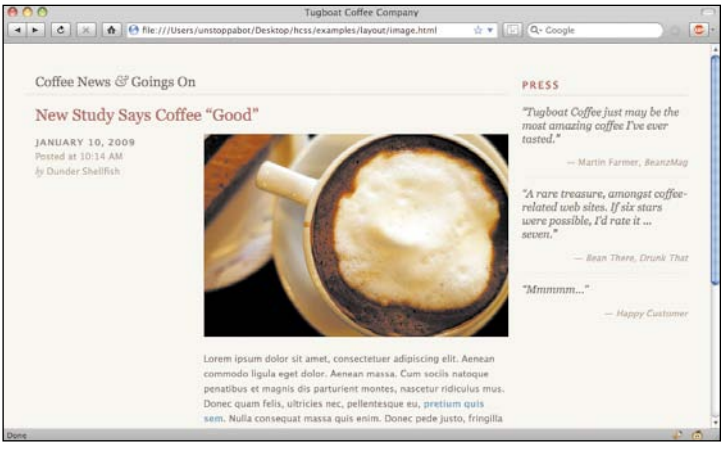



**Рис. 6.28**

Можно даже изменить стиль, который когда-то написал Дэн, и добавить невероятно модный эффект закругленной границы из главы 2.

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

```
p.photo {
   border: 15px solid #e2e1d4;
   background: #e2e1d4;
   border: 15px solid #e2e1d4;
   border-radius: 8px;
   -webkit-border-radius: 8px;
   -moz-border-radius: 8px;
}
```

Горячий кофе, «горячие» эффекты для границ — наконец-то они встретились! И в дизайне Дэна все это может чудесным образом менять свой масштаб (рис. 6.29).

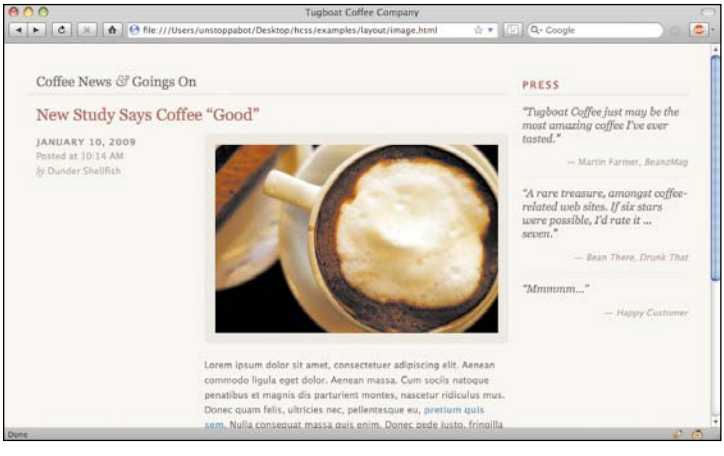



**Рис. 6.29**

Еще одно преимущество метода, позволяющего накладывать ограничения на изображения с помощью свойства `max-width: 100%`, состоит в том, что он подходит и для других элементов фиксированной ширины. К примеру, если вы хотите добавить в разметку объект Flash, можно чуть-чуть расширить правило.

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

Ну вот и все. Современные браузеры находятся на такой ступени развития, что они могут разумно менять масштаб элементов фиксированной ширины, сохраняя все пропорции. А наши таблицы стилей немного им в этом помогают, используя свойство `max-width`.

**СОВЕТ**

Используете sIFR для создания уникального шрифтового оформления на вашем сайте? Вы обрадуетесь, когда узнаете, что последние сборки sIFR 3 идеально поддерживают «резиновые» и гибкие интерфейсы, перераспределяя текст автоматически при изменении размера контейнера заголовка. Посмотреть на эту технологию в действии можно в моем блоге (*http://unstoppablerobotninja.com/*), а получить последнюю сборку sIFR 3 можно на *http://wiki.novemberborn.net/sifr3/*.

Точнее, *большинство* браузеров это делают. С остальными придется работать жестко, но ради их же блага.

## IE И ЕГО ДАЛЕКО НЕ СОВЕРШЕННАЯ РЕАЛИЗАЦИЯ CSS — СИДЕЛИ НА ТРУБЕ...

Как вы, возможно, уже знаете, версии Internet Explorer до версии 7 не поддерживают свойство `max-width`. И как бы нам ни хотелось вообще забыть о существовании IE6, на сегодняшний день этот браузер все еще занимает существенную долю рынка. Но означает ли отсутствие поддержки `max-width`, что мы, образно говоря, оказались в лодке без весел?

Вообще-то нет. В специальную таблицу стилей для IE мы можем добавить такой код.

```
img,
object {
   width: 100%;
}
```

Это правило сильно отличается от `max-width: 100%;`, и потому требует осторожности: оно задает не верхний предел ширины изображения, а отношение ширины элемента к ширине его контейнера.

Для изображений вроде нашего `coffee.jpg`, которое наверняка всегда будет слишком большим для своего контейнера, новое правило будет работать хорошо. Но изображения меньшего размера могут оказаться растянутыми и, следовательно, искаженными.

Так что если в вашей разметке есть места, где точно будут встречаться элементы слишком большого размера, подходящие для «резиновой» верстки, вы можете дополнительно усложнить селекторы и таким образом изолировать ваше правило. Например, можно сделать так.

```
img. full,
p. photo img,
object.full {
   width: 100%;
}
```

Это дополнительное усложнение — эффективное средство для ограничения сферы действия правила `width: 100%;`. Теперь мы можем быть уверены, что это правило не приведет к неуместному увеличению размера элемента в проблемных областях макета. При соблюдении разумных мер предосторожности это оказывается превосходным способом, позволяющим обойти отсутствие поддержки `max-width` в старых версиях Internet Explorer.

Что ж, задача решена, не так ли? Ну… почти. Собственно говоря, следующая проблема относится не к IE, а к *Windows*.

(Вы заинтригованы. Я в этом уверен.)

## ПРОБЛЕМА ПЛАТФОРМ (ТОЧНЕЕ, ОДНОЙ ПЛАТФОРМЫ)

Если вы посмотрите на наш дизайн в Firefox 2 под Windows или в любой версии Internet Explorer, вы увидите, что `coffee.jpg`, размер которого был изменен с помощью CSS, кажется испорченным. Оказывается, Windows не очень-то хорошо меняет масштаб изображений. В результате они выглядят слишком искусственно, а качество изображений с мелкими деталями (особенно с текстом) может сильно пострадать.

Чтобы это продемонстрировать, я приготовил пример. Я снял экранную копию текста одного из моих старых блогов,

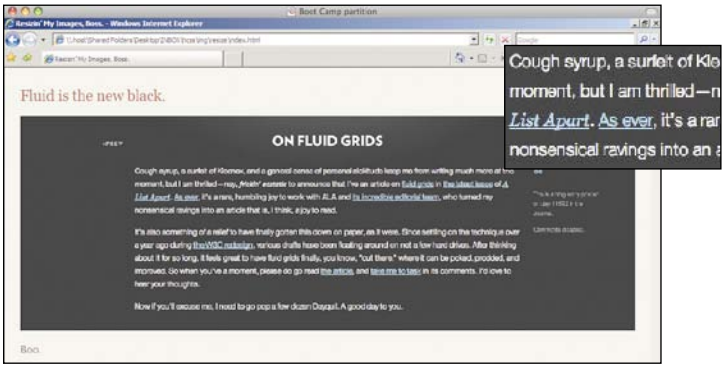



**Рис. 6.30**

и у меня получилось большое изображение. Я изменил его размер с помощью нашего правила `max-width: 100%;`, заменив его на `width: 100%;` для IE6 и более ранних версий. В итоге даже в IE7 (рис. 6.30) потеря качества просто катастрофическая.

Как видите, в IE7 текст почти полностью испорчен (рис. 6.31), а это очень серьезная проблема. Масштабируемые изображения — это замечательно и все такое, но только если в результате не ухудшается качество содержащегося в них контента.


**Рис. 6.31**

К счастью, эта проблема встречается в ограниченном числе браузеров: во всех версиях Internet Explorer, а также в Firefox 2 (и более ранних версиях) под Windows. С Safari/Win (Safari под Windows), Opera/Win и Firefox 3/Win все в порядке, они правильно меняют размер изображений. Поэтому мы еще немного потянем резину с нашим макетом (можно придумать еще много таких метафор): давайте обратимся к проблемным браузерам и попробуем понять, что можно сделать.

Сначала плохая новость.

### Firefox 2 под Windows, мы совсем тебя не знали

К сожалению, проблема с Firefox 2 — одна из тех, с которыми нам придется смириться. Использовать JavaScript для выделения кода, предназначенного для старых версий Firefox, да еще и только под Windows — как минимум, ненадежный способ. Но более важно то, что в Firefox 2 нет скрытых механизмов, позволяющих решить эту проблему с изображениями. Если бы существовал, скажем, какой-нибудь переключатель, который бы заставлял браузер отображать элементы в высоком разрешении, мы бы могли все исправить. Но так как его нет, мы бессильны.

Но здесь есть и положительный момент: пользователи Firefox очень внимательно относятся к обновлениям, и поэтому Firefox 2 сейчас используется все реже и реже. Конечно, ничто не может быть хуже, чем отсутствие работающего решения. Но мы надеемся, что нехватка «переключателя, повышающего качество отображения элементов» затронет не так уж много наших пользователей.

Постойте! Оказывается, в Internet Explorer *есть* такой переключатель. Кто бы мог подумать!

### Специфичные для IE CSS-фильтры, на помощь! (Стоп, разве я это только что написал?)

Давняя проблема браузера Internet Explorer 6 и более ранних версий — его неспособность работать с PNG-изображениями, а точнее, с присущей им альфа-прозрачностью. Чтобы ее обойти, дизайнеры используют специфичный для Microsoft IE CSS-фильтр `AlphaImageLoader` (*http://msdn.microsoft.com/en-us/library/ms532969.aspx*). Если, к примеру, в качестве фона для элемента `#logo` вы вставляете PNG-изображение, добавление следующего кода в специальную таблицу стилей для IE решит проблему с прозрачностью.

```
#logo a {
   background: none;
   filter: progid:
     DXImageTransform.Microsoft.AlphaImageLoader
     (src="/path/to/bg.png",
     sizingMethod="scale");
}
```

**ПРИМЕЧАНИЕ**

Кроме того, существует множество функций JavaScript, предназначенных для решения проблем с прозрачностью PNG-изображений в элементах `img`. Лично мне очень нравится SuperSleight Дрю Макклеллана (*http://24ways.org/2007/supersleight-transparent-png-in-ie6*). Правда в последнее время я пользуюсь библиотекой DD_belatedPNG (*http://www.dillerdesign.com/experiment/DD_belatedPNG/*), которая позволяет создавать более гибкие решения, чем трюки с `AlphaImageLoader`.

Кто знает, что *на самом деле* происходит под капотом IE! Фактически там происходят три вещи.

1. PNG-изображение, которое использовалось в качестве фона для элемента `#logo`, удалено.

2. Это изображение добавлено в объект `AlphaImageLoader`, который располагается «между» фоновым слоем и контентом элемента `div`.

3. Свойство `sizingMethod` (*http://msdn.microsoft.com/en-us/library/ms532920%28VS. 85%29.aspx*) определяет, что делать с изображением внутри объекта `AlphaImageLoader`: обрезать по размеру контейнера (`crop`), считать нормальным изображением (`image`) или изменить масштаб в соответствии с размером контейнера (`scale`).

Какое это имеет отношение к нашей проблеме с отображением элементов? Когда я работал над новым дизайном для W3C, я обнаружил, что применение к изображению фильтра `AlphaImageLoader` существенно повышает качество его отображения в IE — или, по крайней мере, доводит его до уровня других, менее неудачных браузеров. И чтобы автоматизировать процесс, я написал код на JavaScript. Вот он.

```
var imgSizer = {
   Config : {
      spacer : "/path/to/your/spacer.gif",
      imgCache: []
   }

   ,collate : function(oScope) {
      if (document. all &&! window. opera) {
         var c = imgSizer;
         var imgCache = c.Config.imgCache;

         var images = (oScope && oScope.length)?
           oScope: document.getElementsByTagName
           ("img");
         for (var i = 0; i < images.length; i++)
     {
            images [i].origWidth = images [i].
              offsetWidth;
            images [i].origHeight = images [i].
              offsetHeight;

            imgCache.push(images [i]);
            c.ieAlpha(images [i]);
            images[i].style.width = "100%";
         }
```

```
            if (imgCache.length) {
               c.resize(function() {
                  for (var i = 0; i < imgCache.
                    length; i++) {
                     var ratio = (imgCache [i].
                       offsetWidth / imgCache[i].
                       origWidth);
                     imgCache[i].style.height =
                       (imgCache[i].origHeight *
                       ratio) + "px";
                  }
               });
            }
         }
      }
      ,ieAlpha: function(img) {
         var c = imgSizer;
         if (img. oldSrc) {
            img.src = img.oldSrc;
         }
         var src = img.src;
         img.style.width = img.offsetWidth + "px";
         img.style.height = img.offsetHeight + "px";
         img.style.filter = "progid: DXImageTransform.
           Microsoft.AlphaImageLoader (src='" + src
           + "', sizingMethod='scale')"
         img.oldSrc = src;
         img.src = c.Config.spacer;
      }
      ,resize: function(func) {
         var oldonresize = window. onresize;
         if (typeof window. onresize! = 'function')
           {
            window. onresize = func;
         } else {
            window. onresize = function() {
               if (oldonresize) {
                  oldonresize();
               }
               func ();
            }
         }
      }
```

```
}
```

Единственное, что вам нужно сделать, — это изменить путь `spacer` (третья строка сверху) так, чтобы он указывал на прозрачное GIF-изображение размером 1×1. Запустите этот код после загрузки страницы, и он оживит все ваши изображения. Лично я обычно использую `addLoadEvent()` Саймона Уиллисона (*http://simonwillison.net/2004/May/26/addLoadEvent/*), чтобы выстроить в очередь все функции, которые должны запускаться после загрузки страницы. Выглядит это примерно так.

```
<script type="text/javascript">
addLoadEvent(function() {
   imgSizer.collate();
   });
</script>
```

Как и в случае с правилом `width: 100%;` для IE, функция `imgSizer.collate()` очистит *все* изображения в документе, применив трюк с `AlphaImageLoader`. Если для вас это слишком общее правило, функция будет работать и для коллекции элементов `img`.

```
<script type="text/javascript">
addLoadEvent(function () {
   var oImgs = document.getElementById("wrap").
     getElementsByTagName ("img");
   imgSizer.collate(oImgs);
   });
</script>
```

**ПРИМЕЧАНИЕ**

Вам не хочется перепечатывать этот код вручную? Вообще-то, я вас не виню. Хорошая новость для фанатов «копипасты»: я недавно описал этот метод в своем блоге (*http://unstoppable-robotninja.com/entry/fluid-images/*), и там можно скачать этот скрипт. Наслаждайтесь!

Не важно, как вы будете использовать это решение. Преимущества трюка с `AlphaImageLoader` вполне очевидны.

Наконец-то с нашим примером все в порядке (рис. 6.32). Посмотрите, как выглядит текст до и после применения этого метода (рис. 6.33). Уверен, вы согласитесь со мной: хотя нам пришлось написать довольно неприятный кусок специфического кода, качество изображения после изменения его размера оправдывает наши усилия. Более того, последний барьер на пути к гибким, «резиновым» сайтам остался позади.

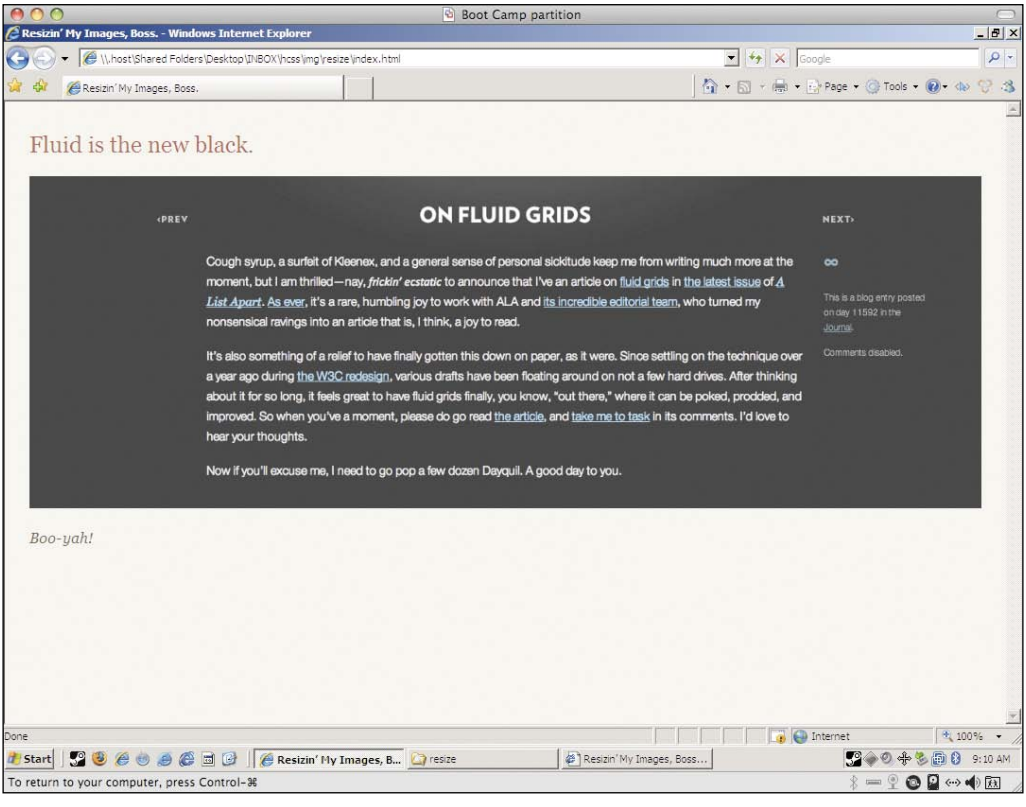



**Рис. 6.32**

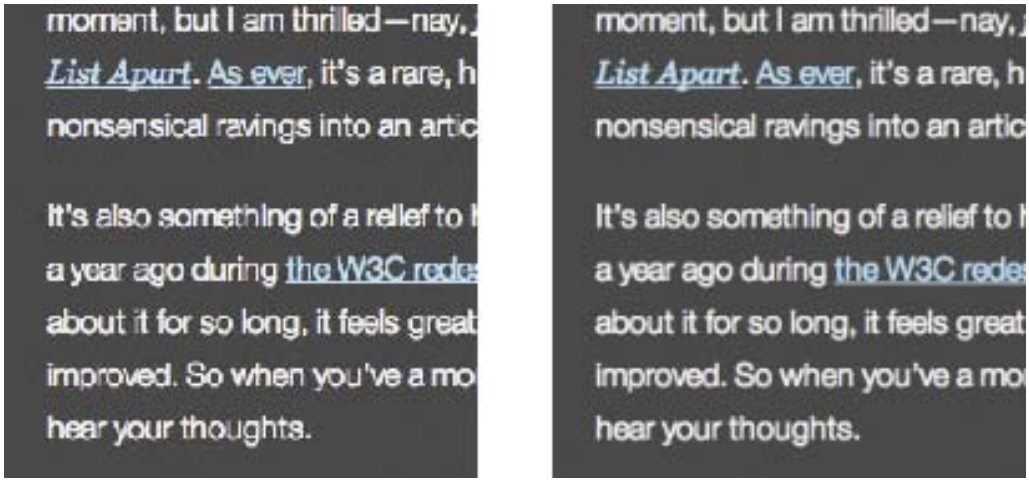



**Рис. 6.33**

# В заключение

Вот это да! Мы совершили путешествие длиной в целую главу, не так ли? Мы просмотрели несметное число инструментов из арсенала веб-дизайнера с гибким мышлением, и нам удалось создать модульную и при этом «*резиновую*» сетку. Но что, если это только теория? Кто-нибудь на самом деле использует все это?

Как оказалось, оба ваших покорных автора (рис. 6.34). И во время работы над новым дизайном сайта W3C наша фирма действительно создала набор готовых к использованию шаблонов, и все они основывались на пуленепробиваемой «резиновой» сетке. Я твердо убежден, что время «резиновых» макетов уже пришло, и я надеюсь, наши попытки разбудить ваше нефиксированное воображение увенчались успехом.

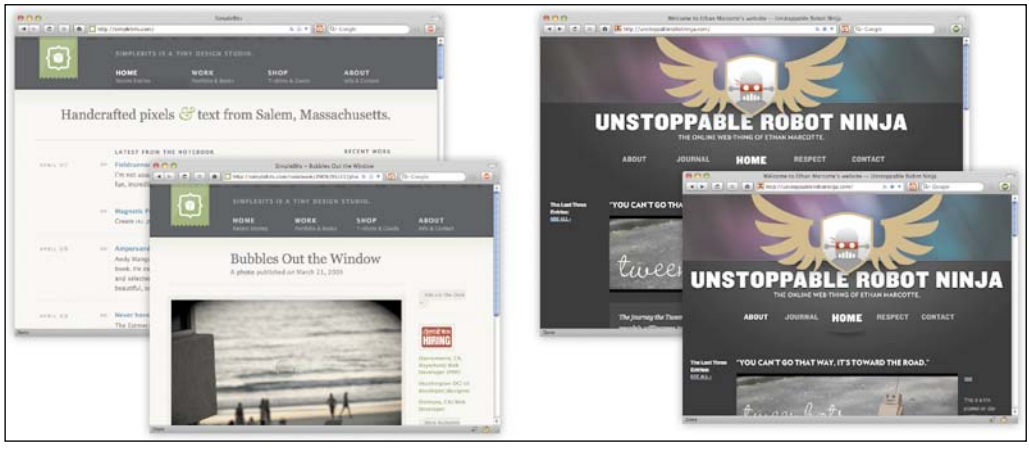



**Рис. 6.34**

Но следует ли из сказанного в этой главе делать вывод, что дизайн фиксированной ширины — это плохо? Отнюдь нет. Но я думаю, пришло время понять, что привычки наших пользователей не так неизменны, как макеты, сделанные в Photoshop. Как мы увидели в этой главе, инструменты для изменения масштаба дизайна вполне доступны — нужно только перестать опираться на «минимальное разрешение экрана».

Когда мы это сделаем, пользователи скажут нам спасибо.

# Глава 7

## Тонкости мастерства

**Смелое архитектурное решение способно превратить обычное здание в достопримечательность, но только с помощью деталей архитектор способен выразить свой замысел.**

Кертис Фентресс, архитектор

В эпиграфе Кертис Фентресс говорит об архитектуре в физическом смысле, однако это рассуждение можно применить и к веб-дизайну. Вы наверняка слышали выражение «Дьявол кроется в деталях». Но в центре внимания Фентресса оказывается нечто другое: детали важны (и нужны) для того, чтобы архитектурное произведение могло *выражать мысль*. И я с ним совершенно согласен. В области веб-дизайна при реализации проекта всплывает целая какофония мелочей, и именно умение экономить драгоценное время, тратя его только на *нужные* мелочи, превращает ваш сайт в дизайнерский шедевр.

Эту последнюю главу я собираюсь посвятить обсуждению еще нескольких мелочей, касающихся нашего старого доброго шаблона Tugboat. Многие из них забавны, но все они окажутся полезными. Примеры из этой главы — это уже тонкости мастерства: они часто очевидны не для всех, но если они присутствуют, мощнее становится и дизайн, и сайт, и смысл. И именно такие мелочи вызывают у меня самый большой восторг.

Давайте начнем с небольшого хитрого приема, имеющего отношение к типографике, а потом посмотрим, как он приведет нас в огромный мир шрифтов Сети.

## Применяйте самый лучший амперсанд из доступных

Типографика играет *ключевую* роль в веб-дизайне. Лично я вообще считаю, что именно Сеть явилась причиной постоянного интереса к этой области дизайна. Подумайте: что это за сайт, если в нем нет гипертекста? Это доступный и распространенный ресурс, и это средство общения людей (и сайтов) в Сети. Так что внимание к искусству типографики может играть ведущую роль в создании содержательных интерфейсов.

Джеффри Зельдман в своей статье «Understanding Web Design» (*http://www.alistapart.com/articles/understandingwebdesign*) пишет следующее.

> *Если и сравнивать Сеть с другими областями, лучше всего подойдет типографика. Потому что веб-дизайн, как и шрифт, дает другим людям возможность высказаться.*

Здесь Джеффри сравнивает веб-дизайн в целом с типографским дизайном. Каждый сайт имеет цель, передает сообщение или же выполняет действие, но визуальное оформление и подача могут отличаться — так же, как одни и те же буквы в разных шрифтах одного типа могут передавать целый спектр эмоций и чувств.

Работая веб-дизайнером, я помешался на шрифтах. Хорошее шрифтовое оформление стало приоритетом во всем, что я делаю.

Я говорю не об использовании каких-то особенных шрифтов, а о *настройке* гипертекста. Оливер Райхенштайн, к примеру, в своей статье «Веб-дизайн на 95% состоит из типографики» («Web Design is 95% Typography», *http://informationarchitects.jp/the-web-is-all-about-typography-period*) утверждает следующее.

> *В эпоху итальянского Возрождения в распоряжении печатника был всего один шрифт, и, тем не менее, работы этого периода — в числе самых прекрасных произведений типографского искусства.*

В отношении нас, веб-дизайнеров, эта идея особенно верна. Ведь нас очень беспокоит то, что мы зависим от ограниченного набора шрифтов, установленных пользователем.

Далее Оливер говорит во что.

> *Информационный дизайн — это не хорошие шрифты, а хорошая типографика… Любой может использовать шрифты, кое-кто может выбрать хорошие шрифты, но лишь немногие владеют искусством типографики.*

Хотя на данный момент мы ограничены в выборе шрифтов, доступных для использования в Сети, мы можем стремиться улучшить типографику, работая с ограниченным набором. А под типографикой я понимаю современные возможности CSS, позволяющие обрабатывать текст, меняя высоту строки, регистр, интервал между знаками, размер, цвет и т. д. Тот факт, что сейчас должное внимание уделяется типографским принципам, сформировавшимся за сотни лет до появления тега `<font>`, уже сам по себе что-то значит.

Стараясь помнить об этом, поговорим о маленьком типографском приеме, который используется в шаблоне Tugboat, и о том, как его реализовать.

## КАК НАЙТИ СВОЕГО ВНУТРЕННЕГО БРИНГХЕРСТА

Я очень рекомендую вам авторитетное издание о науке и искусстве типографики — «Основы стиля в типографике» Роберта Брингхерста (рис. 7.1), хотя это не самое легкое чтиво.

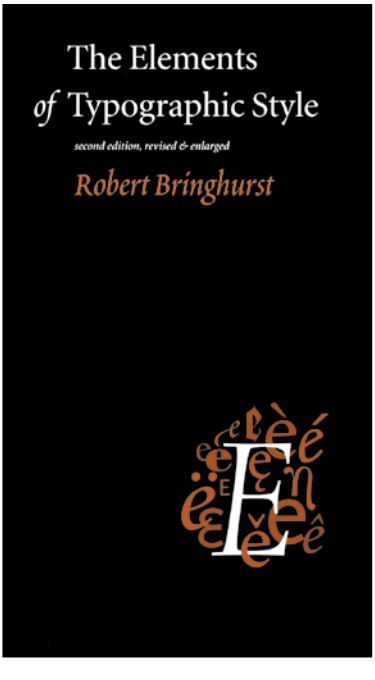



**Рис. 7.1.** Познакомьтесь с «Основами стиля в типографике»

В этой книге вы найдете множество правил и рекомендаций по созданию превосходного шрифтового оформления — причем, напоминаю, здесь речь идет не только о веб-дизайне,

но и о типографике в целом вместе с ее давним легендарным прошлым. Большая часть сведений из этой книги, безусловно, применима *также* и к гипертексту, и в этом вся прелесть типографики. Главное — это не красивые шрифты, а то, как мы их используем.

Ричард Раттер (*http: //webtypography.net/intro/*) взялся за детальный разбор рекомендаций Брингхерста, показывая, как каждое из правил может быть реализовано с помощью пуленепробиваемых CSS (рис. 7.2). Он взвалил на себя огромную задачу, но, будем надеяться, процент несделанного будет со временем уменьшаться.

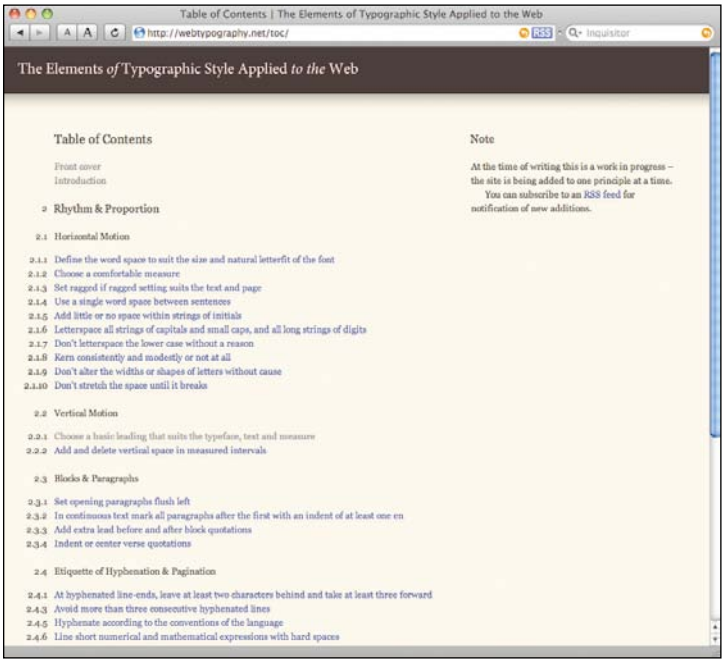



**Рис. 7.2.** Великое дело Ричарда Раттера

## ПРАВИЛО 5.1.3

Правило 5.1.3 из книги Брингхерста «Основы стиля в типографике» гласит:

> *В заголовках и титулах применяйте самый лучший амперсанд из доступных.*

В пояснении Брингхерст говорит о том, что курсивный амперсанд зачастую выглядит красивее и интереснее, чем прямой. Продолжая эту мысль, он пишет следующее.

*Так как амперсанд чаще используется в акцидентном наборе, чем в обычном тексте, более творческие варианты, как правило, оказываются более полезными. Например, можно позаимствовать курсивный амперсанд для текста, набранного прямым начертанием.*

У меня слабость к амперсандам — вот почему это правило обратило на себя мое внимание, и я сразу же решил применить эту идею в Сети к моим заголовкам и титулам, содержащим амперсанд. Но сначала я хочу кое-что рассказать.

## МЫ С САМОГО НАЧАЛА ПРИДЕРЖИВАЛИСЬ ИДЕИ ПРОГРЕССИВНОГО ОФОРМЛЕНИЯ

Помню, как я создавал свой первый сайт. Я тогда жил в Олстоне, Массачусетс, в квартире, которую стоило признать непригодной для проживания. Днем я работал на складе местной студии звукозаписи за $5 в час, а по ночам просматривал исходный код страниц и сражался с HTML.

Я гордился дизайном своего первого сайта. К сожалению, я потерял сам макет и исходный код, но помню, что это была дань уважения игровой приставке Atari 2600 (а почему бы и нет?).

Тогда я был без ума от футуристического и *бесплатного* шрифта Neuropol (рис. 7.3). Непонятно, откуда у меня взялась такая симпатия к этому конкретному шрифту, но он определенно был из разряда тех, которые стал бы устанавливать далеко не каждый желающий посмотреть мой шедевр. Но по-настоящему я этого тогда не понимал. Я хотел, чтобы на моем сайте использовался необычный шрифт, и я выбрал Neuropol для заголовка, полагая, что именно его-то все и увидят.


**Рис. 7.3.** Я выбрал этот жутко безвкусный шрифт для своего первого сайта

К счастью, HTML-редактор, который я тогда использовал, любезно создавал для меня список шрифтов (а поскольку это был конец 90-х, он делал это с помощью тега `<font>`, который сейчас является недопустимым).

```
<font face="Neuropol, Verdana, Arial, sans-serif"
      >
```

HTML-редактор предусмотрительно добавлял запасные варианты на случай, если шрифт Neuropol не установлен пользователем. Эта идея наверняка знакома вам как опытному разработчику, но в то время я этого не знал, но при этом создавал прогрессивное оформление для своего первого сайта, используя необычный шрифт для тех пользователей, у которых он установлен, и запасной вариант для тех, у кого его нет.

## МЫ, КАЖЕТСЯ, ГОВОРИЛИ ОБ АМПЕРСАНДАХ?

Чтобы связать все это с правилами Брингхерста, мы можем применить простую идею прогрессивного оформления к заданию доступных шрифтов для амперсандов, встречающихся в титулах и заголовках. И именно это мы уже сделали для заголовка «Coffee News & Goings On» (об этом рассказывал Итан в предыдущей главе).

Вы, наверное, помните объяснение Итана: хотя для заголовка используется шрифт Georgia, для амперсанда выбран изощренный курсивный вариант шрифта Baskville (рис. 7.4). Внимая призыву Брингхерста, создадим с помощью CSS список шрифтов для «самого лучшего амперсанда из доступных», содержащий запасные варианты для операционных систем, в которых не установлен Baskville.

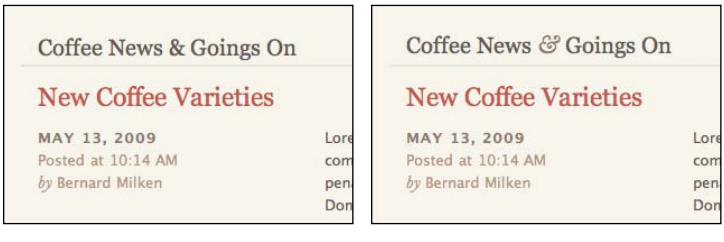



**Рис. 7.4.** Обычный амперсанд из шрифтом Georgia (слева) и курсивный амперсанд из шрифта Baskville (справа)

Мы заключили амперсанд в элемент `<span>`, чтобы создать специальный класс для настройки оформления амперсандов в заголовках всего сайта.

```
<h2>Coffee News <span class="amp">&amp;</span>
    Goings On</h2>
```

И снова перед нами CSS-объявление для класса `.amp`, где в одной из строк перечисляются шрифты, каждый из которых создает уникальный курсивный вариант амперсанда.

```
span. amp {
   font-family: Baskerville, "Goudy Old Style",
     "Palatino", "Book Antiqua", Georgia, serif;
   font-style: italic;
   }
```

Идея в том, что у большинства пользователей системы Mac установлен Baskville, некоторым так повезло, что у них есть Goudy Old Style, а еще больше любителей Windows имеют в своем распоряжении Palatino или Book Antiqua. Я расположил их в порядке их «интересности» (рис. 7.5), рассчитывая, что почти все гости сайта увидят, по крайней мере, более интересный амперсанд, чем обычный вариант, написанный шрифтом Georgia.

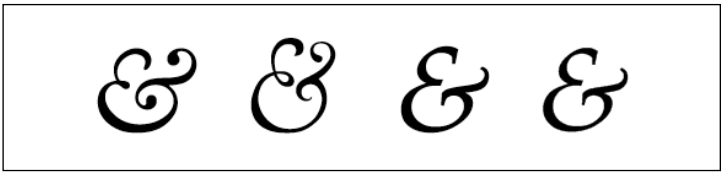


**Рис. 7.5.** Слева направо: курсивные амперсанды, написанные шрифтами Baskville, Goudy Old Style, Palatino и Book Antiqua (последний, как вы могли заметить, основан на Palatino)

Конечно же, вы можете составлять свои списки шрифтов, исходя из вариантов, доступных для Mac и Windows — и здесь есть из чего выбрать.

- На рисунке 7.6 показаны шрифты, установленные по умолчанию в системе Mac; среди них вы найдете несколько очень интересных курсивных амперсандов.
- На рисунке 7.7 изображены интересные амперсанды, которые по умолчанию есть в Windows XP.
- На рисунке 7.8 вы видите несколько новых шрифтов, появившихся в Windows Vista, и их интересные амперсанды.

Просматривая эти таблицы, вы можете начать создавать ваш собственный «список амперсандов Брингхерста», не забывая добавить в него шрифт, который будет доступен в любой операционной системе.

И я снова говорю: нет ничего страшного в том, что амперсанд выглядит неодинаково в разных ситуациях. Сейчас вы уже наверняка стали приверженцем идеи прогрессивного оформления, разве нет? Ведь взамен вы получаете простой, гибкий

Рис. 7.6

Рис. 7.7

Рис. 7.8

способ улучшать шрифтовое оформление в Сети, используя всего лишь гипертекст. И в самом деле, тонкость мастерства.

## Встраивание шрифтов с помощью CSS

До этого я говорил о творческом подходе к выбору шрифтов, которые, возможно, были установлены *некоторыми пользователями*. Но как было бы здорово, если бы мы, дизайнеры, могли сами распоряжаться тем, какие шрифты мы хотим использовать, и встраивать их в макет!

Успешными были уже первые попытки sIFR (*http://wiki.novemberborn.net/sifr3/*) — технологии, использующей CSS/Flash/JavaScript и позволяющей встраивать шрифты, заменяя гипертекст на странице. А совсем недавно появилась технология Cufón (*http://wiki.github.com/sorccu/cufon/about*), которая переводит шрифты в специальный формат, а отрисовка производится с помощью JavaScript.

Однако вполне возможно, что будущее типографики в Сети находится в руках `@font-face`.

Впервые появившаяся в CSS2, исчезнувшая в CSS2.1 и снова добавленная в CSS3, директива `@font-face` позволяет встраивать файлы шрифтов с помощью CSS-объявлений.

Проще всего объяснить это на примере. И, кстати, эту идею легко реализовать.

## ДОБАВЛЕНИЕ @FONT-FACE В ШАБЛОН TUGBOAT

К примеру, представим, что в шаблоне Tugboat вместо Lucida Grande и Georgia мы хотим использовать Archer — приятный геометрический шрифт с засечками, созданный Хофлером и Фрер-Джонсом (*http://typography.com/fonts/font_overview.php?productLineID=100033*).

Сначала напишем объявление, в котором зададим имя шрифта и исходный файл с помощью директивы `@font-face`.

```
@font-face {
   font-family: "Archer Medium";
   src: url(fonts/Archer-Medium. otf) format
     ("opentype");
   }
```

Этот код делает две вещи:

1. создает имя шрифта (любое на ваш выбор), в данном случае — «Archer Medium»;
2. сопоставляет этому имени прямую ссылку непосредственно на файл шрифта (в данном случае — файл в формате OpenType, находящийся в каталоге */css/fonts*).

Теперь, когда этот встроенный шрифт задан, использовать название семейства шрифтов Archer Medium в CSS мы можем везде, где мы бы использовали обычные названия шрифтов — например в элементе `<body>`.

```
body {
   font-family: "Archer Medium", "Lucida Grande",
     "Lucida Sans Unicode", "Lucida Sans",
     Helvetica, Arial, sans-serif;
   color: #51463d;
   font-size: 62.5%;
   background: #f3f2e8;
   }
```

Готово! На рисунке 7.9 показан шаблон Tugboat в браузере Safari: теперь везде используется шрифт Archer Medium. Никаких скриптов, полностью масштабируемый текст, который можно выделять курсором мыши. Правда, есть две маленькие проблемы: специфический формат IE и вопрос о лицензировании шрифтов.

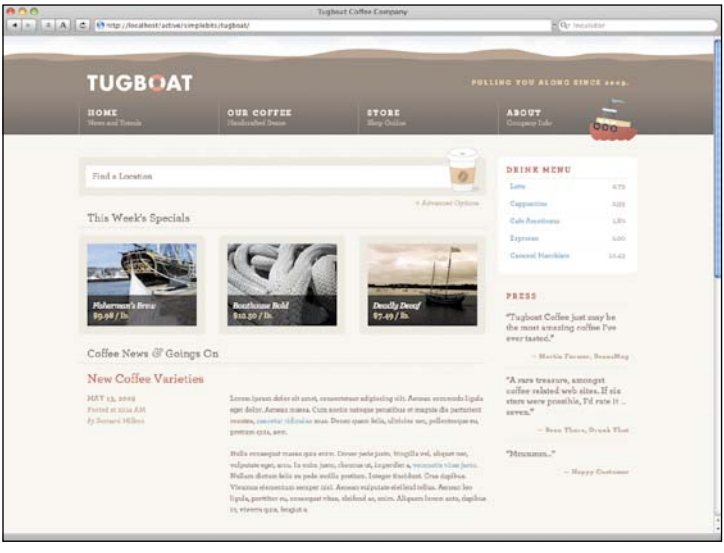



**Рис. 7.9.** Шаблон Tugboat, в котором весь текст написан шрифтом Archer Medium (в браузере Safari)

## ПОДДЕРЖКА @FONT-FACE

Как вам кажется, на сегодняшний день `@font-face` является надежным методом работы со шрифтами в Сети? Вероятнее всего, нет, но мы к этому приближаемся, ведь метод продолжает развиваться. Поддержка встраивания шрифтов OpenType и TrueType есть в Safari 3.1+, Firefox 3.5 и Opera 10.

**СОВЕТ**

Более подробно о поддержке `@font-face` можно прочитать на *http://www.webfonts.info/wiki/index.php?title=%40font-face_browser_support*.

Я сообщу вам шокирующую новость: `@font-face` поддерживается браузером Internet Explorer начиная с версии 4 — правда, IE использует специфичный для Microsoft формат шрифта Embedded Open Type (EOT), который все остальные браузеры решили не внедрять. Поэтому сейчас для того, чтобы шрифты работали в большом количестве браузеров, вам придется создавать два файла: в формате TrueType или OpenType (для Safari, Firefox и Opera) и в формате EOT (для IE). Советую вам почитать статью Йона Тэна о методе, позволяющем угодить всем с помощью условных комментариев (*http://jontangerine.com/log/2008/10/font-face-in-ie-making-web-fonts-work*).

При желании здесь тоже можно придерживаться идеи прогрессивного оформления, добавляя шрифты OpenType и TrueType для всех, кроме IE, и тем самым заставляя IE использовать шрифт, установленный по умолчанию. Но в этой книге мы достаточно жестоко относились к браузеру Internet Explorer, и теперь мы надеемся, что когда-нибудь для всех браузеров будет выбран один формат. Хотя, возможно, я опять слишком оптимистичен.

## ПРОБЛЕМА ЛИЦЕНЗИРОВАНИЯ

Пожалуй, наличие нескольких форматов представляет собой не столь серьезную проблему по сравнению с вопросом законности, возникающим при использовании прямых ссылок на файлы шрифтов. Подобно тому как музыкальная индустрия стремится приспособиться к эпохе цифровой информации, типографы и производители шрифтов опасаются того, что их файлы шрифтов могут оказаться в свободном доступе в Сети, что в свою очередь может привести к пиратству. В настоящее время лишь несколько производителей предлагают лицензию на использование своих шрифтов в `@font-face`, а некоторые даже запрещают это делать.

К примеру, замечательный шрифт Archer, который мы до этого использовали, не подходит для Сети, поскольку разработчики (Хофлер и Фрер-Джонс) в лицензионном соглашении четко указали, что его нельзя использовать в `@font-face`.

Поэтому сейчас нам нужен законный способ встраивать шрифты так, чтобы это было выгодно производителям, но контроль осуществлялся дизайнером CSS. Я убежден в том, что в скором времени появится нечто подобное, поскольку встраивание шрифтов является одним из прогрессивных направлений в веб-дизайне.

## БЕСПЛАТНЫЕ (ПОКА) ШРИФТЫ

Однако оставим вопрос лицензирования. Ведь на самом деле это не мешает нам уже сейчас начать экспериментировать с `@font-face`. Просто нужно быть осторожным в выборе встраиваемых шрифтов. К счастью, существует немало очень удобных *бесплатных* шрифтов, которые вы *прямо сейчас* можете использовать в браузерах, поддерживающих веб-шрифты.

Возьмем, к примеру, бесплатную коллекцию шрифтов Йоса Бьювенги exljbris (рис. 7.10) (хотел бы я знать, как это произносится).

Йос создает бесплатные шрифты высокого качества, причем в лицензионном соглашении указано, что их можно встраивать со ссылкой на источник.


**Рис. 7.10.** Бесплатная коллекция шрифтов Йоса Бьювенги (*http:// www. josbuivenga. demon. nl/index. html*)

На рисунке 7.11 показан шаблон Tugboat, в котором с помощью `@font-face` задан шрифт Museo Medium — один из удивительных бесплатных шрифтов Бьювенги. Это отличный пример того, что мы можем делать сегодня. Прямо сейчас. И это только малая часть всего, что дает нам (и со временем еще даст) добавление новых средств шрифтового оформления в инструментарий веб-дизайнера.

Так что история встраиваемых шрифтов еще далеко не окончена, но здорово, что она начинает набирать обороты. Возможно, после того как в Firefox 3.5 была добавлена поддержка правил `@font-face`, мы приближаемся к переломному моменту в вопросе реального использования этого метода на практике. Если удастся уладить правовые споры между создателями шрифтов

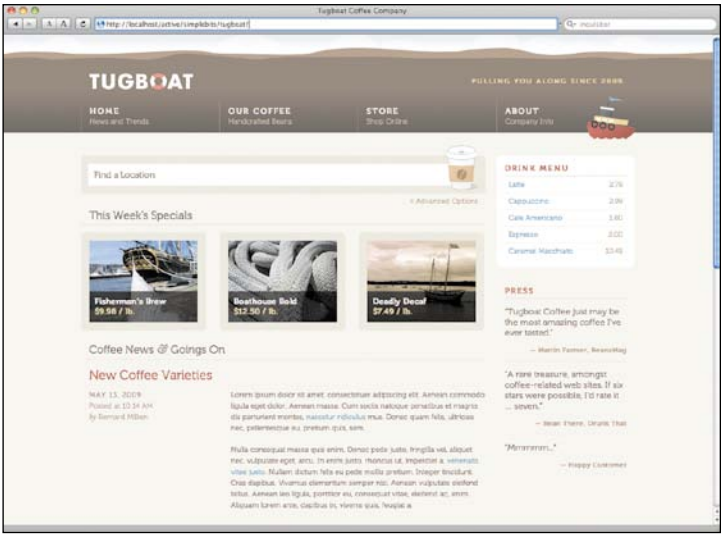



**Рис. 7.11.** Шаблон Tugboat, в котором с помощью @font-face задан шрифт Museo Medium

и дизайнерами сайтов и если IE добавит поддержку форматов TrueType и OpenType, шрифтовое оформление в Сети, безусловно, изменится до неузнаваемости.

Как и в случае с другими замечательными возможностями CSS3, о которых мы говорили в этой книге, время начинать экспериментировать *уже* пришло. Когда в будущем эти технологии получат широкое распространение, вы будете с ними уже хорошо знакомы; в определенных случаях эти прогрессивные инструменты можно использовать прямо сейчас.

## TYPEKIT

Компания Small Batch объявила о работе над весьма интересным проектом, который может сделать веб-шрифты реальностью. Вот что можно прочитать о Typekit на сайте его создателей (*http://typekit.com/*).

> *Мы вместе с производителями шрифтов разрабатываем унифицированную лицензию на использование шрифтов в Сети. Мы создали платформу, позволяющую нам объединить вместе все бесплатные и коммерческие шрифты; при этом скорость работы неимоверно высока, различия в распознавании шрифтов браузерами сведены к минимуму, а разработчики шрифтов получают необходимую защиту, не прибегая к раздражающим и неэффективным DRM.*

*Став пользователем Typekit, вы получите доступ к нашей библиотеке высококачественных шрифтов. Просто добавьте строку кода на JavaScript в разметку, выберите нужные вам шрифты и продолжайте работать над созданием веб-страницы так, как вы привыкли. Только теперь вы сможете использовать реальные шрифты. И эта «реальность» способна изменить веб-дизайн.*

Снимем шляпы перед Small Batch! Они попытались справиться с проблемой законности использования шрифтов, а не просто жаловались на отсутствие разумного решения. Мы будем внимательно следить за развитием этого проекта как законного способа встраивания веб-шрифтов. При этом особенно замечательно то, что появление Typekit будет стимулировать дальнейшее формирование таких технологий. А это прогресс.

## jQuery

Давайте отвлечемся от типографики и немного поговорим о JavaScript. Раньше я никогда не писал о JavaScript, и поэтому считаю себя дилетантом. Но дизайнеру, который живет и дышит CSS, не могла не понравиться библиотека jQuery (рис. 7.12), ведь она позволяет легко и быстро управлять поведением моих интерфейсов привычным способом.

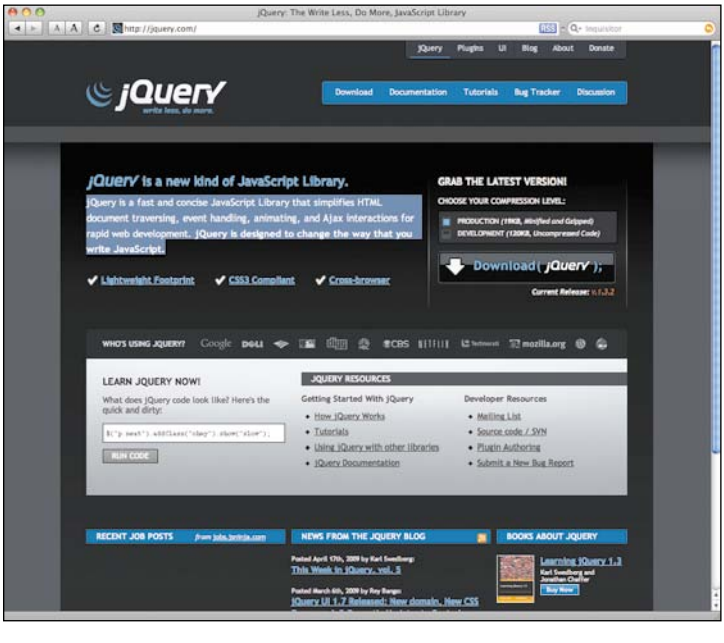



**Рис. 7.12.** jQuery, библиотека JavaScript, на http: //jquery. com/

На одноименном сайте говорится, что jQuery — это

> *…быстрая и компактная библиотека JavaScript, которая упрощает обход HTML-документа, обработку событий, добавление анимации и AJAX-взаимодействий, ускоряющих веб-разработку. jQuery создана для того, чтобы изменить привычный способ написания кода на JavaScript.*

Иными словами, с помощью библиотеки JavaScript вам будет проще реализовывать стандартные дизайнерские идеи при создании интерфейсов. Существует много библиотек, но мне сразу же понравился напоминающий CSS синтаксис селекторов jQuery, использующийся для добавления событий и эффектов к отдельным элементам. Просто понравился.

Основная часть jQuery содержится в одном файле с расширением *.js* — но существует большое сообщество, члены которого делятся своими плагинами для всего, что только можно написать на JavaScript.

## ИСПОЛЬЗОВАНИЕ JQUERY В ШАБЛОНЕ TUGBOAT

В шаблоне Tugboat есть два места, где я использую jQuery для добавления кое-каких деталей. И так как они демонстрируют лишь малую часть возможностей jQuery, я решил, что будет полезно показать, насколько просто и удобно добавлять эффекты и работать со стандартными шаблонами пользовательского интерфейса тем из нас, кто силен в CSS.

### Всплывающее окно «Advanced Options»

Если вы посмотрите на строку поиска «Find a Location» в шаблоне Tugboat (рис. 7.13), то заметите справа внизу небольшую ссылку «Advanced Options». Это стандартное решение, которое используется во многих интерфейсах, когда вы хотите, чтобы при нажатии на ссылку динамически открывался скрытый блок контента, дополнительные поля формы и т. п. При повторном нажатии на ссылку окно закрывается.

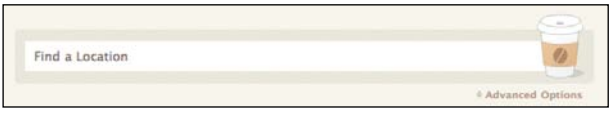



**Рис. 7.13.** Обратите внимание на ссылку «Advanced Options»

В шаблоне Tugboat ссылка открывает (очень старую) карту, на которой пользователю предлагается задать свое местоположение (рис. 7.14).


**Рис. 7.14.** Щелкните на ссылке, чтобы открыть эту карту

В такой ситуации JavaScript оказывается отличным помощником: карта (которая может постоянно присутствовать в разметке независимо от того, отображается она на странице или нет) волшебным образом появляется и исчезает при нажатии на ссылку — и при этом страница не обновляется. Я уверен, что вам уже приходилось работать с такими всплывающими элементами.

Раньше я бы погуглил скрипт, который умел бы показывать и скрывать элемент, а потом немного изменил бы его в соответствии с моей разметкой и стилем. Я, скорее всего, делал бы так каждый раз при необходимости реализовать такой трюк, и, возможно, при этом я все время использовал бы разные скрипты. Короче говоря, я был настоящим фанатом скриптов.

Сейчас в jQuery есть нечто подобное (наряду с другими преимуществами). jQuery позволяет не только легко показывать и скрывать выбранные элементы, но и без проблем добавлять к этим действиям эффекты. Давайте посмотрим, как это сделать.

## Подключение файла jQuery

После того как вы загрузите последнюю версию jQuery (один файл с расширением *.js*), укажите путь к файлу в теге `<head>` документа.

```
<head>
   ...
   <script src="js/jquery-1.3.2.min.js"
     type="text/javascript"></script>
</head>
```

Чтобы выиграть в скорости работы, мы используем в шаблоне Tugboat сжатую версию jQuery (*.min.js*).

Теперь, когда скрипт добавлен, вы можете насладиться всеми замечательными возможностями jQuery. Далее мы напишем небольшую функцию, позволяющую переключать режим отображения элемента Advanced Options.

### Создание карты

Мы хотим включить в разметку формы весь контент, который должен появиться на экране после нажатия на ссылку. Следовательно, мы напишем код для карты в том виде, в котором она должна появиться, а затем скроем ее с помощью CSS.

```
<form class="group" action="/">
   <fieldset>
      <img src="img/cup.png" alt="coffee cup" />
      <input name="location" type="text"
        value="Find a Location" />
   </fieldset>
   <fieldset>
      <label>Locate Yourself on a Map</label>
      <img src="img/map.jpg" alt="map" />
      <img src="img/place-marker.png"
        alt="marker" />
   </fieldset>
   <div>
      <a href="#map">Advanced Options</a>
   </div>
</form>
```

**ПРИМЕЧАНИЕ**

Поскольку Tugboat является вымышленным сайтом, карта является статическим, нефункциональным изображением. Возможно, в один прекрасный день, когда треугольные и фигурные скобки останутся в далеком прошлом, я на самом деле открою магазин и буду заваривать зерна и следить за пенкой. А пока…

Основная разметка формы готова, и теперь мы добавим атрибуты `id`, чтобы задать оформление.

```
<form id="loc" class="group" action="/">
   <fieldset id="location">
```

```
      <img id="cup" src="img/cup.png" alt="coffee
        cup" />
      <input id="loc-text" name="location"
        type="text" value="Find a Location" />
   </fieldset>
<fieldset id="map">
      <label>Locate Yourself on a Map</label>
      <img src="img/map.jpg" alt="map" id="map-
        img" />
      <img src="img/place-marker.png"
        alt="marker" id="marker" />
   </fieldset>
   <div id="loc-adv">
      <a href="#map">Advanced Options</a>
   </div>
</form>
```

### Идентификация элементов

С помощью этих семантических атрибутов `id` мы можем задать расположение и оформление различных элементов формы. Но более важно то, что использование идентификатора `#map` позволит нам скрыть карту средствами CSS с помощью правила `display: none;`.

```
#map {
   display: none;
   margin: 0 1em;
   padding: 2em;
   background: #d5d4c7 url(../img/map-shadow.gif)
     repeat-x top left;
   }
```

Хотя после добавления таких стилей карта выглядит и ведет себя так, как мы хотели, при первой загрузке карта тоже будет скрыта из-за того самого правила `display: none;` в начале таблицы стилей.

Вы помните, что в качестве значения атрибута `href` для ссылки «Advanced Options» используется идентификатор `#map`?

```
<div id="loc-adv" >
   <a href="#map">Advanced Options</a>
</div>
```

Благодаря этому ссылка будет иметь смысл даже в том случае, если CSS (или JavaScript) не поддерживается, поскольку она

не указывает на другой URL; вместо этого она является якорем, который показывает, в каком месте на странице находится карта. Давайте позволим jQuery вмешаться в этот процесс и добавить возможность скрывать и показывать карту щелчком на ссылке.

### Создание функции jQuery

Теперь, когда карта скрыта, ссылка находится на своем месте, а нужные элементы снабжены идентификаторами, мы можем написать небольшую простую функцию jQuery, которая добавит события `hide`/`show` там, где это необходимо.

Как я уже говорил, я люблю jQuery по той причине, что синтаксис селекторов хорошо знаком тем, кто привык работать с CSS. Осуществление перехода от одного элемента к другому и добавление событий — это так же просто, как создание селектора для таблицы стилей.

К примеру, ниже приведен небольшой скрипт, который переключает режим отображения `#map`. В нем используется `toggle();` — встроенная функция jQuery, которая скрывает выбранный элемент, если он отображается, и показывает, если он скрыт. Если использовать такой эффект при нажатии на конкретную ссылку (в данном случае на ссылку `<a>` внутри тега `<div id="loc-adv">`), мы получим то, что нам нужно: при нажатии на «Advanced Options» будет появляться и исчезать дополнительное окно.

```
<head>
   <script src="js/jquery-1.2.6.min. js"
     type="text/javascript"></script>
   <script type="text/javascript">
      // Переключение режима отображения
      // Advanced Options
      $(document).ready(function(){
         $("#loc-adv a").click(function(){
            $("#map").toggle();
            return false;
         });
      });
   </script>
</head>
```

**ПРИМЕЧАНИЕ**

Я добавил эту функцию в тег <head> документа (после импортирования основного файла jQuery), но вы можете также хранить ваши функции во внешнем файле.

Благодаря этому короткому коду на JavaScript, пользователь может при желании показать или скрыть карту нажатием на ссылку «Advanced Options». Как видите, добавлять такого рода возможности не сложнее, чем писать CSS-объявления; при этом разметка остается чистой и не содержит фрагментов кода на JavaScript внутри тегов. jQuery выполняет за вас всю тяжелую работу; в частности, она обходит DOM, выискивая все нужные элементы и применяя к ним соответствующие функции.

### Все становится еще интереснее с функцией slideToggle ()

Еще одно существенное преимущество jQuery состоит в том, что эта технология не только позволяет осуществлять поиск по DOM привычным способом, но и предоставляет в ваше распоряжение множество хитроумных приемов.

Я приведу простейший пример: чтобы скрывать и показывать карту, давайте будем вместо `toggle();` использовать функцию `slideToggle();`, которая добавляет к событиям `show`/`hide` эффект «скольжения». При нажатии на Advanced Options карта будет плавно раскрываться вниз под полем Find a Location (рис. 7.15). Это потрясающий эффект, и чтобы его добавить, нужно всего лишь заменить в нашей функции одну строку кода; это возможно, благодаря встроенным эффектам jQuery.

```
<head>
   <script src="js/jquery-1.2.6.min.js"
     type="text/javascript"></script>
   <script type="text/javascript">
      // Переключение режима отображения Advanced
        Options
      $(document).ready(function(){
         $("#loc-adv a").click(function(){
            $("#map").slideToggle("slow");
            return false;
         });
      });
   </script>
</head>
```


**Рис. 7.15.** И снова очень жаль, что в книгу нельзя вставить видео. Попробуйте представить себе, как карта плавно раскрывается вниз от строки поиска местоположения. Хотя вживую это выглядит гораздо более захватывающе

Вы, наверное, заметили у функции `slideToggle()` параметр `slow`. Это одно из трех значений, задающих скорость анимационного эффекта «скольжения» (другие два значения — это `normal` и `fast`).

Тому, кто работает с jQuery, доступно множество различных эффектов — включая переключение режима отображения элемента, эффекты «скольжения» и «растворения», а также свои собственные анимационные эффекты. Все они требуют простого синтаксиса и используют основной файл jQuery, который вы импортируете в документ.

## ИСПОЛЬЗОВАНИЕ JQUERY ДЛЯ ДОБАВЛЕНИЯ КЛАССА.LAST

Я снова обращаюсь за помощью к jQuery для шаблона Tugboat: на этот раз я хочу добиться автоматического добавления класса `.last` в определенные списки на странице.

Что я имею в виду под классом `.last`? Думаю, вам знакома такая ситуация: у вас есть список, в котором для каждого элемента заданы поля, отступы и, возможно, нижняя граница (чтобы отделять элементы друг от друга).

Но для последнего элемента приходится использовать немного другое оформление: например, требуется уменьшить нижний отступ и/или удалить нижнюю границу. И я часто использую для последнего элемента тег `<li class="last">`, чтобы переопределить стиль, который используется для него по умолчанию. Но так как список может изменяться или увеличиваться в размере, необходимо внимательно следить за тем, чтобы этот класс всегда был добавлен к самому последнему элементу списка.

### Добавление.last к списку Press

В правом поле шаблона Tugboat расположен раздел «Press» — список самых важных и захватывающих цитат из прессы (рис. 7.16).

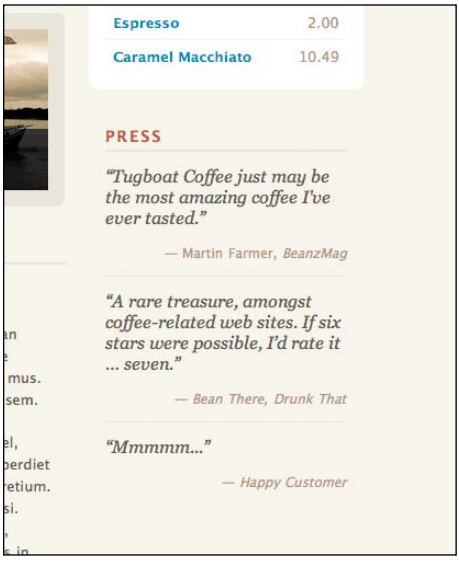



**Рис. 7.16.** Раздел Press

Разметка этого небольшого раздела представляет собой неупорядоченный список элементов `<blockquote>`, каждый из которых состоит из цитаты и источника.

```
<ul class="press">
   <li>
      <blockquote>
         <p>&#8220; Tugboat Coffee just may be
           the most amazing coffee I've ever
           tasted. &#8221;</p>
         <p class="author">&#8212; Martin Farmer,
           <cite>BeanzMag</cite></p>
      </blockquote>
   </li>
   <li>
      <blockquote>
         <p>&#8220; A rare treasure, amongst
           coffee-related web sites. If six stars
           were possible, I&#8217; d rate it
           &#8230; seven. &#8221;</p>
```

```
            <p class="author">&#8212; <cite>Bean
              There, Drunk That</cite></p>
         </blockquote>
      </li>
      <li>
         <blockquote>
            <p>&#8220; Mmmmm...&#8221;</p>
            <p class="author">&#8212; <cite>Happy
              Customer</cite></p>
         </blockquote>
      </li>
</ul>
```

Таблицы стилей для цитат содержат почти минимальный набор правил: для каждого элемента списка заданы отступы и поля, а внизу добавлена горизонтальная линия.

Для элементов `<blockquote>` и сведений об авторе также существуют отдельные таблицы стилей (в основном для оформления текста и задания отбивки).

```
ul.press li {
   margin: 1em 0;
   padding: 0 0 1em 0;
   border-bottom: 1px solid #e2e1d4;
   }

ul.press li blockquote {
   font-family: Georgia, serif;
   font-size: 1.2em;
   font-style: italic;
   line-height: 1.5em;
   }

ul.press li blockquote p {
   margin: 0 0 1em 0;
   }

ul.press li blockquote p. author {
   margin: 0;
   font-family: "Lucida Grande", "Lucida Sans
     Unicode", "Lucida Sans", Helvetica, Arial,
     sans-serif;
   font-size:.8em;
   text-align: right;
   color: #9c836e;
```

```
    }
ul. press li blockquote p. author a {
    font-weight: bold;
    }
```

В результате (рис. 7.17) у каждого элемента есть нижняя граница, в том числе и у последнего. Но вообще-то мы хотели, чтобы у последнего элемента граница (и поля) отсутствовали.

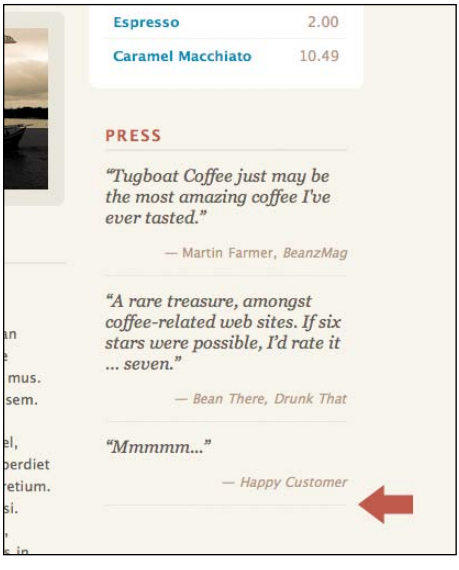



**Рис. 7.17.** У всех элементов, включая последний, есть нижняя граница

Чтобы переопределить свойства `border-bottom` и `padding-bottom`, добавим к последнему элементу списка класс `.last`.

```
<ul class="press">
...
   <li class="last">
      <blockquote>
         <p>&#8220; Mmmmm...&#8221;</p>
         <p class="author">&#8212; <cite>Happy
           Customer</cite></p>
      </blockquote>
   </li>
</ul>
```

А затем напишем короткое объявление, отменяющее границу и нижнее поле для этого класса.

```
ul.press li.last {
   padding-bottom: 0;
   border-bottom: none;
   }
```

Такое решение, в общем-то, делает все, что нужно. Но в тех случаях, когда список редактируется, увеличивается или уменьшается в размере или еще как-то изменяется, нужно не забывать перемещать этот класс в последний элемент — а это доставляет определенные неудобства. Кроме того, если серверная часть CMS или среда разработки автоматически генерирует список, вам придется позаботиться о том, чтобы этот класс был добавлен к последнему элементу.

То есть в некоторых случаях совсем нетрудно добавить этот класс вручную. Но порой это доставлет неудобства или приводит к тому, что какой-то список оказывается пропущенным.

Так как мы уже умеем использовать jQuery для создания скриптов, мы можем вообще не добавлять этот класс в разметку. Пусть jQuery сделает это автоматически с помощью поиска элементов в DOM; во время загрузки страницы jQuery динамически найдет последний элемент списка и добавит в него этот класс. jQuery берет на себя ответственность за изменения, происходящие со списком, причем независимо от того, были они проведены вручную или программными средствами (серверной системой).

Мы не будем ничего менять в таблице стилей. Класс `.last` должен убирать нижнее поле и границу.

```
ul.press li.last {
   padding-bottom: 0;
   border-bottom: none;
   }
```

А вместо того, чтобы добавлять класс в разметку, мы напишем небольшую функцию jQuery, которая сделает это за нас (мы добавили ее в тег `<head>` рядом с предыдущей).

```
<head>
   <script src="js/jquery-1.2.6.min.js"
      type="text/javascript"></script>
   <script type="text/javascript">
      // Переключение режима отображения Advanced
        Options
      $(document).ready(function(){
```

```
            $("#loc-adv a").click(function(){
               $("#map").slideToggle("slow");
               return false;});});

// Добавление класса .last в определенные списки
      $(document).ready(function(){
         $("ul.press li: last").addClass("last");
      });
   </script>
</head>
```

Используя фильтр селектора jQuery (`:last`), мы на самом деле подразумеваем следующую команду: найти последний элемент `<li>` в списке `<ul class="press">` и добавить к нему класс `.last`.

Готово.

### Внешний вид или поведение?

В каком-то смысле с помощью JavaScript мы заполняем дыры, для которых, вероятнее всего, лучше было бы использовать CSS. В то время как JavaScript лучше всего подходит для управления *поведением* страницы, здесь мы используем его только для создания *внешнего вида*. Вообще-то это не лучшая идея. Но в данном случае (если вы уже применяете jQuery для решения других задач) это всего лишь маленькая функция, которая, кстати, несильно ухудшит качество страницы, если JavaScript будет отключен или не будет поддерживаться (просто внизу появится граница и слишком большое поле).

**СОВЕТ**

Если вы хотите узнать более подробно о том, как еще можно использовать JavaScript вместо CSS, почитайте статью Эрика Мейера «JavaScript нас всех спасет» («JavaScript Will Save Us All» *http://meyerweb.com/eric/thoughts/2008/10/22/javascript-will-save-us-all/*).

### А как насчет :last-child?

В CSS3 также включен псевдокласс `:last-child`, который в общем-то решил бы проблему еще более эффективно. Нам опять не пришлось бы добавлять класс в разметку; достаточно было бы изменить объявление, которое отменяет границу и поле, используя этот псевдокласс.

```
ul.press li:last-child {
   padding-bottom: 0;
   border-bottom: none;
   }
```

Безусловно, *это* оптимальный вариант. Но, к сожалению, этот псевдокласс не поддерживается в Internet Explorer (а также в Safari 3.0 и более ранних версиях). Если вас устраивает появление границы в этих браузерах, используйте прогрессивное оформление, не задумываясь.

**СОВЕТ**

Для тех, кому интересно: псевдокласс `:first-child`, выполняющий обратное действие, был включен в CSS2, и он не поддерживается в IE6, а его реализация в IE7 и IE8 оставляет желать лучшего. Подробное сравнение совместимости приводится на *http://www.quirksmode.org/css/contents. html*.

### ГАЛОПОМ ПО ЕВРОПАМ

Те два маленьких способа использования jQuery в шаблоне Tugboat, о которых я сейчас рассказал, очень поверхностно демонстрируют то, что на самом деле позволяет сделать это фреймворк. Помимо основного файла jQuery включает расширенный фреймворк, созданный мощным сообществом разработчиков на JavaScript, который позволяет выполнять множество разнообразных задач с помощью плагинов.

Независимо от того, какой JavaScript-фреймворк вы используете (возможно, вы даже пишете скрипты сами), основная мысль состоит вот в чем: JavaScript может оказаться ценным помощником в создании тех деталей, которые существенно повышают эффективность взаимодействия интерфейса с пользователем. Для меня jQuery — это привычный, интересный и изящный способ задать поведение дизайна и добавить элементы диалога.

## Движущийся фон (многоуровневый скроллинг для ленивых)

Напоследок мы совершим «невидимое прикосновение» (привет Филу Коллинзу) к тому, что является своего рода приятным обобщением понятия тонкостей мастерства.

Для кого-то эффект так и останется незамеченным, а у остальных непременно вызовет улыбку… «Многоуровневый скроллинг»

фона — простой способ сделать подвижность Сети ее же инструментом. Хотя, безусловно, добавление *движения* к тому, что кажется статичным, требует дополнительных затрат времени.

## ПАРТИЗАНСКАЯ ТАКТИКА

Я впервые увидел эффект «многоуровневого скроллинга» на сайте компании Clearleft для Silverback (рис. 7.18) — приятной программы, которая позволяет проводить тестирование юзабилити «не отходя от кассы».

**ПРИМЕЧАНИЕ**

Наверное, это все-таки длилось меньше часа.

Хотя это *действительно* превосходная программа, особое внимание я обратил на то, как выполнен дизайн сайта. В частности, меня поразили растения, нависающие над гориллой: при изменении размера окна браузера становились будто бы трехмерными. Если увеличить или уменьшить ширину окна, три слоя с изображениями растений перемещаются вдоль горизонтальной оси, причем с разной скоростью. А самый верхний слой размыт, что усиливает эффект глубины. Удивительно! И я, наверное, целый час растягивал и сжимал окно, тихонько посмеиваясь.

**СОВЕТ**

Термин «многоуровневый скроллинг» («parallax scrolling») пришел к нам из аркад 80-х годов, где фоновые изображения двигались медленнее, чем изображения, находящиеся на переднем плане. За счет этого в плоской (2D) игре создавался эффект трехмерности. Более подробно об этом можно прочитать на *http://en.wikipedia.org/wiki/Parallax_scrolling*.

### Вроде бы невидимые детали

Пространственный эффект при изменении размера окна браузера — это один из лучших примеров тонкого мастерства. Его реализация требует много сил и внимания, однако результат пользователь может так и не заметить. Существенную часть этой книги мы посвятили разговору о мелочах, которые могут различаться в зависимости от браузера, и о том, как важно побороть в себе стремление к однородности дизайна.

Эффект «многоуровневости» использует тот факт, что ширина окна является своего рода движущейся мишенью. Это всегда переменная величина, на которую невозможно опираться. Но вместо того чтобы привязывать графические элементы к тем или иным частям страницы, почему бы не сделать их независимыми, позволив взаимодействовать друг с другом?


**Рис. 7.18.** Программа для тестирования юзабилити Silverback, созданная компанией Clearleft (http: //silverbackapp. com/). Попробуйте изменить размер окна и следите за тем, как свисающие растения перемещаются с разными скоростями

«Резиновые» сетки, о которых Итан говорил в предыдущей главе, являются еще более экстремальным воплощением этой идеи.

### Как это делается

Пол Аннет рассказывает о том, как он создавал эффект «многоуровневости» на сайте Silverback, в статье «Как повторить эффект многоуровневого скроллинга с сайта Silverback» («How to recreate Silverback's parallax effect» *http://carsonified.com/blog/features/how-to-recreate-silverbacks-parallax-effect/*). Он использовал три слоя PNG-изображений, расположение которых по горизонтали задается с помощью различных процентных значений.

Я скажу лишь общие слова по поводу кода: если в качестве значений начального положения трех слоев фона использовать `20%`, `40%` и `150%`, то повторяющиеся по горизонтали слои будут двигаться при изменении ширины окна браузера с разными скоростями.

```
body {
   background: url(../img/vines-back.png)
     repeat-x 20% 0;
   }
#container {
   background: url(../img/vines-middle.png)
```

```
      repeat-x 40% 0;
   }
#container-inner {
   background: url(../img/vines-front.png)
      repeat-x 150% 0;
   }
```

Это очень хитрый прием; используя его, вы практически не ограничены в выборе слоев, их количества и скоростей, с которыми они двигаются.

### Эксперименты с отрицательными процентными значениями

В конце своей статьи Пол говорит о том, что использование отрицательных процентных значений приводит к интересному результату. И здесь он вспоминает сайт The Rissington Podcast (*http://therissingtonpodcast.co.uk/*), уже упоминавшийся в главе 5, который тоже может похвастаться эффектом «многоуровневого скроллинга».

Если использовать отрицательные значения, то изображения будут двигаться влево, то есть часто не в том направлении, в котором меняется размер окна (если вы возьметесь за правый нижний угол окна и потащите его вправо, изображения будут двигаться в противоположном направлении). И, как поясняет Пол, чем выше процентное значение (отрицательное), тем быстрее будет двигаться изображение.

## ЭФФЕКТ «МНОГОУРОВНЕВОСТИ» ДЛЯ ЛЕНИВЫХ

Немного упростив метод, описанный Полом Аннетом, я стал использовать отрицательные процентные значения для задания расположения повторяющихся по горизонтали фоновых изображений. Даже в самой простой ситуации, когда всего одно изображение заполняет фон, движение картинки в направлении, противоположном изменению размера окна, создает эффект, похожий на «многоуровневый скроллинг» — только сделать это гораздо проще.

### Как это используется в шаблоне Tugboat

К примеру, в шапке шаблона Tugboat есть фоновое изображение, на котором вы можете разглядеть небо, облака и холмы из кофе (рис. 7.19). Это картинка размером 1024×56 пикселов, которая повторяется по горизонтали, чтобы независимо

от величины окна шапка занимала всю его ширину и выглядела как единое целое.

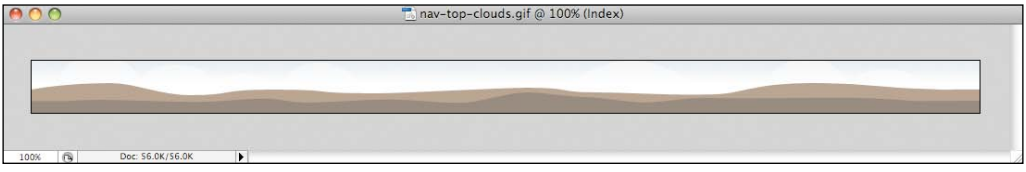



**Рис. 7.19.** Фоновое изображение, которое будет повторяться по горизонтали

Это изображение накладывается на шапку, цвет которой совпадает с цветом нижней части холма, благодаря чему оно может иметь фиксированную высоту, в то время как шапка будет растягиваться в зависимости от размера окна.

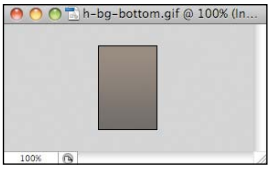



**Рис. 7.20.** Цвет градиента меняется от более темного коричневого оттенка к более светлому, совпадающему с цветом шапки

Кроме того, под шапкой мы добавили повторяющееся градиентное изображение (рис. 7.20). Его цвет меняется от более темного оттенка коричневого (внизу) к более светлому, совпадающему с цветом шапки (вверху).

Для содержимого шапки нам понадобится два элемента-контейнера, чтобы правильно расположить все наши фоновые изображения: облака и холмы — сверху, градиент — внизу (рис. 7.21).

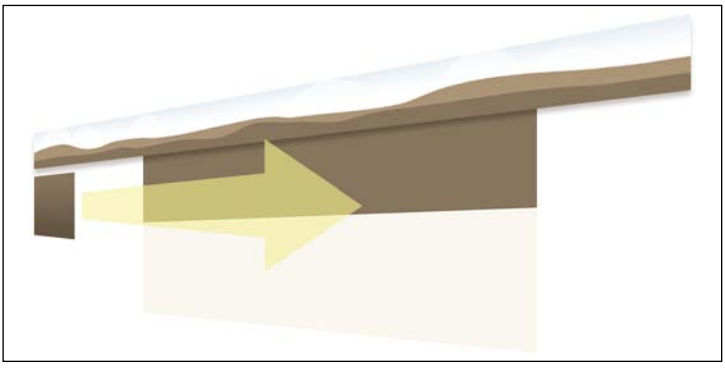


**Рис. 7.21.** Объемная картинка: изображение с облаками/холмами над коричневой шапкой и повторяющееся градиентное изображение под ней

## Разметка шапки шаблона Tugboat

Для создания структуры шапки шаблона Tugboat будем использовать такую разметку (в ней мы зададим логотип, слоган, графические элементы и список панели навигации).

```
<div id="header" class="group"><div id="header-
    inner">
  <div id="nav" class="group">
    <div id="logo" class="group">
      <h1><img src="img/logo-lofi.gif"
        alt="Tugboat" /></h1>
      <p>Pulling you along since 2009.</p>
    </div>
    <img src="img/tugboat.png" alt="tugboat"
      id="tugboat" />
    <ul class="group">
      <li><a href="#"><strong>Home <em>News
        and Trends</em></strong></a></li>
      <li><a href="#"><strong>Our Coffee
        <em>Handcrafted Beans</em></strong>
        </a></li>
      <li><a href="#"><strong>Store <em>Shop
        Online</em></strong></a></li>
      <li><a href="#"><strong>About
        <em>Company Info</em></strong></a>
        </li>
    </ul>
  </div>
</div></div> <!-- /header -->
```

Обратите внимание на *два* внешних контейнера `<div>`, которые обозначены как `#header` и `#header-inner`. Это те самые элементы, с помощью которых мы будем добавлять фоновые изображения, а также зададим коричневый цвет фона.

### Добавление фоновых изображений

Сначала добавим градиентное изображение, которое должно располагаться вдоль нижнего края шапки и над коричневым фоном. Чтобы аккуратно отделить шапку от светло-коричневого фона страницы, создадим белую границу шириной в 1 пиксел.

```
#header {
  border-bottom: 1px solid #fff;
  background: #766557 url(../img/h-bg-bottom.gif)
    repeat-x bottom left;
  }
```

Далее добавим повторяющееся по горизонтали изображение облаков и холмов, которое должно располагаться вдоль

верхнего края шапки. Для этого зададим его в качестве фона для элемента `#header-inner`.

```
#header {
   border-bottom: 1px solid #fff;
   background: #766557 url(../img/h-bg-bottom.gif)
     repeat-x bottom left;
   }
#header-inner {
   padding: 70px 0 0 0;
   background: url(../img/nav-top-clouds.gif)
     repeat-x top left;
   }
```

На рисунке 7.22 показан результат заполнения фона этими изображениями. Такой подход гарантирует нам, что независимо от размера окна браузера облака/холмы и градиент будут занимать всю ширину окна и выглядеть единым целым.


**Рис. 7.22.** Результат заполнения фона изображениями

### Отрицательные процентные значения для изображений, повторяющихся по горизонтали

Теперь вернемся к методу создания эффекта «многоуровневости» для ленивых. Все, что нам нужно, — это задать расположение изображения с облаками/холмами по горизонтали с помощью *отрицательных* процентных значений (как я уже говорил, чем больше это значение, тем выше скорость, с которой будет двигаться картинка).

```
#header-inner {
   padding: 70px 0 0 0;
   background: url(../img/nav-top-clouds.gif)
     repeat-x -80% 0;
   }
```

Итак, мы задали начальное положение фонового изображения: `80%` по горизонтали и `0` (то же самое, что `top`) по вертикали. Вот и всё.

Теперь когда пользователи будут менять ширину окна, облака и холмы будут двигаться в противоположном направлении. Мы создали эффект «псевдо-многоуровневости», причем за треть цены!

### Признак мастерства

Опять же, не каждый пользователь заметит эту небольшую деталь, но зато ее невероятно *просто* реализовать. Попробуйте всегда использовать отрицательные процентные значения для изображений, повторяющихся по горизонтали. Я, например, буду приятно удивлен, увидев этот эффект на каком-нибудь сайте.

## Заключение

Ну что ж, друзья, книга подошла к концу. Пока я это печатаю, дни становятся длиннее, щебечут птицы, распускаются цветы. Что-то новое чувствуется в весеннем воздухе (я не имею в виду пыльцу, от которой все чихают).

У меня есть ощущение, то в мире веб-дизайна происходит множество удивительных вещей. И я надеюсь, что эта книга пробудит в вас желание экспериментировать с чем-нибудь новым и очень интересным — и при этом вы будете помнить о принципах пуленепробиваемости, которые превращают хороший веб-дизайн в первоклассный.

Я надеюсь, вам доставило удовольствие слушать, как Итан и я рассказывали о том, какой принцип является для нас самым важным при разработке дизайнерского проекта: мелочи, конечно, играют роль, но еще большее значение имеет расстановка приоритетов. Значительная часть работы над дизайном — это убеждение себя в том, что вы все делаете правильно. И я надеюсь, что мне удалось помочь вам сделать еще один шаг в этом направлении.

А что касается кофейной компании Tugboat — как я уже говорил, когда мне вконец наскучат треугольные и фигурные скобки, я, возможно, на самом деле открою кофейню. Первая чашка — за счет заведения.

Пока!

***Д. Седерхольм, И. Маркотт***

# CSS ручной работы

**Библиотека специалиста**

*Перевела с английского Т. Качковская*


| | |
|---|---|
| Заведующий редакцией | *А. Кривцов* |
| Руководитель проекта | *А. Юрченко* |
| Ведущий редактор | *Ю. Сергиенко* |
| Научный редактор | *А. Гребеньков* |
| Литературный редактор | *В. Шрага* |
| Корректор | *В. Нечаева* |
| Верстка | *Ю. Орлова* |

Подписано в печать 10.09.10. Формат 70x100/16. Усл. п. л. 19,35.
Тираж 2000. Заказ 0000.

ООО «Лидер», 194044, Санкт-Петербург, пр. Б. Сампсониевский, дом 29а.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2;
95 3005 — литература учебная.

Отпечатано по технологии CtP в ОАО «Печатный двор» им. А. М. Горького.
197110, Санкт-Петербург, Чкаловский пр., 15.

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

**УВАЖАЕМЫЕ ГОСПОДА!**
**КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»**
**ВЫ МОЖЕТЕ ПРИОБРЕСТИ**
**ОПТОМ И В РОЗНИЦУ У НАШИХ**
**РЕГИОНАЛЬНЫХ ПАРТНЕРОВ.**

## ДАЛЬНИЙ ВОСТОК

**Владивосток**
«Приморский торговый дом книги»
тел./факс: (4232) 23-82-12
e-mail: bookbase@mail.primorye.ru

**Хабаровск**, «Деловая книга», ул. Путевая, д. 1а
тел.: (4212) 36-06-65, 33-95-31
e-mail: dkniga@mail.kht.ru

**Хабаровск**, «Книжный мир»
тел.: (4212) 32-85-51, факс: (4212) 32-82-50
e-mail: postmaster@worldbooks.kht.ru

**Хабаровск**, «Мирс»
тел.: (4212) 39-49-60
e-mail: zakaz@booksmirs.ru

## ЕВРОПЕЙСКИЕ РЕГИОНЫ РОССИИ

**Архангельск**, «Дом книги», пл. Ленина, д. 3
тел.: (8182) 65-41-34, 65-38-79
e-mail: marketing@avfkniga.ru

**Воронеж**, «Амиталь», пл. Ленина, д. 4
тел.: (4732) 26-77-77
http://www.amital.ru

**Калининград**, «Вестер»,
сеть магазинов «Книги и книжечки»
тел./факс: (4012) 21-56-28, 6 5-65-68
e-mail: nshibkova@vester.ru
http://www.vester.ru

**Самара**, «Чакона», ТЦ «Фрегат»
Московское шоссе, д.15
тел.: (846) 331-22-33
e-mail: chaconne@chaccone.ru

**Саратов**, «Читающий Саратов»
пр. Революции, д. 58
тел.: (4732) 51-28-93, 47-00-81
e-mail: manager@kmsvrn.ru

## СЕВЕРНЫЙ КАВКАЗ

**Ессентуки**, «Россы», ул. Октябрьская, 424
тел./факс: (87934) 6-93-09
e-mail: rossy@kmw.ru

## СИБИРЬ

**Иркутск**, «ПродаЛитЪ»
тел.: (3952) 20-09-17, 24-17-77
e-mail: prodalit@irk.ru
http://www.prodalit.irk.ru

**Иркутск**, «Светлана»
тел./факс: (3952) 25-25-90
e-mail: kkcbooks@bk.ru
http://www.kkcbooks.ru

**Красноярск**, «Книжный мир»
пр. Мира, д. 86
тел./факс: (3912) 27-39-71
e-mail: book-world@public.krasnet.ru

**Новосибирск**, «Топ-книга»
тел.: (383) 336-10-26
факс: (383) 336-10-27
e-mail: office@top-kniga.ru
http://www.top-kniga.ru

## ТАТАРСТАН

**Казань**, «Таис»,
сеть магазинов «Дом книги»
тел.: (843) 272-34-55
e-mail: tais@bancorp.ru

## УРАЛ

**Екатеринбург**, ООО «Дом книги»
ул. Антона Валека, д. 12
тел./факс: (343) 358-18-98, 358-14-84
e-mail: domknigi@k66.ru

**Екатеринбург**, ТЦ «Люмна»
ул. Студенческая, д. 1в
тел./факс: (343) 228-10-70
e-mail: igm@lumna.ru
http://www.lumna.ru

**Челябинск**, ООО «ИнтерСервис ЛТД»
ул. Артиллерийская, д. 124
тел.: (351) 247-74-03, 247-74-09,
247-74-16
e-mail: zakup@intser.ru
http://www.fkniga.ru, www.intser.ru