

НОВАЯ
БОЛЬШАЯ КНИГА
CSS

Дэвид Макфарланд



O'REILLY®

ПИТЕР®

CSS



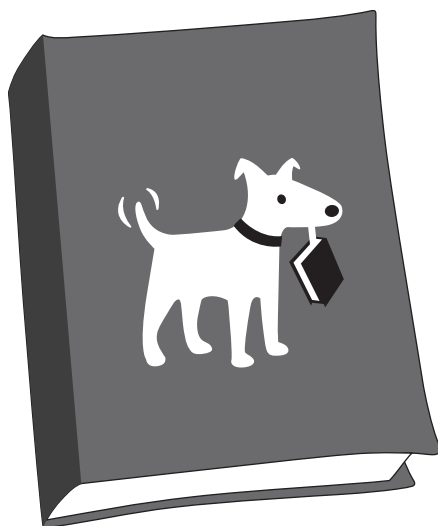
David Sawyer McFarland

O'REILLY®

Beijing | Cambridge | Farnham | Köln | Sebastopol | Tokyo

НОВАЯ
БОЛЬШАЯ КНИГА

CSS



Дэвид Макфарланд



Санкт-Петербург · Москва · Екатеринбург · Воронеж
Нижний Новгород · Ростов-на-Дону
Самара · Минск

2016

ББК 32.988.02-018
УДК 004.738.8
М17

Макфарланд Д.

М17 Новая большая книга CSS. — СПб.: Питер, 2016. — 720 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-496-02080-0

Технология CSS3 позволяет создавать профессионально оформленные сайты, но тонкости этого языка могут оказаться довольно сложными даже для опытных веб-разработчиков. Полностью переработанное четвертое издание этой книги поможет вам поднять навыки работы с HTML и CSS на новый уровень; оно содержит множество ценных советов, описаний приемов, а также инструкции, написанные в стиле справочного руководства. Веб-дизайнеры, как начинающие, так и опытные, при помощи этой книги быстро научатся создавать красивые веб-страницы, которые молниеносно загружаются как на ПК, так и на мобильные устройства.

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

ББК 32.988.02-018
УДК 004.738.8

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

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

ISBN 978-1491918050 англ.
ISBN 978-5-496-02080-0

© Copyright © 2015 David Sawyer McFarland. All rights reserved.
© Перевод на русский язык ООО Издательство «Питер», 2016
© Издание на русском языке, оформление ООО Издательство «Питер», 2016
© Серия «Бестселлеры O'Reilly», 2016

Краткое содержание

Об авторе	13
О творческой команде	14
Благодарности	15
Введение	16

Часть I. Основы CSS

Глава 1. HTML и CSS	28
Глава 2. Создание стилей и таблиц стилей	46
Глава 3. Селекторы: выбор форматлируемых элементов	65
Глава 4. Механизм наследования стилей	110
Глава 5. Управление сложной структурой стилей: каскадность	120

Часть II. Применение CSS

Глава 6. Форматирование текста	142
Глава 7. Поля, отступы, границы	211
Глава 8. Добавление графики на веб-страницы	257

Глава 9. Построение навигационной системы сайта	309
Глава 10. Преобразования, переходы и анимация с помощью CSS	349
Глава 11. Форматирование таблиц и веб-форм	387

Часть III. Верстка страниц с помощью CSS

Глава 12. Введение в CSS-верстку	414
Глава 13. Макеты на основе обтекаемых элементов	427
Глава 14. Позиционирование элементов на странице	462
Глава 15. Адаптивный веб-дизайн	491
Глава 16. Система модульной верстки Skeleton	528
Глава 17. Профессиональная flexbox-верстка	561

Часть IV. Профессиональные приемы CSS-верстки

Глава 18. Профессиональные приемы CSS-верстки	600
Глава 19. Профессиональный дизайн с помощью Sass	621

Часть V. Приложения

Приложение 1. Справочник свойств CSS	668
Приложение 2. Информационные ресурсы, посвященные CSS	706
Указатель	711

Оглавление

Об авторе	13
О творческой команде	14
Благодарности	15
Введение	16
Что такое CSS?	16
Что необходимо знать	16
HTML: структура языка	17
Типы документов	17
Как работают HTML-элементы	18
HTML5: больше элементов	19
Программное обеспечение, используемое для CSS	20
Об этой книге	22
Основы	23
Об ▶ этих ▶ стрелках	24
Соглашения, использованные в данной книге	25
Интернет-ресурсы	25
Примеры к книге	26

Часть I. Основы CSS

Глава 1. HTML и CSS	28
HTML: прошлое и настоящее	28
Верстка HTML-кода вместе с CSS	31
Важность объявления типа документа	43
Как работают каскадные таблицы стилей	44

Глава 2. Создание стилей и таблиц стилей	46
Анатомия стиля	46
Концепция таблиц стилей	49
Внутренние таблицы стилей.	50
Внешние таблицы стилей.	52
Практикум: создание стилей	53
Глава 3. Селекторы: выбор форматируемых элементов	65
Селекторы тегов	65
Классы: точное управление	67
Идентификаторы: отдельные элементы веб-страницы	71
Форматирование групп элементов	73
Форматирование вложенных элементов	75
Псевдоклассы и псевдоэлементы	80
Селекторы атрибутов.	84
Дочерние селекторы	86
Дочерние псевдоклассы.	90
Родственные селекторы	93
Селектор :target.	93
Селектор :not()	95
Практикум: использование селекторов.	96
Глава 4. Механизм наследования стилей	110
Что такое наследование?	110
Упрощение таблиц стилей через наследование	111
Ограничения наследования	112
Практикум: наследование	114
Глава 5. Управление сложной структурой стилей: каскадность	120
Каскадность стилей	120
Особенности каскадности: какие стили имеют преимущество	125
Управление каскадностью	129
Практикум: механизм каскадности	135

Часть II. Применение CSS

Глава 6. Форматирование текста	142
Использование шрифтов	142
Использование веб-шрифтов	148

Использование службы Google Fonts	163
Форматирование текста цветом	171
Изменение размера шрифта	175
Форматирование символов и слов	180
Добавление тени	184
Форматирование абзацев	186
Форматирование списков	193
Практикум: форматирование текста	198
Глава 7. Поля, отступы, границы	211
Понятие блочной модели	211
Управление размерами полей и отступов	213
Добавление границ	221
Установка цвета фона	225
Скругление углов	226
Добавление тени	229
Изменение высоты и ширины	232
Управление обтеканием контента с помощью плавающих элементов	239
Практикум: поля, фон и границы	244
Глава 8. Добавление графики на веб-страницы	257
Каскадные таблицы стилей и элемент <code>img</code>	257
Добавление фоновых изображений	258
Управление повтором фоновых изображений	263
Позиционирование фоновых изображений	264
Сокращенная запись свойства <code>background</code>	276
Использование множественных фоновых изображений	278
Использование градиентных фонов	280
Практикум: совершенствуем изображения	290
Практикум: создание фотогалереи	295
Практикум: использование фоновых изображений	300
Добавление на веб-страницу фонового изображения	300
Замена границ изображениями	302
Использование графики для маркированных списков	304
Персонализация боковой панели	306
Глава 9. Построение навигационной системы сайта	309
Выборка форматируемых ссылок	309
Форматирование ссылок	313

Создание панелей навигации	320
Использование ролловеров	330
Форматирование ссылок определенного типа	331
Практикум: форматирование ссылок	334
Практикум: создание панели навигации	340
Глава 10. Преобразования, переходы и анимация с помощью CSS	349
Преобразования	349
Переходы	360
Анимация	368
Практикум	380
Добавление анимации	382
Глава 11. Форматирование таблиц и веб-форм	387
Разумное применение таблиц	387
Форматирование таблиц	389
Форматирование строк и столбцов	394
Форматирование веб-форм	396
Практикум: форматирование таблиц	401
Практикум: форматирование веб-форм	406
 Часть III. Верстка страниц с помощью CSS	
Глава 12. Введение в CSS-верстку	414
Типы макетов веб-страниц	414
Принцип CSS-верстки	417
Стратегии верстки	421
Глава 13. Макеты на основе обтекаемых элементов	427
Использование обтекаемых элементов при верстке	431
Решение проблем с обтекаемыми элементами	435
Практикум: многоколоночные макеты	449
Глава 14. Позиционирование элементов на странице	462
Принципы работы свойств позиционирования	462
Эффективные стратегии позиционирования	476
Практикум: позиционирование элементов страницы	483

Глава 15. Адаптивный веб-дизайн.	491
Основы адаптивного веб-дизайна	491
Создание адаптивного дизайна веб-страницы	493
Медиазапросы	495
Гибкие сетки	503
Гибкие изображения	508
Практикум: адаптивный веб-дизайн	512
Глава 16. Система модульной верстки Skeleton	528
Принцип модульной сетки	528
Структурирование HTML-кода под модульную сетку	530
Использование системы модульной верстки Skeleton	532
Создание и именованние колонок	536
Практикум: использование системы модульной верстки	544
Глава 17. Профессиональная flexbox-верстка	561
Знакомство с методом flexbox-верстки	561
Свойства flex-контейнера	564
Свойства flex-элементов	574
Практикум: создание flexbox-макета	588

Часть IV. Профессиональные приемы CSS-верстки

Глава 18. Профессиональные приемы CSS-верстки.	600
Добавление комментариев	600
Организация стилей	602
Устранение конфликтов стилей в браузере	610
Использование селекторов потомков	614
Глава 19. Профессиональный дизайн с помощью Sass.	621
Понятие Sass	621
Интеграция Sass	623
Основы Sass	627
Организация стилей с помощью фрагментов Sass	631
Переменные Sass	635
Вложенные селекторы	639

Наследование (или расширение) свойств	645
Примеси	650
Использование медиазапросов.	658
Поиск и устранение ошибок с помощью карт CSS-кода	663

Часть V. Приложения

Приложение 1. Справочник свойств CSS	668
Значения свойств CSS	668
Свойства текста.	673
Свойства списков.	678
Отступы, границы и поля.	680
Фоны.	686
Компоновка макета	689
Свойства анимации, преобразований и переходов	696
Свойства таблицы	701
Прочие свойства	703
Приложение 2. Информационные ресурсы, посвященные CSS.	706
Справочники	706
Справочная информация по CSS	707
Подсказки, приемы и советы по CSS.	707
CSS-навигация.	708
CSS-верстка.	709
Демонстрационные сайты	709
Указатель	711

Об авторе



Дэвид Макфарланд (David McFarland) — веб-разработчик, преподаватель и автор. Создает сайты с 1995 года: именно тогда он разработал свой первый проект — онлайн-журнал для специалистов в области коммуникаций.

Дэвид преподавал веб-дизайн в Высшей школе журналистики в Беркли, Центре электронного искусства (Electronic Art) и Государственном университете Портленда. В настоящий момент он главный преподаватель на сайте онлайн-образования Treehouse (teamtreehouse.com).

О творческой команде

Нэн Барбер (Nan Barber) — редактор серии книг Missing Manuals («Исчерпывающее руководство»). Живет в Массачусетсе вместе со своим мужем. Ее электронный адрес: nanbarber@oreilly.com.

Мелани Ярброух (Melanie Yarbrough) — литературный редактор и композитор. Проживает и трудится в Кембридже, штат Массачусетс. Увлекается выпечкой и любит совершать прогулки на велосипеде вокруг города. Ее электронный адрес: myarbrough@oreilly.com.

Молли Ивс Бровер (Molly Ives Brower) — внештатный редактор и корректор. Любит Интернет с тех пор, как в 1990 году получила адрес BITNET. В наше время ее можно найти на сайте mjibrower.com или в социальной сети Twitter, где она известна под ником [@vintagereader](https://twitter.com/vintagereader). Ее электронный адрес: molly@mjibrower.com.

Рон Штраус (Ron Strauss) — составитель алфавитного указателя. Специализируется на составлении указателей для различных книг, посвященных информационным технологиям. Рон также талантливый скрипач. Он живет в Северной Калифорнии со своей женой и по совместительству коллегой Энни и карликовым пинчером Кенгой. Адрес электронной почты: rstrauss@mchsi.com.

Рич Костер (Rich Koster) — бета-ридер. Купил свой первый компьютер Mac (17-дюймовый MacBook Pro) в 2009 году и больше никогда не переходил на сторону пользователей Windows. В третьем издании книги «iPhone. Исчерпывающее руководство» Рич выступил техническим редактором. Он женат, воспитывает детей и в свободное время поддерживает собственноручно созданный сайт Disney Echo (DisneyEcho.emuck.com).

Благодарности

Большое спасибо всем, кто помогал в работе над этой книгой, включая моих студентов, которые всегда оценивали технические издания с позиции новичков. Благодарю своих технических редакторов, Дэниел Куинн (Daniel Quinn) и Дженнифер Дэвис (Jennifer Davis), которые предостерегли меня от досадных ошибок. Кроме того, мы все в долгу перед веб-дизайнерами, которые стали новаторами, используя CSS с творческим подходом, и поделились своими наработками в сообществе, посвященном веб-дизайну.

Спасибо Дэвиду Поугу (David Poque), который много лет назад помог мне начать это длинное приключение. Спасибо Нэн Барбер за приведение в порядок моих рукописей и прочую помощь в написании этой книги.

Дэвид Сойер Макфарланд

Введение

Каскадные таблицы стилей, или Cascading Style Sheets (CSS), обеспечивают творческую свободу в разметке и дизайне веб-страниц. Пользуясь CSS, вы сможете украсить текст страниц привлекательными заголовками, буквицами и рамками, как в красочных глянцевах журналах. Можно точно разместить и позиционировать изображения, сделать колонки и создать баннеры, выделить ссылки динамическими эффектами. Можно также добиться постепенного появления и исчезновения элементов, перемещения объектов по странице или медленного изменения цвета кнопки при прохождении над ней указателя мыши.

Вы думаете, что все это довольно сложно? *Напротив!* Каскадные таблицы стилей как раз и предназначены для упрощения процесса оформления веб-страниц. Следующие несколько страниц будут посвящены изучению основ CSS.

Что такое CSS?

CSS — это язык стилей. Он используется для того, чтобы придать страницам на HTML — фундаментальном языке Всемирной паутины — совершенный вид. Надеюсь, вы будете использовать каскадные таблицы стилей, чтобы сделать свои страницы идеальными. После прочтения этой книги вы сможете создавать красивые, функциональные и простые в использовании сайты.

Прежде чем перейти к изучению каскадных таблиц стилей, необходимо понять, что такое язык HTML.

Что необходимо знать

Эта книга предполагает, что вы уже знакомы с языком HTML. Подразумевается, что вы создали пару сайтов (или по крайней мере несколько веб-страниц) и знакомы с основными элементами, такими как `html`, `p`, `h1`, `table`, составляющими основу языка гипертекстовой разметки документов. CSS бесполезен без HTML, поэтому вы должны знать, как создать простейшую веб-страницу с использованием основных HTML-элементов.

Если вы раньше создавали веб-страницы на HTML, но чувствуете, что знания требуются освежить, вам поможет следующий раздел книги.

ПРИМЕЧАНИЕ

Если вы только знакомитесь с HTML и возможностями его применения на практике, то посетите следующие сайты: HTML Dog (tinyurl.com/oujmfqw) и W3Schools (w3schools.com/html). Если вам

больше нравится читать книги, обратитесь к руководствам по созданию сайтов: «HTML5. Недостающее руководство» Мэтью Макдональда (издательство «БХВ-Петербург») или «Изучаем HTML, XHTML и CSS» Элизабет Фримен и Эрика Фримена (издательство «Эксмо»).

HTML: структура языка

В языке гипертекстовой разметки HTML используются простые команды, именуемые тегами, для определения различных частей — фрагментов. Ниже приведен HTML-код простой веб-страницы:

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Это заголовок веб-страницы</title>
  </head>
  <body>
    <p>А это абзац этой веб-страницы</p>
  </body>
</html>
```

Конечно, пример очень простой, но демонстрирует все основные элементы, необходимые обычной веб-странице. В нем вы заметите то, что называется объявлением типа документа — doctype, за ним следует *открывающий* тег <html> (со скобками), потом элемент head (голова, *раздел заголовка*), следом body (*тело*, раздел тела), а в нем непосредственно содержимое веб-страницы. Все это завершается *закрывающим* тегом </html>. Открывающий и закрывающий теги образуют *HTML-элемент*.

Типы документов

Все веб-страницы начинаются с объявления типа документа — строки кода, определяющей разновидность HTML, которой вы пользовались при написании страницы. В течение многих лет использовались два типа документов — HTML 4.01 и XHTML 1.0, и каждый из них имеет два стиля: *строгий* и *переходный*. Например, объявление переходного типа документа HTML 4.01 имеет следующий вид (другие объявления типа документа для HTML 4.01 и XHTML 1.0 выглядят примерно так же):

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

ПРИМЕЧАНИЕ

Примеры всех разновидностей объявлений типа документа можно найти на сайте tinyurl.com/opw4nq.

Если посмотреть на код примера HTML-страницы, показанный в этом разделе, то вы увидите, что в нем используется краткая форма объявления типа документа:

```
<!doctype html>
```

Объявление типа документа появилось в языке HTML5. По сравнению с предшественниками, в HTML5 заложена простота и рациональность использования. В этой книге применяется объявление типа документа из HTML5, поддерживаемое любым популярным браузером (даже старым Internet Explorer 6). Применять другие объявления, отличные от простого `doctype` из HTML5, не имеет смысла.

ПРИМЕЧАНИЕ

Несмотря на то что объявление типа документа работает в старых браузерах, это не значит, что они поддерживают все элементы или особенности HTML5. Например, Internet Explorer 8 или более ранние версии новые HTML5-элементы не распознают. Чтобы в этих версиях внести в элементы стиль с помощью CSS, нужно будет воспользоваться кодом JavaScript. Как инструктировать старые браузеры на поддержку современных веб-страниц, будет показано далее.

Независимо от предпочитаемого типа документа, объявляемого с помощью `doctype`, важно, чтобы использовалось объявление хотя бы одного из них. Без этого ваши страницы будут выглядеть по-разному в зависимости от браузера вашего посетителя, поскольку браузеры, не имеющие в качестве руководства объявления типа документа, по-разному отображают информацию, форматированную с помощью CSS.

Каждое объявление типа документа требует от вас написания HTML-кода определенным образом. Например, элемент для разбиения строк имеет в HTML 4.01 следующий вид:

```
<br>
```

Но в XHTML у этого элемента такой вид:

```
<br />
```

И здесь проявляется еще одно преимущество HTML5: он допускает применение любого из этих вариантов.

Как работают HTML-элементы

HTML-элементы состоят из тегов. В приведенном выше примере, как и в HTML-коде любой веб-страницы, большинство элементов используют пары тегов, начиная и завершая какой-то фрагмент — блок текста или другие команды. Будучи заключенными в скобки, эти теги представляют собой команды, которые говорят браузеру, каким образом отображать веб-страницу. Теги являются «разметочной» (markup) частью гипертекстового языка разметки — Hypertext Markup Language.

Открывающий тег каждого элемента показывает браузеру, где команда начинается, а заканчивающий — где заканчивается. Закрывающий тег всегда предворяется прямым слешем (/) после первого символа скобки (<).

На любой веб-странице обычно имеются как минимум следующие четыре HTML-элемента.

- Самая первая строка примера содержит **объявление типа документа** — элемент `doctype`, рассмотренный в предыдущем разделе.

- Тег `<html>` требуется в начале веб-страницы и (с добавленным слешем) в конце: `</html>`. Элемент `html` сообщает браузеру, что документ является программным кодом на языке HTML. Все содержимое страницы, включая остальные элементы, находится между открывающим и закрывающим тегами элемента `html`.

Если представить веб-страницу в виде дерева, то элемент `html` будет его стволом. Две основные части любой веб-страницы — *раздел заголовка* и *тело* — представляют собой ветви.

- Раздел заголовка веб-страницы (`head`) содержит ее название. Здесь также может содержаться другая информация, не отображаемая при просмотре веб-страницы, например описание страницы, которая предназначена для браузеров и поисковых машин. Раздел заголовка заключается в открывающий и закрывающий теги элемента `head`.

Кроме того, раздел заголовка может содержать информацию, используемую браузером для оформления HTML, имеющегося на странице, и для придания странице интерактивности. Вы увидите, что раздел заголовка может содержать код CSS (вроде того, который вы будете учиться создавать) или ссылку на другой файл, содержащий таблицу стилей.

- *Тело* веб-страницы, следующее непосредственно за разделом заголовка и заключенное в теги элемента `body`, содержит все, что должно появиться в окне браузера: заголовки, текст, изображения и т. д.

Внутри тела страницы, как правило, можно найти следующие элементы:

- элемент `p` — открывающий тег `<p>` начинает абзац, а закрывающий `</p>` завершает;
- элемент `strong` — выделяет текст как важный; например, код `Внимание!` сообщит браузеру о том, что слово «Внимание!» должно быть выделено;
- элемент `a`, или элемент привязки, — создает гиперссылку, при щелчке на которой можно переместиться в другую позицию веб-страницы или на другую страницу (нужно указать браузеру эту ссылку путем размещения ее внутри открывающего тега `<a>`, например `Щелкните здесь!`).

Браузер знает, что при щелчке кнопкой мыши на ссылке со словами *Щелкните здесь!* посетитель вашей страницы должен перейти на сайт с адресом `http://www.piter.com`. Часть тега `a` — слово `href` — называют атрибутом, а URL (*унифицированный указатель ресурса*, или URL-адрес) является его значением. В этом примере `http://www.piter.com` — значение атрибута `href`.

HTML5: больше элементов

HTML5, актуальная версия языка HTML, существует уже несколько лет. Иногда вы будете слышать названия, не относящиеся к HTML-элементам, например локальные хранилища (способ хранения данных с сайта на компьютере посетителя), геолокация (способ определения координат компьютера посетителя) и рисование на веб-странице с помощью библиотеки WebGL. Строго говоря, эти технологии не

являются частью HTML, но они относятся к новым возможностям браузера, появившимся вместе с HTML5.

В этой книге термин *HTML5* всегда относится к типу документа HTML5, а также к новым элементам, являющимся частью нового стандарта HTML5. HTML5 не несет каких-либо кардинальных отличий от своих предшественников — его целью является забота о том, чтобы Всемирная паутина продолжала работать так же, как и прежде, поддерживая новые потребности дизайнеров. В версии HTML5 были добавлены новые элементы. Например, элемент `header` может включать в себя содержимое, которое обычно встречается в верхней части страницы, — логотип и общие для всего сайта навигационные ссылки. Новый элемент `nav` заключает в себе набор ссылок, использующихся для навигации по сайту, а элемент `footer` размещает в себе все, что обычно помещается в нижней части страницы, например юридическую информацию, контакты по электронной почте и т. д.

Кроме того, в HTML5 добавлены новые элементы, позволяющие внедрять на страницу видео- и аудиоконтент, и новые элементы формы, добавляющие такие сложные компоненты, как ползунковые регуляторы, всплывающие панели выбора даты, а также встроенную браузерную поддержку проверки допустимости данных, введенных в форму (которая гарантирует правильное заполнение ваших форм посетителями). На протяжении всей книги, особенно в следующей главе, вы будете встречать примеры использования языка HTML5.

Программное обеспечение, используемое для CSS

Чтобы создавать веб-страницы на языках HTML и CSS, вполне достаточно обычного текстового редактора, такого как Блокнот (Notepad) в операционной системе Windows или Text Edit в OS X. После верстки нескольких сотен строк кода HTML или CSS вы, наверное, захотите пользоваться программой, более подходящей для работы с веб-страницами. В этом разделе перечислены некоторые из них. Одни бесплатные, другие придется приобрести.

ПРИМЕЧАНИЕ

Существуют сотни программ, которые могут помочь вам в создании веб-страниц, поэтому здесь приводится неполный список. Все же это самые популярные программы, которыми пользуются любители CSS на сегодняшний день.

Бесплатное программное обеспечение

На данный момент создано много бесплатных программ для редактирования веб-страниц и таблиц стилей. Если вы все еще пользуетесь обычным текстовым редактором, то имеет смысл попробовать одну из нижеприведенных программ.

- **Brackets** (Windows, OS X, Linux; brackets.io). Бесплатный текстовый редактор с открытым кодом, созданный под руководством компании Adobe, содержит множество инструментов для работы с языком HTML и каскадными таблицами стилей. Он будет особенно полезен веб-дизайнерам и разработчикам.

- **Atom** (Windows, OS X, Linux; atom.io). Еще один бесплатный текстовый редактор с открытым кодом, созданный разработчиками из компании GitHub, чрезвычайно популярный благодаря функциям обмена кодом и совместной разработки сайта. Как и программа Brackets, этот новый текстовый редактор в первую очередь предназначен для веб-разработчиков.
- **jEdit** (Windows, OS X, Linux; jedit.org). Бесплатный текстовый редактор, использующий Java, работает практически на всех компьютерах. В нем вы найдете большинство тех функций, которые доступны в коммерческих программах, включая подсветку синтаксиса для CSS.
- **Notepad++** (Windows; notepad-plus.sourceforge.net). Множество людей просто преклоняются перед этим текстовым редактором. Естественно, в нем есть встроенная функциональность, которая идеально подходит для написания HTML- и CSS-кода, включая подсветку синтаксиса, — HTML-элементы и другие ключевые слова имеют собственные цвета, что значительно облегчает их поиск среди других элементов HTML и CSS.

Платное программное обеспечение

Существует множество коммерческих программ для создания сайтов: от недорогих текстовых редакторов до мощных комплексов для верстки кода.

- **EditPlus** (Windows, editplus.com) — относительно недорогой (\$35) текстовый редактор, который поддерживает подсветку синтаксиса, FTP, автозавершение ввода и другие функции.
- **skEdit** (OS X, skedit.com) — редактор веб-страниц (\$30), полная поддержка FTP/SFTP, подсказка команд и другие полезные функции.
- **Coda2** (OS X; panic.com/coda) — многофункциональное средство для создания веб-страниц стоимостью \$99. Включает в себя текстовый редактор, средство предварительного просмотра страниц, FTP- и SFTP-клиент и графический интерфейс для создания CSS-стилей.
- **Sublime Text** (OS X, Windows, Linux; sublimetext.com) — мощный текстовый редактор (\$70), созданный веб-разработчиками. Обычно используется в компаниях, занимающихся веб-дизайном.
- **Dreamweaver** (OS X и Windows, adobe.com/ru/products/dreamweaver.html) — визуальный редактор веб-страниц (подписка на месяц стоит от \$19,99). Он позволяет видеть, как ваша страница выглядит в браузере. Программа также включает мощный текстовый редактор и превосходные инструменты создания кода CSS. Для эффективного использования этого приложения см. документацию по программе или печатные издания (например, книгу «Adobe Dreamweaver CC. Официальный учебный курс», Эксмо, 2014).

ПРИМЕЧАНИЕ

Здесь речь идет о программах, которые позволяют редактировать код, написанный на языках HTML и CSS. Для создания веб-страниц вам достаточно изучить всего одну из них.

Об этой книге

Всемирная паутина — действительно очень удобное изобретение. К сожалению, правила, по которым *работает* Всемирная сеть, не так просты для понимания. Программисты и технические специалисты, которые пишут официальную документацию, поясняющую основные понятия ее функционирования, не ориентируются на среднестатистического пользователя. Зайдите на сайт tinyurl.com/ncyzfj8, чтобы осознать, может ли это быть понятно обычному человеку.

Люди, приступающие к изучению CSS, как правило, не знают, с чего начать. А тонкости, имеющиеся в CSS, могут сбить с толку даже маститых веб-разработчиков. Цель этой книги — служить руководством для обучения. В ней вы найдете пошаговые инструкции для создания красивых веб-страниц с использованием языка CSS.

Эта книга написана так, чтобы помочь читателям любого уровня. Для извлечения максимальной пользы из материала вы обязательно должны учиться на приведенных примерах HTML и CSS. Если же вы никогда раньше не создавали веб-страницы, то обратитесь к практикуму в конце главы 2. Материал, содержащийся в этих главах, написан для тех, кто уже немного освоился в данной области и имеет средний уровень знаний. Если же вы плохо знаете принципы создания веб-страниц, то для лучшего понимания освещаемой темы должны ознакомиться с текстом врезок «В курс дела!». С другой стороны, если у вас имеется большой опыт создания веб-страниц, обратите внимание на врезки «Для опытных пользователей». Они содержат подсказки, приемы и методы для опытных программистов.

Основные разделы книги. Книга разделена на пять частей. Первые четыре части содержат по несколько глав, а последняя часть состоит из приложений.

- **Часть I. Основы CSS.** Здесь описано создание каскадных таблиц стилей в целом и дан краткий обзор ключевых понятий, таких как *наследование*, *селекторы* и *каскадность* таблиц стилей. Попутно с изучением CSS вы получите основные навыки написания HTML-кода. Практикумы закрепят вводимые в главах основные понятия и позволят вам почувствовать эффективность использования CSS.
- **Часть II. Применение CSS.** Перенесет вас в реальный мир веб-дизайна. Вы изучите наиболее важные свойства CSS и их использование для форматирования текста, попрактикуетесь в создании полезных инструментов навигации и сможете улучшить внешний вид своих экспериментальных веб-страниц, добавив графику. Вы также узнаете о том, как с помощью CSS создавать простую анимацию. Эта часть также содержит рекомендации о том, как создавать красивые таблицы и формы.
- **Часть III. Верстка страниц с помощью CSS.** Поможет вам разобраться с самой запутанной, но очень полезной функцией CSS: с управлением размещением элементов на вашей странице. Вы познакомитесь со схемами дизайна (размещение контента в две и три колонки) и узнаете, как добавить боковые панели. Будет рассказано о двух основных методах позиционирования элементов на странице:

абсолютном и относительном. Вы также научитесь создавать сайты, адаптируемые для лучшего восприятия в браузерах настольных систем, планшетных компьютеров и мобильных устройств, и применять flexbox — новый мощный инструмент для создания макета веб-страниц.

- **Часть IV. Профессиональные приемы CSS-верстки.** Содержит советы от профессионалов по улучшению ваших каскадных таблиц стилей, а также расскажет о Sass — мощном и эффективном способе верстки таблиц стилей.
- **Часть V. Приложения,** включая два справочника. Справочник свойств CSS описывает каждое свойство в отдельности в простой и доступной форме, чтобы вы могли быстро узнать о полезных свойствах CSS, которые раньше вам могли не попадаться, или быстро освежить в памяти уже знакомые свойства. Во втором приложении приводится описание инструментов и средств для создания и применения каскадных таблиц стилей.

ОСНОВЫ

При чтении книги и выполнении примеров на компьютере вы должны быть знакомы с некоторыми терминами и понятиями.

- **Щелчок кнопкой мыши.** Пользуясь мышью или тачпадом (трекпадом) вашего компьютера, вы можете выполнить три действия. *Щелчок кнопкой* мыши означает, что нужно навести указатель мыши на какой-либо объект на экране монитора, а затем, не перемещая указатель, нажать и отпустить левую кнопку мыши (тачпада). *Щелчок правой кнопкой* мыши означает соответственно быстрое нажатие правой кнопки мыши, опять-таки не перемещая указатель. *Двойной щелчок* кнопкой мыши означает соответственно быстрое нажатие левой кнопки мыши дважды, опять-таки не перемещая указатель. *Перетаскивание* мышью означает перемещение указателя при нажатой и удерживаемой кнопке мыши.

Если говорится, что нужно щелкнуть кнопкой мыши, нажав клавишу ⌘ (OS X) или Ctrl (Windows), то следует щелкнуть левой кнопкой мыши, предварительно нажав клавишу ⌘ или Ctrl.

- **Меню.** Это строка с кнопками, находящаяся сверху вашего экрана или окна: **Файл (File), Редактирование (Edit)** и т. д. Чтобы появился список команд, соответствующих каждому конкретному пункту меню, нужно просто щелкнуть кнопкой мыши на соответствующем слове (пункте меню), в результате раскроется перечень команд меню. Подразумевается, что вы умеете запускать программы, открывать сайты и скачивать файлы. Вы должны уметь пользоваться меню **Пуск (Start)** в Windows или панелью **Dock** в OS X, а также окном **Панель управления (Control Panel)** в Windows или **Системные настройки (System Preferences)** в OS X.
- **Сочетания клавиш.** Убирая руки с клавиатуры для перемещения указателя мыши, вы теряете время и можете сбиться. Поэтому многие опытные компьютерные

специалисты вместо команд меню везде, где только можно, используют сочетания клавиш. Когда для быстрого вызова той или иной функции предлагается сочетание клавиш, подобное **Ctrl+S** (**⌘+S**) (это сочетание клавиш сохраняет изменения, внесенные в текущий документ), это говорит о том, что вам нужно удерживать в нажатом состоянии клавишу **Ctrl** или клавишу **⌘** и в это время нажать клавишу **S**, а затем отпустить обе клавиши.

Об ▶ этих ▶ стрелках

В этой книге вам будет попадаться текст такого рода: «Откройте папку Система ▶ Библиотека ▶ Шрифты (System ▶ Library ▶ Fonts)». Это сокращение более пространной инструкции, которая предписывает открыть три последовательно вложенные друг в друга папки и должна звучать следующим образом: «На жестком диске найдите папку под названием Система (System). Откройте ее. В окне системной папки есть папка Библиотека (Library); дважды щелкните на ней, чтобы открыть. В этой папке находится папка Шрифты (Fonts). Дважды щелкните на ней и также откройте».

Точно такие же сокращенные записи команд со стрелками помогут вам открыть нужный пункт меню, как показано на рис. 0.1.

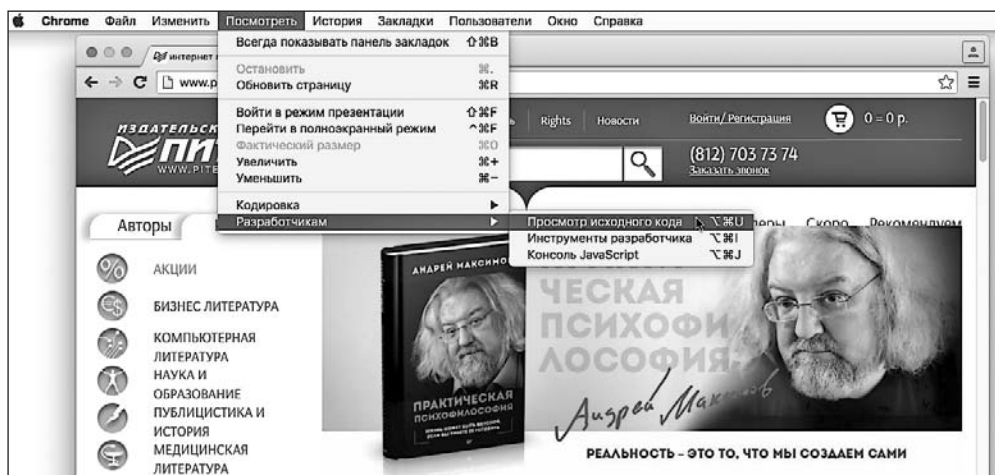


Рис. 0.1. Запись со стрелками позволяет упростить инструкции по использованию меню

Например, запись **Посмотреть ▶ Разработчикам ▶ Просмотр исходного кода** (**View ▶ Developer ▶ View Source**) представляет собой краткий способ дать указание, выполняемое на рис. 0.1: «В пункте меню **Посмотреть** (**View**) выберите подпункт **Разработчикам** (**Developer**), а после этого выберите подпункт **Просмотр исходного кода** (**View Source**)».

Соглашения, использованные в данной книге

В текущем подразделе приводится список соглашений, которые используются в данной книге.

Шрифт для названий

Применяется для отображения URL, а также названий папок и выводимой на экран информации.

Шрифт для команд

Используется для имен файлов, названий путей, имен переменных и команд. Например, путь будет выглядеть так: `/Developer/Applications`.

Шрифт с постоянной шириной

Применяется для отображения примеров исходного кода и содержимого файлов.

Полужирный шрифт с постоянной шириной

Используется для выделения кода, добавленного в старый код.

Вам следует обращать особое внимание на специальные заметки, отделенные от основного текста.

ПРИМЕЧАНИЕ

Это подсказка, пожелание, заметка общего типа. Содержит полезную прикладную информацию по рассматриваемой теме.

ВНИМАНИЕ

Это предостережение или указание, говорящее о том, что вам необходимо быть внимательными. Оно часто указывает на то, что ваши деньги или ваша частная информация могут оказаться под угрозой.

СОВЕТ

Это совет. Советы содержат полезную информацию по рассматриваемой теме, зачастую выделяя важные концепции или лучшие практические решения.

Интернет-ресурсы

Приобретя данное руководство, вы получаете не только книгу для чтения. Во Всемирной паутине можно найти файлы-примеры для получения практического опыта, а также советы, статьи и, может быть, даже несколько видеосюжетов по конкретной теме.

Вы также можете связаться с издательством «Питер» и сообщить о том, что вам понравилось (или не понравилось) в данной книге. Для этого заходите на сайт piter.com.

Примеры к книге

Эта книга создана для того, чтобы работа над вашими веб-проектами велась быстрее и профессиональнее. Поэтому естественно, что половина всего ценного, что есть в этой книге, доступно во Всемирной паутине.

По ходу чтения вам будут встречаться *практикумы* — пошаговые обучающие уроки, которые вы сможете выполнять, используя исходные данные (графические файлы и незаконченные веб-страницы). Их можно загрузить с сайта github.com/mrightman/css_4e. От простого прочтения этих пошаговых уроков толку будет мало. Но если их прорабатывать на компьютере, можно разобраться, каким образом профессиональные дизайнеры создают веб-страницы.

Среди примеров вы также найдете законченные страницы, которые позволят вам сравнить свою работу с требуемым конечным результатом.

ЧАСТЬ I

ОСНОВЫ CSS

Глава 1. HTML и CSS

Глава 2. Создание стилей и таблиц стилей

Глава 3. Селекторы: выбор форматируемых элементов

Глава 4. Механизм наследования стилей

Глава 5. Управление сложной структурой стилей: каскадность

1 HTML и CSS

Каскадные таблицы стилей без языка HTML ничто. Код на языке HTML образует структуру веб-страниц с их содержимым (контентом). Хотя страница, созданная с помощью только языка HTML, не очень красива, без него Всемирной паутины не существовало бы. Поэтому для получения от CSS наибольшей отдачи ваш HTML-код должен предоставить однородную, хорошо скроенную основу. В этой главе вы познакомитесь с основами каскадных таблиц стилей, узнаете, как создавать HTML-код, более приспособленный под нужды CSS.

Важно отметить, что повсеместное использование на сайте CSS существенно облегчает создание HTML-кода. Вам больше не нужно будет применять средства HTML для создания и улучшения дизайна страниц (собственно, HTML для этого никогда и не предназначался). Все, что связано с графическим дизайном страниц, обеспечивается с помощью CSS. Соответственно, работа над основным кодом веб-страниц на языке HTML упрощается, поскольку HTML-страницы, написанные для совместной работы с CSS, имеют менее громоздкий, более понятный и прозрачный код. При этом, естественно, страницы загружаются быстрее, что немало важно для посетителей вашего сайта.

Посмотрите на рис. 1.1. Дизайн обеих страниц одинаков, однако верхняя создана лишь с использованием CSS, а нижняя — только с помощью HTML. Размер HTML-файла верхней страницы всего 4 Кбайт, в то время как его размер для нижней страницы, полностью написанной на HTML, — почти в четыре раза больше (14 Кбайт). Подход к написанию веб-страниц с применением исключительно HTML требует намного большего объема кода, чтобы достичь практически тех же визуальных эффектов: 213 строк HTML-кода по сравнению с 71 строкой для версии с применением CSS.

HTML: прошлое и настоящее

Язык HTML на сегодняшний день составляет основу для написания любой веб-страницы во Всемирной паутине. При использовании CSS написание кода на языке HTML упрощается. Теперь вам не нужно использовать HTML-элементы (такие как элемент `font`) для управления оформлением веб-страниц. Эту работу выполняют каскадные таблицы стилей. Прежде чем мы начнем знакомство с CSS, давайте поговорим о прошлом и настоящем.

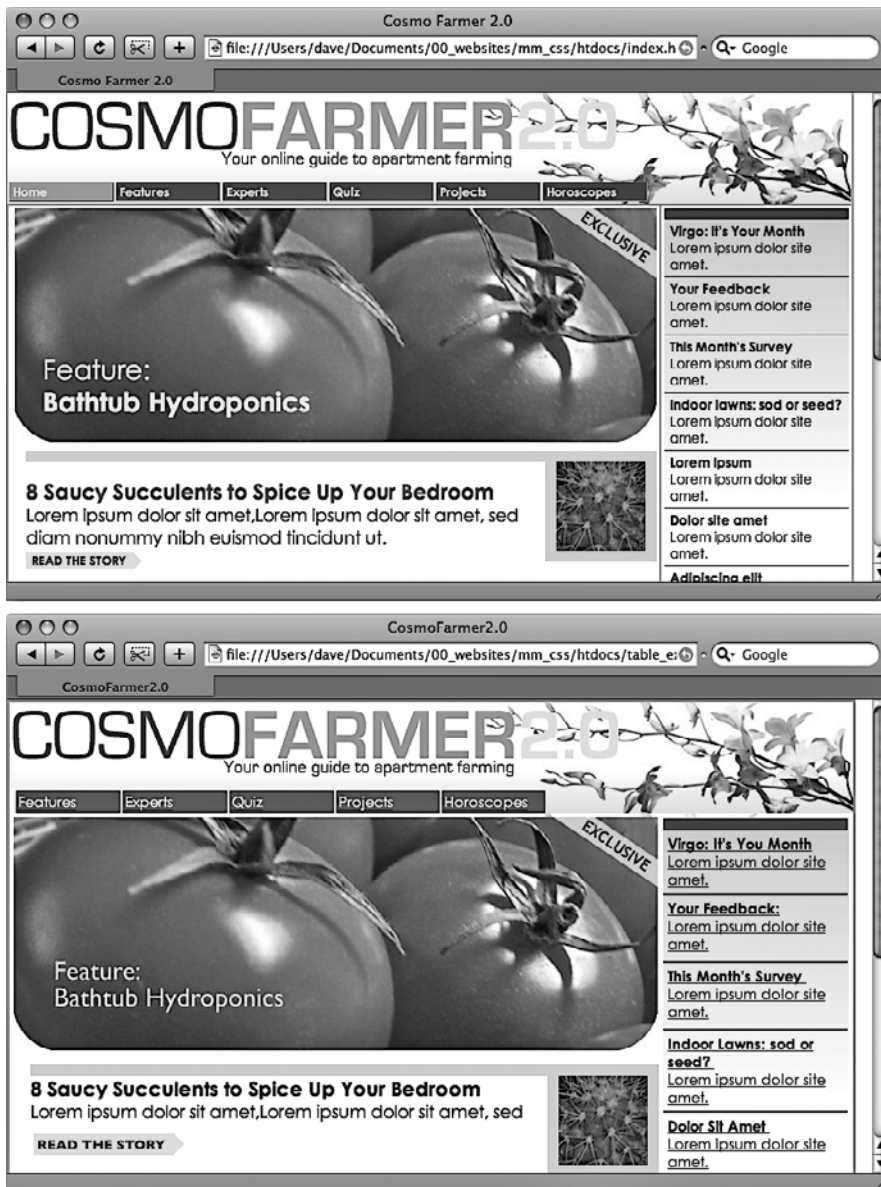


Рис. 1.1. Каскадные таблицы стилей упрощают использование языка HTML

Прошлое HTML: лишь бы все выглядело хорошо

Когда группа ученых создала Всемирную паутину, которая первоначально предназначалась для совместного использования и поиска технической документации, никто к графическим дизайнерам не обращался. Язык HTML нужен был ученым только для структурирования информации с целью ее более простого восприятия.

Возьмем для примера элемент `h1`. Он выделяет в тексте важный заголовок, а элемент `h2` создает подзаголовок с меньшим размером шрифта. Другой широко используемый элемент — `ol` (ordered list — «упорядоченный список») — создает нумерованный список для перечислений.

Когда язык HTML стали применять не только ученые, но и обычные пользователи, они захотели, чтобы их веб-страницы выглядели красиво. С тех пор дизайнеры веб-страниц начали использовать теги для форматирования страниц в дополнение к их основному назначению — структурированию данных. Например, вы можете применить открывающий и закрывающий теги элемента `blockquote` (предназначен для цитирования материала из другого источника) к любому тексту, который нужно выделить небольшим отступом вправо. Вы можете пользоваться тегами элементов заголовков, чтобы выделить любой текст полужирным шрифтом большего размера, независимо от того, заголовок это или нет.

Достаточно сложный способ применения элемента `table` был придуман веб-дизайнерами, чтобы создавать колонки текста, а также точно позиционировать изображения и текст на странице. Поскольку элемент первоначально предназначался для отображения таких таблиц, как результаты исследовательских данных, расписания поездов и т. д., проектировщикам пришлось упражняться в применении элемента `table` самыми необычными способами, создавая неоднократно вложенные таблицы, чтобы содержимое веб-страниц смотрелось красиво.

Тем временем производители браузеров также прилагали усилия в совершенствовании языка разметки, вводя новые HTML-элементы и атрибуты для улучшения дизайна веб-страниц. Например, элемент `font` позволяет определять цвет, начертания и один из семи кеглей шрифта (это приблизительно в 100 раз меньше типоразмеров шрифта, чем предлагает, скажем, редактор Microsoft Word).

Наконец, когда проектировщики не могли добиться нужных результатов, они часто применяли графику. Например, использовали очень большой рисунок в качестве фона для веб-страницы или *нарезали* его на маленькие графические файлы и собирали их воедино в таблицах, чтобы воссоздать оригинальное изображение.

По мере использования всех этих премудростей и применения атрибутов элементов, широкого употребления изображений и т. д. для изменения дизайна страниц HTML-код разрастался до неузнаваемости. Но сложный код замедляет визуализацию страницы в браузере, а также снижает скорость ее загрузки.

Настоящее HTML: подготовка рабочего материала для CSS

Независимо от того, что представляет собой веб-страница — календарь промышленного сезона, схему проезда к ближайшему супермаркету или фотоальбом прошлого дня рождения вашего ребенка, — именно ее дизайн создает имидж, заставляя сайт выглядеть профессионально. Хороший дизайн страниц сайта помогает донести послание его посетителям, которые должны легко найти там именно то, что ищут.

Таким образом, чтобы веб-страницы на языке HTML выглядели привлекательно, дизайнерам приходилось усиленно трудиться. Дизайн с помощью CSS позволяет языку HTML вернуться к исполнению своей прямой обязанности — созданию

структуры документа. Использование HTML для дизайна веб-страниц на сегодняшний день является признаком дурного тона. Поэтому не волнуйтесь по поводу того, что у элемента `h1` слишком большой размер шрифта или отступы упорядоченного списка очень велики. Вы сможете изменить их с помощью каскадных таблиц стилей. Во время написания кода думайте о HTML как о средстве структурирования. Используйте его, чтобы упорядочить содержимое страницы, а CSS — чтобы сделать это содержимое привлекательным.

Верстка HTML-кода вместе с CSS

Для тех, кто не очень хорошо знаком с веб-дизайном, возможно, будут полезны некоторые подсказки по языку HTML. Если раньше вы уже создавали веб-страницы, то сможете избавиться от стиля написания HTML-кода, который лучше быстрее забыть. Сейчас речь пойдет о способе написания HTML-кода для его совместного использования с кодом CSS.

Думайте о структуре

HTML придает особый вид тексту путем деления его на логические блоки и их определения на веб-странице: например, главное значение элемента `h1` — создать заголовок, предшествующий основному содержимому страницы. Заголовки второго, третьего уровней и т. д. — подзаголовки — позволяют делить содержимое страниц на менее важные, но связанные разделы. У веб-страницы, как и у книги, которую вы держите в руках, должна быть логическая структура. У каждой главы этой книги есть заголовок (отформатированный, например, элементом `h1`), а также несколько разделов и, соответственно, подзаголовков (например, с элементом `h2`), которые, в свою очередь, содержат подразделы с заголовками более низкого уровня. Представьте, насколько сложнее было бы читать эту книгу, если бы весь текст состоял из одного длинного абзаца, без деления на разделы, подразделы, пункты, без выделения примечаний, гиперссылок и т. д.

ПРИМЕЧАНИЕ

Руководство по языку HTML можно найти на сайте tinyurl.com/4fmwq8. Краткий список всех HTML-элементов есть в справочнике Mozilla Developer по адресу tinyurl.com/p8rex1q.

Помимо заголовков, в HTML есть множество других элементов для разметки содержимого веб-страницы, а также для определения назначения ее каждого логического фрагмента. Наиболее часто применяют следующие элементы: `p` — для создания абзацев текста, `ul` — для создания маркированных (нумерованных) списков. Далее по степени применения идут элементы, отображающие специфичное содержимое, например `abbr` — сокращения, аббревиатуры, `code` — программный код.

При написании HTML-кода для CSS используйте элементы, размещая их рядом друг с другом, насколько это возможно. Ориентируйтесь на роль, которую играет фрагмент текста на веб-странице, а не на внешний вид, который текст приобретает благодаря этому элементу (рис. 1.2). Например, перечень ссылок

на панели навигации — это и не заголовок, и не абзац текста. Больше всего это похоже на маркированный список. Таким образом, выбираем элемент `ul`. Вы можете сказать, что элементы маркированного списка расположены вертикально один за другим, а нам требуется горизонтальная панель навигации, в которой все ссылки располагаются горизонтально. Об этом можно не беспокоиться. Средствами CSS очень просто преобразовать вертикальный список ссылок в элегантную горизонтальную панель навигации, о чем вы сможете прочитать в главе 9.



The Urban Agrarian Lifestyle
A Revolution in Indoor Agriculture
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.

Рис. 1.2. Пример изменения оформления текста на странице с помощью CSS

Разнообразие HTML-элементов

Пестрого ассортимента HTML-элементов не всегда достаточно для оформления широкого диапазона контента, который вы хотели бы опубликовать на своей веб-странице. Конечно, элемент `code` прекрасно подходит для разметки программного кода, но большинство людей считают элемент `pre` удобнее. К счастью, HTML обеспечивает несколько «структурных» элементов, которые позволяют лучше идентифицировать и группировать контент и в процессе работы обеспечивают «маркеры», помогающие назначить те или иные стили CSS определенным элементам на странице. К их числу относятся элементы `div` и `span`, существующие практически с момента появления языка HTML. Язык HTML5 предоставил широкий спектр элементов, которые позволяют группировать контент. Например, элемент `footer` можно использовать для отображения колонтитулов — такой информации, как сведения об авторских правах, контактных данных, или списка источников.

До появления CSS для достижения определенных визуальных эффектов дизайнеры пользовались элементом `font` и другими средствами HTML:

```
<p>  
  <strong>  
    <font color="#0066FF" size="5" face="Verdana,  
      Arial, Helvetica, sans-serif">Urban Agrarian  
      Lifestyle</font></strong>  
  <br />  
  <font color="#FF3300" size="4" face="Georgia,  
    Times New Roman, Times, serif">  
  <em>  
  <strong>A Revolution in Indoor Agriculture  
  <br /></strong></em></font>  
  Lorem ipsum dolor sit amet...  
</p>
```

Используя CSS, можно добиться тех же результатов (и даже лучше) с гораздо меньшим объемом HTML-кода (см. рис. 1.2):

```
<h1>The Urban Agrarian Lifestyle</h1>
<h2>A Revolution in Indoor Agriculture</h2>
<p>Lorem ipsum dolor sit amet...</p>
```

Кроме того, применяя CSS для дизайна веб-страницы, вы используете HTML по его прямому назначению, то есть именно для разметки кода веб-страницы на логические фрагменты, не заботясь о форматировании и внешнем виде страницы.

Использование элементов `div` и `span`

Элементы `div` и `span` применяются на протяжении всего существования Всемирной паутины. Обычно они используются для организации и группирования контента, с чем не всегда справляются другие HTML-элементы. Они похожи на пустые сосуды, которые вы сами и заполняете. Элемент `div` (предназначен для деления на фрагменты) определяет любой отдельный блок содержимого, как, например, абзац или заголовок, поэтому называется *блочным*. Поскольку у элементов `div` и `span` нет никаких свойств для визуализации, вы можете применять к ним CSS-стили, чтобы фрагменты внутри этих тегов выглядели так, как вам хочется. Однако вы также можете логически объединить любой набор таких элементов, как заголовок, несколько абзацев, маркированный список и т. д., в единственном блоке `div`.

Элемент `div` — замечательное средство разбивки веб-страницы на такие логические фрагменты, как баннер, колонтитул, боковая панель и т. д. Впоследствии, используя CSS, вы сможете позиционировать любой из этих фрагментов в выбранное место веб-страницы, создавая сложную схему разметки (см. часть III).

В КУРС ДЕЛА!

Простой HTML — помощь поисковым системам

Научившись писать HTML-код так, чтобы использовать его исключительно для структурирования содержимого документов, и применяя CSS как инструмент для дизайна, форматирования, придания страницам законченного внешнего вида, вы обнаружите дополнительные преимущества простого и понятного HTML. С одной стороны, вы сможете поднять позиции своих веб-страниц в поисковых системах, заняв верхние строки в списках таких поисковых систем, как Google, Yahoo! и Bing. Когда поисковые системы сканируют просторы Всемирной паутины в поисках новых сайтов и новой информации, они систематизируют данные, просматривая *весь* HTML-код веб-страниц на предмет актуального содержимого. Старый способ написания HTML-кода с применением элементов форматирования (например, `font`) и множества таблиц для верстки дизайна страницы влияет на работу поискового сервера. Некоторые поисковые системы прекращают чтение HTML-кода страницы по-

сле просмотра определенного количества символов. Если у вас на страницах имеется большой объем HTML-кода для создания дизайна, то поисковый сервер может упустить важное содержимое страниц или отдельные страницы вообще не будут систематизированы.

Простой и структурированный HTML, напротив, будет легко просматриваться и систематизироваться поисковыми системами. Использование элемента `h1` для выделения важных тем страниц (в противоположность форматированию текста большим полужирным шрифтом) — грамотный подход, поскольку поисковые серверы придают большое значение содержимому этого элемента во время сканирования страниц.

Рекомендации Google по созданию веб-страниц для правильного восприятия их поисковыми системами смотрите по адресу tinyurl.com/olx2lyk.

Элемент `span` применим к *строчным* элементам страницы, то есть к словам, фразам, которые находятся в пределах абзаца текста или заголовка. Его можно использовать точно так же, как и другие строчные HTML-теги, к примеру, как `a` (элемент привязки, позволяющий добавить ссылку к фрагменту текста) или `strong` (позволяющий отформатировать фрагмент текста в абзаце полужирным начертанием). Можно применить элемент `span`, например, к названию компании и затем использовать CSS, чтобы выделить это название другим шрифтом, цветом и т. д.

Рассмотрим пример использования этих элементов в работе. К ним добавлены атрибуты `id` и `class`, предназначенные в том числе для применения стилей к фрагментам страницы.

```
<div id="footer">
  <p>Собственность 2015. <span class="bizName">SuperCo.com</span></p>
  <p>Звоните в службу поддержки потребителей по номеру 555-555-5501
    для подробной информации.</p>
</div>
```

Разговор об этих элементах не ограничится кратким введением. Они часто встречаются на тех веб-страницах, где широко применяется CSS, и эта книга поможет вам научиться использовать их в комбинации с CSS для получения творческого контроля над веб-страницами.

Элементы в HTML5

Элемент `div` имеет общий характер — это блочный элемент, используемый для разбиения страницы на разделы. Одна из целей HTML5 — предоставление в распоряжение разработчиков широкого выбора из других, *семантически* более осмысленных элементов. Придание HTML-коду большей семантики просто означает использование элементов, в точности описывающих свое содержимое. Как уже упоминалось в текущем разделе, вы можете воспользоваться элементом `h1` (заголовок первого уровня), помещая в него текст, описывающий основное содержимое страницы. По аналогии с этим элемент `code` четко дает понять, информация какого сорта в него помещена (программный код).

HTML5 включает в себя множество различных элементов, имена которых отражают тип их содержимого. Они могут использоваться вместо элемента `div`. Элемент `article` (статья), к примеру, применяется для обозначения раздела страницы, содержащего завершенную, независимую публикацию, статью: запись блога, описание товара в интернет-магазине или просто основной текст страницы. Точно так же элемент `header` (верхний колонтитул) является признаком *верхнего колонтитула*, или *баннера* в верхней части страницы, который обычно содержит логотип, навигацию, относящуюся ко всему сайту, название страницы с рекламным слоганом и т. д.

ПРИМЕЧАНИЕ

Дополнительные сведения об HTML-элементах можно найти на сайтах tinyurl.com/d4gsdrr и tinyurl.com/pc3ddbс, а также в книге Дженнифер Роббинс «HTML5. Карманный справочник», Вильямс, 2015.

Многие HTML5-элементы предназначены для расширения возможностей обычного элемента `div`. Для структурирования содержимого страницы часто используются и другие HTML5-элементы.

- `section` (раздел) — группирует взаимосвязанное содержимое, например главу книги. К примеру, вы можете разбить содержимое главной страницы на три раздела: вводную информацию о сайте, контактную информацию и самые свежие новости.
- `aside` (отступление) — предназначен для обозначения содержимого, связанного с неким контентом. Например, пометки на полях в печатном журнале.
- `footer` (нижний колонтитул) — содержит информацию, которая обычно помещается в нижнем колонтитуле страницы, например сведения об авторских правах, другая правовая информация, ссылки для навигации по сайту и т. д. Но на количество элементов `footer` на одной странице ограничений не накладывается, вы можете, скажем, поместить нижний колонтитул внутри элемента `article`, чтобы хранить в нем информацию, относящуюся к публикации, например сноски, ссылки или выписки.
- `nav` (навигация) — используется для обозначения содержимого в виде основных навигационных ссылок.
- `figure` (рисунок) — применяется для иллюстраций. Вы можете поместить в него элемент `img`, а также `figcaption`, предназначенный для отображения подрисуночной подписи — пояснения к фотографии или иллюстрации, находящейся внутри элемента `figure`.

СОВЕТ

Разобраться в том, какой из HTML5-элементов лучше использовать, то есть должен ли ваш текст быть статьей — `article` или разделом — `section`, бывает порой непросто. Удобная блок-схема, которая поможет разобраться в предназначении новых элементов разбиения информации, имеющих в HTML5, представлена по адресу tinyurl.com/o298cs6.

Существуют и другие HTML5-элементы, и многие из них просто предоставляют более описательную альтернативу элементу `div`. В этой книге используются как элементы `div`, так и новые HTML5-элементы, придающие веб-странице более выраженную организацию содержимого. Недостаток HTML5 заключается в том, что браузер Internet Explorer 8 и его более ранние версии не распознают новые элементы без посторонней помощи (см. врезку «Обходной прием» далее в этой главе).

Следует также заметить, что, кроме чувства сопричастности к самым последним течениям в веб-дизайне, от использования некоторых из этих HTML5-элементов на самом деле нет никакой ощутимой пользы. Например, использование элемента `article` просто для того, чтобы содержать в нем сводку новостей веб-страницы, не улучшит ее внешний вид. Если хотите, можете и впредь применять элемент `div`, избегая элементов разбиения страницы на разделы, предлагаемых HTML5.

Кроме того, даже используя HTML5-элементы, вы все же иногда будете вынуждены прибегать к помощи `div` для простой группировки других HTML-элементов. Вы будете делать это в тех случаях, когда необходимо переместить группу элементов в другое место на странице, чтобы назначить им последовательный фоновый цвет или нарисовать контур и добавить тень.

Представление о макете страницы

Когда для обозначения основной темы страницы используется элемент `h1`, а для добавления текстового абзаца — элемент `p`, вы в конечном счете хотите увидеть привлекательную страницу. Когда после прочтения части III книги вы научитесь использовать CSS для компоновки макета страницы, вам уже не нужно будет следить за дизайном еще на этапе написания HTML-кода.

Макет страницы можно представить себе в виде искусно расположенных прямоугольных элементов (пример такого расположения показан на рис. 1.3). В конечном счете дизайн страницы, состоящей из двух вертикальных колонок текста, фактически представляет собой лишь два смежных прямоугольника. Верхний колонтитул — логотип, лозунг, поля поиска и элементов навигации по сайту — представляет собой не что иное, как широкий блок (прямоугольник), занимающий всю верхнюю часть окна браузера. Иными словами, если вы представите себе группировку и макет содержимого страницы, то должны увидеть блоки, установленные друг на друга, следующие друг за другом и находящиеся друг под другом.



Рис. 1.3. Стандартный двухколоночный макет страницы, включающий такие основные структурные блоки, как баннер (в верхней части), колонка основного содержимого (слева по центру), боковая панель (справа по центру) и нижний колонтитул (в нижней части)

ОБХОДНОЙ ПРИЕМ

Как заставить браузер Internet Explorer 8 поддерживать HTML5-элементы

Язык HTML5 предоставляет в ваше распоряжение много новых элементов. От описывающих характер своего содержимого, например элемента `nav`, и до представляющих дополнительные возможности, таких как элемент `video`, который предназначен для вставки видеоконтента, и элемент `audio`, предназначенный для вставки аудиофайла. По мере изучения языка HTML5 вы, возможно, начнете применять эти новые элементы на своих веб-страницах.

К сожалению, Internet Explorer 8 и его более ранние версии не распознают эти новые элементы и не будут реагировать на применяемый к ним код CSS. Действительно, если использовать HTML5 и просматривать веб-страницы в Internet Explorer 8, эта книга не принесет никакой пользы. Но... это не совсем так. Есть один способ включения устаревших версий Internet Explorer в список поддержки, заставляющий их понимать весь CSS-код, применяемый к HTML5-элементам. Нужно просто поместить перед закрывающим тегом `</head>`, находящимся в верхней части кода вашего HTML-файла, следующий фрагмент:

```
<!--[if lt IE 9]> <script src="//html5shiv.
googlecode.com/ svn/trunk/html5.js"></script>
<![endif]->
```

Этот несколько необычный фрагмент кода представляет собой то, что называется условным комментарием Internet Explorer для вставки кода на языке JavaScript, видимого только версиями Internet Explorer (IE), предшествующими 9-й. Иными словами, на этот код реагируют только IE 6, 7 и 8, а все остальные браузеры (включая более новые версии Internet Explorer) его попросту игнорируют. Этот код инструктирует более ранние версии IE загружать небольшой JavaScript-сценарий, позволяющий браузеру распознавать HTML5-элементы и применять к ним код CSS, предназначенный для этих элементов.

Код влияет только на то, как браузер выводит HTML5-элементы на экране или при печати, и не заставляет браузер «понимать» те HTML5-элементы, которые на самом деле совершают какие-либо действия. Например, IE 8 и более ранние версии браузера не поддерживают элемент `video` и не могут проигрывать HTML5-видеоролики (даже с добавленным JavaScript-сценарием).

Если вы не знаете, нужно ли учитывать поддержку Internet Explorer 8 в вашем проекте, прочитайте врезку «ЧаВо» на следующей странице.

В HTML-коде эти блоки, или структурные единицы, создаются с помощью элемента `div` или одного из структурных HTML5-элементов, например `footer`, `header`, `article` и `aside`. Заключите HTML-элементы, к примеру образующие область баннера, в один `div`-контейнер, HTML-код, формирующий колонки текста, — в другой `div`-контейнер и т. д. Если вы уже разбираетесь в HTML5, то можете создать дизайн, показанный на рис. 1.3, с помощью элемента `header` для верхнего баннера, элемента `article` — для основного текста, элемента `aside` или `section` — для боковой панели и элемента `footer` — для нижнего колонтитула страницы. Иными словами, если вы собираетесь сгруппировать HTML-элементы в какой-либо позиции страницы, их нужно заключить в такие элементы разбиения на разделы, как `div`, `article`, `section` или `aside`.

Как вы узнаете из части III, каскадные таблицы стилей содержат мощные инструменты для работы с макетом страницы. Вы можете буквально поместить HTML-код в любой позиции окна браузера. Новые объекты, например `flexbox` (см. главу 17), дают вам свободу при написании HTML-кода. Однако я все же рекомендую группировать связанный контент в элементах-контейнерах (`div`) или структурном элементе HTML5.

ЧАВО

Стоит ли волноваться насчет Internet Explorer 8?

Я предполагаю, что Internet Explorer 6 уже отслужил свое и о нем больше не стоит беспокоиться. Но так ли это? А как насчет других версий Internet Explorer?

Если вы веб-дизайнер, то, наверное, на вашем компьютере установлены самые последние версии браузеров Internet Explorer, Firefox, Safari, Chrome или Opera. Однако все зависит от вашей аудитории. Возможно, им следует обновить браузер или их компьютеры настолько стары, что не в состоянии работать с новыми версиями программного обеспечения.

К счастью, браузеры Internet Explorer 6 и 7 стремительно уходят в историю, хотя все еще используются в таких странах, как Китай, Индия и Венесуэла (tinyurl.com/q8htudu). Возможно, указанные выше версии установлены на экспонатах в музеях компьютерной истории.

А вот с Internet Explorer 8 по-прежнему нужно считаться. Он не очень популярен, однако, в зависимости

от источников исследования, с ним еще работают от 2 до 19 % всех пользователей в мире. Данные об использовании браузеров можно найти на сайтах NetMarketShare (tinyurl.com/8nhqrh3) и GlobalStats StatCounter (gs.statcounter.com).

Но даже статистика, включающая географический регион, в котором находится аудитория вашего сайта, не дает достоверной картины относительно пользователей, предпочитающих этот браузер. Если вы создаете сайт, нацеленный на прогрессивных веб-дизайнеров, вполне вероятно, что он практически никогда не будет просматриваться в программе Internet Explorer 8. Но если ваш сайт предназначен для жителей отдаленных регионов России, то вам, возможно, придется иметь дело с IE 8 (а возможно, даже с IE 6 и 7). Лучше всего определить ту часть трафика, которая приходится на запросы от различных браузеров, путем просмотра журналов веб-серверов или оформить подписку на сервисе Google Analytics (google.com/analytics/), чтобы можно было отслеживать браузеры ваших посетителей (а также многое другое).

Корпорация Microsoft объявила, что прекратит поддержку браузера Internet Explorer 8 в январе 2016 года. В это время пользователи операционной системы Windows должны будут обновиться до более поздних версий программы или перейти к другому браузеру, например Chrome или Firefox. Основная проблема браузера Internet Explorer 8 заключается в том, что он не может работать с HTML5-элементами, а это значит, что вам необходимо формировать их непосредственно с помощью каскадных таблиц стилей. Если вы действительно хотите, чтобы ваш сайт правильно отображался в браузере IE 8, то для определения структуры страниц вместо HTML5-элементов, описанных в подразделе «Использование элементов `div` и `span`» выше, используйте элемент `div` или JavaScript-сценарий, описанный во врезке «Как заставить браузер Internet Explorer 8 поддерживать HTML5-элементы».

О каких HTML-элементах рекомендуется забыть

Каскадные таблицы стилей позволяют использовать более простой и понятный HTML-код. Вам больше не нужно работать с устаревшими элементами и атрибутами языка HTML для создания и улучшения дизайна веб-страниц. Элемент `font` — наглядный тому пример. Единственная его цель — изменить цвет, размер и начертание шрифта текста страницы. Он не выполняет никакого структурирования и не делает страницу логически более понятной.

Далее приведен список элементов и атрибутов языка HTML, которые вы можете легко, без ущерба для внешнего вида страниц заменить CSS-стилями.

- **Избавьтесь от элемента `font` для управления форматированием текста.** Каскадные таблицы стилей выполняют это гораздо лучше (о форматировании текста читайте в главе 6).
- **Не используйте элементы `b` и `i` для изменения начертания шрифта.** Если вы хотите действительно выделить текст, пользуйтесь элементом `strong` (обычно браузеры отображают текст, выделенный этим элементом, с полужирным начертанием). Если вы хотите придать тексту чуть меньший акцент, то пользуйтесь для его выделения элементом `em` (браузеры выделяют содержимое этого элемента курсивом). Кроме того, с помощью каскадных таблиц стилей можно выделить текст курсивом, сделать его полужирным или и то и другое одновременно.

В то время как в версии HTML 4 пытались отказаться от использования элементов `b` и `i`, HTML5 вернул их к жизни. В языке HTML5 элемент `b` позволяет сменить начертание шрифта текста на полужирное без семантического выделения. Аналогично элемент `i` используется для смены начертания шрифта текста на курсивное, без подчеркивания его значимости.

СОВЕТ

Чтобы выделить название публикации курсивом, пользуйтесь элементом `cite` — таким образом вы убьете сразу двух зайцев. Он одновременно выделяет название курсивом и помечает его как цитату для поисковых систем. Конечно, каскадные таблицы стилей позволяют делать с элементами все что угодно, поэтому, если вы хотите сослаться на публикацию, но не выделять ее курсивом, можете также воспользоваться элементом `cite`.

- **Не пользуйтесь элементом `table` для компоновки макета страницы.** Применяйте его только с целью отображения табличной информации (например, электронных таблиц, списков, диаграмм). Из части III этой книги вы узнаете, что каскадные таблицы стилей позволяют компоновать макеты веб-страниц гораздо быстрее и с меньшим объемом кода, нежели при использовании элемента `table`.
- **Не злоупотребляйте элементом `
`.** Если вы привыкли пользоваться им для вставки разрывов строк, не создавая новый абзац, то можете попасть в затруднительное положение (иногда браузеры автоматически добавляют промежуток между абзацами, а также между заголовками и абзацами; раньше разработчики шли сложными обходными путями, чтобы избежать появления этого интервала между абзацами, заменяя единственный элемент `p` несколькими элементами разрыва строки и элементом `font`, чтобы первая строка абзаца была *похожа* на заголовок). Использование свойства `margin` в CSS дает свободу определения таких параметров, и вы с легкостью можете установить интервал между абзацами, заголовками и другими блочными элементами (см. раздел «Управление размерами полей и отступов» главы 7).

ПРИМЕЧАНИЕ

Из главы 5 вы узнаете о технике, называемой сбросом стилей. Она устраняет проблему лишних разрывов строк, вставляемых между абзацами и другими элементами.

В целом добавление в элементы атрибутов, управляющих цветом, границами, фоновыми изображениями, выравниванием текста, форматированием таблиц, — устаревший стиль написания HTML-кода. Сюда также входят атрибуты позиционирования изображений, текста в абзацах и ячейках таблицы. Вместо этого обратитесь к средствам CSS, которые обеспечат выравнивание текста (см. раздел «Форматирование абзацев» главы 6), установку границ (см. раздел «Добавление границ» главы 7), фоновые параметры (см. раздел «Установка цвета фона» главы 7) и позиционирование изображений (см. раздел «Добавление фоновых изображений» главы 8).

В КУРС ДЕЛА!

Проверяйте правильность кода веб-страниц

В языке HTML существуют определенные правила: например, элемент `html` охватывает все содержимое веб-страницы, а элемент `title` должен находиться внутри `head`. Если забыть об этих правилах или просто сделать при наборе опечатку, то *некорректный* HTML-код станет причиной некоторых проблем (например, веб-страница в различных браузерах отобразится по-разному). Более того, даже правильный CSS-код может вместе с ошибочным HTML-кодом работать не так, как ожидалось. К счастью, существуют программные средства для проверки правильности — *валидности* — HTML-кода.

Самый легкий способ проверить HTML-код — выполнить синтаксический контроль (валидацию) на сайте W3C по

адресу validator.w3.org (рис. 1.4). Консорциум Всемирной паутины — World-Wide Web Consortium (W3C) — организация, ответственная за определение стандартов веб-технологий и языков программирования, включая HTML и CSS. При нахождении ошибок на ваших веб-страницах W3C-валидатор сообщит о них. Можно либо указать адрес существующей страницы во Всемирной паутине, либо загрузить файл с HTML-кодом на сайт валидатора, либо вставить HTML-код веб-страницы в окно формы на сайте и нажать кнопку запуска проверки.

Расширение Web Developer для браузеров Chrome, Firefox и Opera (tinyurl.com/2353zt) позволит быстро проверить код страницы.

Рекомендации для веб-дизайнеров

Всегда неплохо иметь подробное пошаговое руководство к действию. Если вы все еще не уверены в том, как именно пользоваться языком HTML для создания хорошо структурированных веб-страниц, просмотрите представленные ниже несколько советов для начинающих.

- Используйте заголовки, чтобы указать относительную важность текста. Если два заголовка имеют одинаковую степень важности в теме вашей страницы, то применяйте элемент заголовка одного уровня. Если один из заголовков имеет меньшую значимость, то есть является подтемой другого, применяйте заголовок на уровень ниже. Например, от заголовка `h2` переходите к подзаголовку `h3` (рис. 1.5). Лучше использовать заголовки по порядку и стараться не пропускать их номера, например, никогда не переходите от элемента `h2` сразу к `h5`.
- Используйте элемент `p` для абзацев текста.
- Применяйте маркированные списки (`ul`), если у вас есть перечень связанных элементов, таких как ссылки навигации, содержания, подсказки и др.



Рис. 1.4. HTML-валидатор Консорциума Всемирной паутины (W3C) быстро проверит правильность кода веб-страницы

- Пользуйтесь нумерованными списками (`ol`), чтобы определить ряд последовательно выполняемых операций или порядок набора элементов. В практикумах этой книги (см. раздел «Практикум: форматирование текста» главы 6) есть наглядный пример такого списка.
- Чтобы создать словарь терминов и их определений, пользуйтесь элементом `dl` (список определений терминов) в сочетании с элементами `dt` (название термина) и `dd` (определение термина). Просмотреть пример использования этой конструкции можно по адресу tinyurl.com/ne888za.
- Если вы хотите включить цитату в виде отрывка текста с другого сайта, чье-либо высказывания и др., используйте элемент `blockquote` для длинного контекста (высказываний) и элемент `q` — для вставки краткой цитаты в более длинный абзац, например:

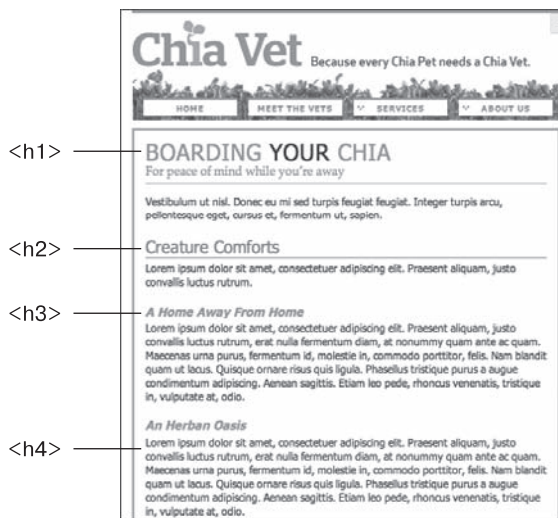


Рис. 1.5. Используйте элементы заголовков (h1, h2 и т. д.) так, как если бы писали доклад: расположите их в тексте в порядке важности. Для самого первого заголовка используйте элемент h1, который должен обозначать «Слушайте! Этот заголовок отражает содержимое всей страницы»

<p>Говорят, что Марк Твен как-то раз написал:

<q>Самой холодной из всех проведенных мною зим было лето в Сан-Франциско</q>.

К сожалению, на самом деле он никогда ничего похожего на эту знаменитую цитату не писал.

</p>.

- Применяйте такие малоизвестные элементы, как `cite`, чтобы сослаться на книжный заголовок, газетную статью или сайт, а `address` — для обозначения контактной информации автора страницы (удобно применять для указания авторских прав).
- Не используйте элементы или их атрибуты для изменения внешнего вида текста, изображений. Все это выполняют каскадные таблицы стилей.
- Если нет HTML-элемента, соответствующего контенту, который вы хотите выделить на странице для придания ему определенного внешнего вида, то пользуйтесь элементами `div` и `span`. О них я расскажу в следующих главах.
- Не злоупотребляйте элементом `div`. Некоторые дизайнеры полагают, что `div` — это все, что им нужно, и при этом игнорируют элементы, которые могут быть более уместны. Возьмем пример создания навигационной панели. Можно добавить блок `div` на страницу и заполнить его кучей ссылок. Но ведь гораздо лучше использовать маркированный список (элемент `ul`): в конце концов, навигационная панель сама по себе является списком ссылок. Как уже ранее упоминалось, HTML5 предоставляет несколько новых элементов, которыми можно заменить `div`, например `article`, `section` и `footer`. Для панели навигации можно воспользоваться HTML5-элементом `nav`.

- Никогда не забывайте указывать закрывающие теги HTML-элементов. Открывающий тег `<p>` требует соответствующего ему закрывающего тега `</p>`, как и любые другие элементы, за исключением одиночных, например `
` и ``.
- Проверяйте синтаксис своих веб-страниц с помощью W3C-валидатора (см. рис. 1.4 и врезку «В курс дела!» ранее в этой главе). Плохо написанный HTML-код, как и код с опечатками, вызовет множество непредсказуемых ошибок браузера.

Важность объявления типа документа

Язык HTML следует некоторым правилам. Вы сами должны сообщить браузеру, какой версией языка HTML пользуетесь, включив то, что называется *объявлением типа документа*, в начало веб-страницы. Это объявление типа документа указывается в первой строке HTML-файла и определяет используемую вами версию HTML (например, HTML5 или HTML 4.01 Transitional).

Если указать объявление типа документа с ошибкой или опустить его, то браузер переключится в другое состояние, называемое *режимом совместимости*. Режим совместимости — это попытка производителей браузеров заставить свое программное обеспечение вести себя подобно устаревшим браузерам, выпускавшимся ранее 1999 года (Netscape 4, Internet Explorer 5). Если современный браузер сталкивается со страницей, в которой отсутствует правильное объявление типа документа, то он «думает», что эта страница была написана в текстовом редакторе HTML давным-давно, и отображает ее так, как это сделал бы старый браузер. Именно поэтому без правильного объявления типа документа ваши отформатированные с помощью каскадных таблиц стили веб-страницы, возможно, не будут смотреться так, как они должны выглядеть в соответствии с текущими стандартами. Если, проверяя веб-страницу в браузере, вы невольно просматриваете ее в режиме совместимости, то можете не ломать себе голову, пытаясь исправить проблемы отображения. Они связаны с неправильным объявлением типа документа, а не с ошибками в коде HTML или CSS.

ПРИМЕЧАНИЕ

Для более подробного ознакомления с техническими особенностями работы браузеров в режиме совместимости посетите сайты tinypurl.com/nr9dehz и tinypurl.com/pm8k7ng.

Указать правильное объявление типа документа достаточно просто. Нужно только знать используемую версию HTML. Если используется HTML5, то все существенно упрощается. Объявление типа документа приобретает простой вид:

```
<!doctype html>
```

Поместите этот код в верхнюю часть своего HTML-файла, и все будет в порядке. Если вы по-прежнему пользуетесь старыми версиями HTML или XHTML, например HTML 4.01 Transitional и XHTML 1.0 Transitional, то объявление типа документа будет иметь более запутанный вид.

Если используется версия HTML 4.01 Transitional, то добавляйте приведенное ниже объявление типа документа в самом начале веб-страницы:

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


Объявление типа документа для XHTML 1.0 Transitional имеет похожий вид. Кроме того, необходимо добавить кое-что еще в открывающий тег `<html>`. Эта строка определяет тип используемого языка XML, в нашем случае это XHTML.

```
<!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">
```

Если все это навевает на вас тоску, упростите свою жизнь и воспользуйтесь объявлением типа документа, предлагаемым версией HTML5. Оно имеет краткую форму, легко запоминается, работает во всех браузерах и используется практически во всех новых веб-страницах. Это объявление типа документа можно указать, даже если не применяются никакие новые HTML5-элементы.

ПРИМЕЧАНИЕ

Большинство визуальных средств верстки веб-страниц, в число которых входит программа Dreamweaver, при создании новой веб-страницы автоматически добавляют объявление типа документа, а многие текстовые редакторы, поддерживающие код на языке HTML, предлагают использовать для добавления объявления типа документа специальные сочетания клавиш.

Как работают каскадные таблицы стилей

Каскадные таблицы стилей работают с HTML-кодом, но не имеют никакого отношения к языку HTML. Это совершенно другой язык. HTML структурирует документ, упорядочивая информацию в заголовки, абзацы, маркированные списки и т. д., в то время как CSS тесно взаимодействует с браузером, чтобы оформление HTML-документа имело совершенный вид.

Например, вы могли бы использовать HTML, чтобы превратить фразу в заголовок, отделяя его от содержимого страницы, но лучше использовать CSS для форматирования заголовка, скажем, крупным полужирным красным шрифтом с позиционированием на 50 пикселей от левого края окна. В CSS это форматирование текста включает в себя *стиль* — правило, описывающее внешний вид конкретной части веб-страницы. А *таблица стилей* является набором таких стилей.

Можно также создавать стили специально для работы с изображениями. Например, указать формат размещения изображений на странице. С помощью стилей можно выровнять изображение по правому краю веб-страницы, поместить его в цветную рамку, отделить от окружающего текста полем шириной 50 пикселей.

Браузер применяет созданные вами стили для текста, изображений, заголовков и других элементов страницы. Например, вы можете создать стиль и применить его к одному абзацу на странице, чтобы мгновенно изменить размер, цвет и семейство шрифта в этом абзаце. Вы можете создавать стили, которые будут применяться к конкретным HTML-элементам. Так, например, браузер будет одинаково отображать на сайте все заголовки первого уровня (h1) независимо от того, где они расположены. Вы даже можете создавать стили, которые будут применяться только к определенным элементам, отмеченным вами особым образом в HTML-коде.

Создав стиль один раз, можно применять его к текстовым фрагментам, изображениям, заголовкам и любым другим элементам страницы сколько угодно раз. На-

пример, вы можете выбрать абзац текста и применить к нему стиль, тут же изменяющий размер, цвет и шрифт текста. Можно также создать стили для определенных HTML-элементов так, чтобы, например, все заголовки первого уровня (элементы h1) на вашем сайте были отображены в одинаковом стиле независимо от того, где они размещены.

В КУРС ДЕЛА!

Версии CSS

Как производители операционных систем и смартфонов iPhone, так и создатели CSS постоянно выпускают новые версии своих программных продуктов. Версия CSS 1, появившаяся в 1996 году, стала основой для дальнейшего совершенствования языка каскадных таблиц стилей. В первой версии языка CSS были введены основные понятия и команды: базовая структура таблиц стилей, концепция селекторов (см. главу 3) и большая часть свойств CSS.

В версии CSS 2 были добавлены новые возможности, включая совместимость с различными принтерами, мониторами и другими устройствами. Здесь также были добавлены новые селекторы и возможность точно позиционировать элементы на веб-страницах.

Следующая версия, CSS 2.1, на сегодняшний день является стандартом языка каскадных таблиц стилей. Она унаследовала все возможности и особенности CSS 1. Сюда добавлены некоторые новые свойства, а также исправлены неточности предыдущих версий. Она не содержит радикальных изменений по сравнению с CSS 2, и большинство браузеров давно адаптированы под эту версию.

Несколько лет назад была выпущена версия CSS3 (с частичной поддержкой браузерами). В отличие

от предыдущих версий, CSS3 не является единым стандартом. По мере возрастания сложности CSS Консорциум W3C разбил CSS на несколько отдельных модулей — модуль Selectors, модуль Values and Units, модуль Box Alignment и т. д. Поскольку каждый модуль может разрабатываться независимо от других, какого-то единого стандарта под названием CSS3 не существует. Фактически работа над уровнем 3 модуля Selectors уже завершена и начата работа над уровнем 4.

Иными словами, то, что известно как CSS3, на самом деле является разобщенной коллекцией различных модулей на разных стадиях завершения. Поддержка некоторых новейших модулей уже включена разработчиками браузеров в их продукты, а другие модули в любом отдельно взятом браузере могут не иметь никакой поддержки. В будущем какой-либо стандарт CSS4 не появится, будут только новые версии различных модулей, каждый из которых будет находиться на своем уровне разработки.

Поэтому в данной книге рассматривается ядро CSS 2.1 (которое просто было перенесено в различные модули CSS3), а также наиболее впечатляющие, популярные и широко поддерживаемые новые свойства CSS.

2 Создание стилей и таблиц стилей

Даже самые сложные и красивые сайты (в том числе и тот, который представлен на рис. 2.1) когда-то начинались с определения единственного CSS-стиля. Постепенно, по мере добавления все новых таблиц стилей, можно разработать полностью законченный дизайн сайта. Независимо от того, являетесь вы новичком в изучении CSS или профессиональным веб-дизайнером, всегда следует помнить несколько основных правил создания таблиц стилей. В этой главе мы начнем изучение CSS с основных понятий, которыми нужно руководствоваться в процессе создания и использования таблиц стилей.

СОВЕТ

Многие люди лучше воспринимают материал, обучаясь сразу на конкретных примерах, предпочитая их чтению книг и руководств. Если вы сначала хотите попробовать свои силы в создании таблиц стилей, а затем вернуться к этому теоретическому материалу, чтобы прочитать о том, что только что сами сделали, перейдите к практикуму в конце текущей главы.

Анатомия стиля

Определение стиля в CSS, устанавливающего внешний вид какого-либо элемента (фрагмента) веб-страницы, — это всего лишь правило, которое сообщает браузеру, что и каким образом форматировать: изменить цвет шрифта заголовка на синий, выделить фото красной рамкой, создать меню шириной 150 пикселей для списка гиперссылок. Если бы стиль мог говорить, он сказал бы: «Браузер, сделай, чтобы вот *это* выглядело *так-то*».

Фактически определение стиля состоит из двух основных элементов: самого элемента веб-страницы, который непосредственно подлежит форматированию браузером, — селектора, а также команд форматирования — блока объявления. Селекторами могут быть заголовок, абзац текста, изображение и т. д. Блоки объявления могут, например, окрасить текст в синий цвет, добавить красную рамку (границу) вокруг абзаца, установить фотографию в центре страницы — возможности форматирования бесконечны.

Например, тело сайта, показанного на рис. 2.1, включает простой стиль:

```
body {  
  font-family: "Franklin-Book", Helvetica, Arial, sans-serif;  
  color: #222;  
}
```



Рис. 2.1. Любая отформатированная с помощью CSS веб-страница состоит из отдельных определений стилей

ПРИМЕЧАНИЕ

Как сотрудники Консорциума W3C, так и веб-дизайнеры часто называют CSS-стили правилами. В этой книге оба термина взаимозаменяемы.

Разумеется, CSS-стили не могут быть написаны на обычном языке, как, например, предыдущий абзац. У них есть собственный язык. В частности, чтобы установить красный цвет и размер шрифта 1,5 em для всех абзацев на веб-странице, нужно написать следующее:

```
p { color: red; font-size: 1.5em; }
```

Этот стиль как бы говорит браузеру: «Раскрась текст всех абзацев веб-страницы, заключенных в элемент p, красным цветом и установи размер шрифта равным 1,5 em (*em* — единица измерения размера шрифта текста в браузере, см. главу 6). Любой стиль, даже самый простой, содержит несколько элементов (рис. 2.2).

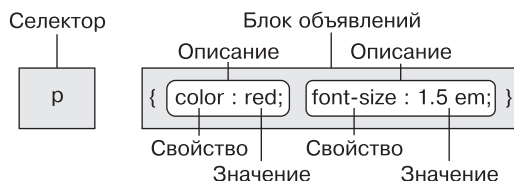


Рис. 2.2. Стиль (или правило) состоит из двух основных частей: селектора и блока объявления

- **Селектор.** Как уже было отмечено, селектор сообщает браузеру, к какому элементу или элементам веб-страницы применяется стиль: к заголовку, абзацу, изображению или гиперссылке. На рис. 2.2 селектор `p` обращается к элементу `p`, передавая браузеру, что все элементы `p` нужно форматировать, используя объявления, указанные в данном стиле. Благодаря большому разнообразию селекторов, предлагаемых языком CSS, и небольшому творческому потенциалу вы сможете мастерски форматировать веб-страницы (в следующей главе селекторы описаны более подробно).
- **Блок объявления.** Код, расположенный сразу за селектором, содержит все команды форматирования, которые можно применить к этому селектору. Блок начинается с открывающей фигурной скобки `{` и заканчивается закрывающей `}`.
- **Объявление.** Между открывающей и закрывающей фигурными скобками блока объявления можно добавить одно или несколько *объявлений* — команд форматирования. Каждое объявление имеет две части — *свойство* и *значение*. Двоеточие отделяет имя свойства от его значения, и все объявление заканчивается точкой с запятой.
- **Свойство.** Каскадные таблицы стилей предлагают большой выбор команд форматирования, называемых *свойствами*. Свойство представляет собой слово или несколько написанных через дефис слов, определяющих конкретный стиль. У большинства свойств есть соответствующие простые для понимания имена, такие как `font-size`, `margin-top`, `background-color` и т. д. (в переводе с английского: размер шрифта, верхний отступ, цвет фона). В следующих главах будет описано множество полезных свойств CSS. После имени свойства нужно добавить двоеточие, чтобы отделить его от значения.

СОВЕТ

В приложении 1 вы найдете удобный перечень свойств CSS.

- **Значение.** Наконец настал тот момент, когда вы можете задействовать свой творческий потенциал, присваивая *значения* CSS-свойствам: к примеру, устанавливая фоновый цвет синим, красным, фиолетовым, салатным и т. д. Как будет описано в следующих главах, различные свойства каскадных таблиц стилей требуют указания определенных типов значений — цвета (`red` или `#FF0000`), размера (`18px`, `200%` или `5em`), URL-адреса (`images/background.gif`), а также определенных ключевых слов (`top`, `center` или `bottom`).

Вам не обязательно описывать стиль на одной строке, как изображено на рис. 2.2. У стилей может быть множество свойств форматирования, и есть возможность упорядочить код таблицы стилей, разбивая объявления на строки. Например, поместите селектор и открывающую скобку на одной строке, каждое объявление — далее на отдельных строках, а закрывающую фигурную скобку — отдельно на последней строке стиля. Это будет выглядеть следующим образом:

```
p {  
  color: red;  
  font-size: 1.5em;  
}
```

Любой браузер игнорирует символы пробела и табуляции, так что вы можете спокойно добавлять их, создавая хорошо читаемые стили CSS. Кроме того, при перечислении свойств полезно сделать отступ табуляцией или несколькими пробелами для явного отделения селектора от блока объявления. К тому же один пробел между двоеточием и значением свойства, конечно, необязателен, но он обеспечивает дополнительную удобочитаемость стилей. Фактически можно добавить любое количество пробелов там, где вам захочется. Например, `color:red,color:red` и `color : red` — все варианты будут правильно работать.

ПРИМЕЧАНИЕ

Не забывайте указывать в конце каждой пары «свойство/значение» точку с запятой:

```
color: red;
```

Пропуская эту точку с запятой, вы собьете с толку браузер, в результате чего таблица стилей будет нарушена и веб-страница отобразится некорректно.

Однако не стоит переживать — описанная ошибка достаточно распространена, поэтому используйте валидатор каскадных таблиц стилей, о котором будет рассказано далее в этой главе.

Концепция таблиц стилей

Конечно, один стиль не превратит веб-страницу в произведение искусства. Он может выделить абзацы красным цветом, но, чтобы придать сайту красивый и стильный внешний вид, вам придется указать множество различных стилей. Весь набор определяемых CSS-стилей включается в *таблицу стилей*. Таблицы стилей бывают двух видов — *внутренние* и *внешние* — в зависимости от того, где определена информация о стилях: непосредственно в самой веб-странице или в отдельном файле, связанном с веб-страницей.

Внутренние или внешние таблицы стилей? Еще с момента изобретения CSS внешние таблицы стилей были лучшим способом создания дизайна веб-страниц. Они упрощают создание и поддержку сайтов и ускоряют их обновление. Внешняя таблица стилей сосредотачивает всю информацию о стилях в едином файле, который вы затем присоединяете к странице, написав для этого всего одну строку кода. Вы можете присоединить одну и ту же внешнюю таблицу стилей к каждой странице сайта, создавая таким образом единый дизайн. А обновление внешнего вида всего сайта будет заключаться лишь в редактировании единственного текстового файла — внешней таблицы стилей.

Внешние таблицы стилей позволяют веб-страницам загружаться быстрее. Когда вы используете такие таблицы стилей, веб-страницы содержат только HTML-код, без кода громоздких вложенных таблиц для форматирования, без элементов `font` и подобных, без кода внутренних каскадных таблиц стилей. Кроме того, когда браузер загрузит внешнюю таблицу стилей, он сохранит этот файл на клиентском компьютере посетителя веб-страницы (в специальной системной папке, называемой *кэшем*) для быстрого доступа к нему. Когда посетитель веб-страницы переходит к другим страницам сайта, которые используют ту же внешнюю таблицу стилей, браузеру нет необходимости снова загружать

таблицу стилей. Он попросту загружает запрашиваемый HTML-файл и использует внешнюю таблицу стилей из своего кэша, что дает существенный выигрыш во времени загрузки страниц.

ПРИМЕЧАНИЕ

Когда вы работаете над своим сайтом и пользуетесь предварительным просмотром в браузере, кэш работает без всякой пользы для вас. Подробнее об этом рассказано во врезке «Обходной прием» далее в этой главе.

ОБХОДНОЙ ПРИЕМ

Не попадитесь с кэшированием

Кэш браузера обеспечивает значительное увеличение скорости просмотра веб-страниц.

Всякий раз, когда браузер открывает веб-страницу, он помещает загруженную информацию в кэш. Сюда также попадают такие часто используемые файлы, как внешние каскадные таблицы стилей или изображения, что позволяет сэкономить время загрузки веб-страниц. Вместо того чтобы в следующий раз (при просмотре других страниц этого же сайта или повторного просмотра этих страниц в будущем) повторно загружать те же файлы, браузер может использовать их прямо из кэша, будь то просмотренная ранее веб-страница или рисунок.

Однако не всегда то, что хорошо для посетителей сайта, удобно для вас. Поскольку браузер кэширует файлы внешних таблиц стилей CSS и повторно обращается к ним, это часто сбивает с толку, например во время работы над дизайном сайта. Допустим, работая над форматированием веб-страницы, которая использует внешние таблицы стилей, вы просматриваете ее в браузере, чтобы убедиться, что достигли желаемого результата. Но не все выглядит так, как было задума-

но, хотя, на ваш взгляд, в CSS-коде ошибок нет. Вы возвращаетесь в редактор HTML-кода и вносите изменения в файл внешних таблиц стилей CSS. Снова вернувшись в браузер и перезагрузив страницу для просмотра результатов только что внесенных изменений, вы видите, что никаких изменений не произошло! Вы только что попали в ловушку, связанную с особенностями кэша. Дело в том, что при перезагрузке веб-страницы браузер не всегда перезагружает данные, уже находящиеся в кэше, в том числе внешние таблицы стилей. Таким образом, невозможно увидеть, как выглядит веб-страница, отформатированная с помощью только что отредактированного CSS-кода из внешней таблицы стилей.

Чтобы обойти эту путаницу, можно выполнить *принудительную перезагрузку* страницы (вместе с перезагрузкой всех связанных файлов), нажав и удерживая клавишу Ctrl (⌘), а затем нажав кнопку Обновить (Reload) в окне браузера. Кроме того, в браузерах Chrome и в Internet Explorer для этой цели предназначено сочетание клавиш Ctrl+F5 (⌘+F5), в Firefox — Ctrl+Shift+R (⌘+Shift+R), а Ctrl+R (⌘+R) работает как в Safari, так и в Chrome.

Внутренние таблицы стилей

Внутренняя таблица стилей — это набор стилей, являющийся частью кода веб-страницы, которая всегда должна находиться между открывающим и закрывающим тегами `<style>` и `</style>` HTML-кода в теле веб-страницы, то есть внутри элемента `head`. Например:

```
<style>
hl {
  color: #FF7643;
```

```
    font-family: Arial;
  }
  p {
    color: red;
    font-size: 1.5em;
  }
</style>
</head>
<body>
```

<!-- Далее следует остальная часть вашей веб-страницы... -->

ПРИМЕЧАНИЕ

Вы можете поместить элемент `style` и все его стили после элемента `title`, но веб-дизайнеры обычно размещают их прямо перед закрывающим тегом `</head>`, как показано в примере выше. Однако если вы также используете на страницах JavaScript-код, он должен располагаться после таблиц стилей. Часто JavaScript-сценарии полагаются на CSS, поэтому, добавляя таблицы стилей первыми, вы гарантируете, что JavaScript-код будет иметь все необходимые для своего выполнения данные.

Элемент `style` относится к HTML, а не к CSS, но именно он сообщает браузеру, что данные, содержащиеся внутри, являются кодом CSS, а не HTML. Создание внутренней таблицы стилей идентично созданию внешней, с той лишь разницей, что перечень стилей не выносится в отдельный файл, а заключается между тегами элемента `style`.

Внутренние таблицы стилей можно легко добавить в веб-страницу, и так же просто перейти к редактированию HTML-кода этой же веб-страницы. Однако эти таблицы стилей отнюдь не являются самым эффективным способом для проектирования дизайна сайта, состоящего из множества страниц. С одной стороны, вам придется копировать и вставлять код внутренней таблицы стилей в каждую страницу сайта, а это не только трудоемкая, но еще и бессмысленная работа. Этот код делает каждую страницу вашего сайта громоздкой. К тому же такая страница медленно загружается.

Кроме того, внутренние таблицы стилей доставляют много трудностей при обновлении дизайна сайта. Например, нужно изменить представление заголовков первого уровня (заключенных в элемент `h1`), которые первоначально отображались крупным полужирным шрифтом зеленого цвета. Теперь вы хотите, чтобы заголовки были написаны маленьким шрифтом Courier синего цвета. Используя внутренние таблицы стилей, пришлось бы редактировать *каждую* страницу сайта. У кого-нибудь найдется столько времени? К счастью, для устранения данной проблемы нашлось простое решение — использование внешних таблиц стилей.

ПРИМЕЧАНИЕ

Иногда можно прибегнуть к способу добавления информации о стилях непосредственно к каждому конкретному HTML-элементу без применения таблицы стилей. В практикуме этой главы будет показано, как выполнить такой маневр, используя встроенные стили. Мы не рекомендуем применять встроенные стили для разработки веб-страниц, однако многие программисты JavaScript используют их для динамического добавления HTML-контента на страницы. Это яркий пример создания и работы встроенных стилей.

Внешние таблицы стилей

Внешняя таблица стилей — это не что иное, как текстовый файл, содержащий весь набор стилей CSS. Он не должен включать в себя HTML-код, поэтому никогда не добавляйте в файл внешней таблицы стиля элемент `style`. Добавок имя этого файла всегда должно заканчиваться расширением CSS. Можно присвоить какое угодно имя этому файлу, но лучше, чтобы оно было информативным. Назовите файл внешней таблицы стилей, например, `global.css`, `site.css` или просто `styles.css`, если это общая таблица стилей, связанная со всеми страницами вашего сайта, или `form.css`, если он содержит код для форматирования веб-форм сайта.

В КУРС ДЕЛА!

Проверяйте правильность CSS-кода

Так же как вы должны были удостовериться в правильности написания HTML-кода веб-страниц (см. врезку «В курс дела!» в главе 1), проверьте CSS-код на отсутствие ошибок. На сайте Консорциума доступен инструмент для проверки синтаксиса кода CSS: tinyclub.com/382hc. Он работает, как и HTML-валидатор: можно указать URL-адрес веб-страницы (или только адрес к внешнему CSS-файлу), загрузить CSS-файл или скопировать и вставить код в форму, а затем нажать кнопку запуска проверки.

При наборе CSS-кода очень просто сделать опечатку или ошибку, которой вполне достаточно для того, чтобы изменить до неузнаваемости весь тщательно продуманный дизайн страниц сайта. Если веб-страница, содержащая CSS-код, выглядит не так, как вы ожидали, то причиной тому может быть небольшая ошибка в коде. CSS-валидатор Консорциума — первое средство поиска проблем с дизайном веб-страниц.

СОВЕТ

Если имеется веб-страница с внутренней таблицей стилей, а вы хотите использовать внешнюю таблицу, то вырежьте код стилей, расположенный между тегами элемента `style` (без самих тегов). Потом создайте новый текстовый файл и вставьте в него CSS-код. Сохраните файл с расширением CSS, например `global.css`, и свяжите его с вашей веб-страницей, как описано далее.

Создав внешнюю таблицу стилей, вы должны подключить ее к формируемой веб-странице. Это можно сделать с помощью HTML-элемента `link`:

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

Элемент `link` обладает двумя атрибутами:

- `rel="stylesheet"` — указывает тип ссылки; в данном случае это ссылка на таблицу стилей;
- `href` — определяет местонахождение внешнего CSS-файла на сайте. Значение этого атрибута — URL-адрес, который будет отличаться в зависимости от того, где расположен CSS-файл. Он работает так же, как атрибут `src` при добавлении изображения на страницу или атрибут `href` гиперссылки, указывающей на другую веб-страницу.

СОВЕТ

На одну и ту же страницу вы можете добавить несколько элементов link, указывающих на разные каскадные таблицы стилей. Способы организации стилей описаны в главе 18.

Практикум: создание стилей

В этом разделе речь пойдет об основных приемах создания CSS-стилей, в том числе встроенных, а также внутренних и внешних таблиц стилей. По мере прочтения книги на практических примерах вы научитесь создавать различные CSS-стили, от простых элементов дизайна до полноценных CSS-ориентированных макетов веб-страниц. Перед началом урока нужно загрузить файлы с материалом для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку Download ZIP в правом нижнем углу страницы). Файлы каждой главы находятся в отдельных папках с названиями 02 (для второй главы), 03 (для третьей) и т. д.

ПРИМЕЧАНИЕ

Кроме папок с заданиями к каждой главе, вы обнаружите другие папки с уже выполненными заданиями. Например, в папке 02_finished находятся файлы готовых заданий к главе 2. Используйте их в качестве подсказки, если запутались и хотите сравнить свой результат с конечным.

Затем следует запустить редактор HTML-кода, которым вы пользуетесь, будь то простой текстовый редактор (Блокнот (Notepad) или TextEdit) или программный комплекс для визуального проектирования (Sublime Text, Notepad++ или Adobe Dreamweaver).

Создание встроенного стиля

Размещая код CSS непосредственно в HTML-коде страницы, вы создаете встроенный стиль. Встроенные стили не экономят ни время загрузки веб-страниц, ни трафик, поэтому нет никаких причин для их использования. В крайнем случае, если обязательно нужно изменить стиль единственного элемента одной веб-страницы, можно прибегнуть к встроенным стилям. (Например, создавая HTML-форматированные электронные письма, лучше использовать встроенные стили. Это, к слову, единственная возможность заставить CSS работать в Gmail.) Если вы все-таки применяете этот метод и хотите, чтобы стиль работал должным образом, то уделите особое внимание размещению команд форматирования внутри элементов, которые следует отформатировать. Рассмотрим пример, наглядно демонстрирующий, как это делается.

1. В редакторе HTML-кода откройте файл 02\index.html.

Этот простой и изящно написанный HTML5-файл содержит несколько заголовков, абзац, маркированный список и информацию об авторском праве в элементе address. Начнем с создания встроенного стиля для элемента h1.

2. В открывающем элементе <h1> укажите свойство стиля style="color: #6A94CC;". Элемент должен выглядеть следующим образом:

```
<h1 style="color: #6A94CC;">
```

Атрибут `style` относится к HTML, а не к CSS, поэтому после него указывается символ `=`, а значение атрибута — весь код CSS — заключено в кавычки. Код CSS — это только та часть, что находится в кавычках. В данном случае вы добавили свойство `color`, которое воздействует на цвет текста, и установили его значение равным `#6A94CC`, то есть шестнадцатеричному коду, определяющему синий цвет (об изменении цвета текста читайте в главе 6). Двоеточие отделяет имя свойства от значения, которое вы хотите установить для данного свойства. Далее проверим результат в браузере.

3. Откройте страницу `index.html` в браузере.

Запустите любой браузер и выберите команду меню **Файл** ▶ **Открыть** (**File** ▶ **Open**) или нажмите сочетание клавиш **Ctrl+O** и выберите файл `index.html` в папке **02** с файлами примеров на своем компьютере (вы можете просто перетащить этот файл из окна папки в открытое окно браузера). Во многих HTML-редакторах доступна функция **Предварительный просмотр в браузере** (**Preview in Browser**), которая при нажатии определенного сочетания клавиш или выборе пункта меню открывает страницу для просмотра в браузере. Изучите справочные материалы к HTML-редактору: возможно, в нем есть команда, которая сэкономит ваше время. Открыв страницу в браузере, вы увидите, что заголовок стал синим. Встроенные стили могут содержать несколько свойств CSS. Добавим в тег еще одно свойство.

4. Вернитесь в HTML-редактор. После точки с запятой за кодом `#6A94CC` наберите код `font-size: 3em;`.

Точка с запятой отделяет два различных свойства. Тег `<h1>` должен выглядеть следующим образом:

```
<h1 style="color: #6A94CC; font-size: 3em;">
```

5. Посмотрите на страницу в браузере. Нажмите кнопку **Обновить** (**Reload**), но сначала удостоверьтесь, что сохранили HTML-файл.

Теперь заголовок стал гораздо больше. Вы почувствовали, как непросто добавлять встроенные стили? Придание соответствующего вида всем заголовкам `h1` веб-страницы требует выполнения тех же действий над каждым из них. На это могут уйти часы и даже целые дни набора и добавления кода в ваш HTML-файл.

6. Вновь перейдите в редактор веб-страниц и удалите из элемента `h1` весь код атрибута форматирования, вернув его к нормальному виду.

Далее мы создадим таблицу стилей внутри веб-страницы (окончательная версия этой части практикума представлена файлом `inline-style.html` в папке `02_finished`).

Создание внутренней таблицы стилей

Вместо встроенных стилей лучше использовать таблицу стилей, содержащую множество стилей CSS, каждый из которых придает внешний вид своему элементу страницы. Прочитав этот раздел, вы научитесь создавать стиль, который изменяет внешний вид всех заголовков первого уровня за один прием. Этот единственный стиль автоматически отформатирует все элементы `h1` веб-страницы.

1. В файле `index.html`, открытом в текстовом редакторе, установите курсор сразу после закрывающего тега `</title>`, нажмите клавишу **Enter** и наберите код `<style>`.

Теперь HTML-код должен выглядеть следующим образом (текст, который нужно добавить, выделен полужирным шрифтом):

```
<title>Большая книга CSS -- Глава 2</title>
<b>style</b>
</head>
```

Открывающий тег `<style>` указывает на начало таблицы стилей. Желательно сразу же, как только вы набираете открывающий тег, закрывать его, поскольку об этом очень легко забыть, переключив внимание на написание CSS. В данном случае вы закроете элемент `style` до того, как станете добавлять стили CSS.

2. Нажмите клавишу **Enter** дважды и наберите код `</style>`.

Теперь вы добавите селектор CSS, обозначающий начало вашего первого стиля.

3. Щелкните кнопкой мыши между открывающим и закрывающим тегами элемента `style` и введите `h1 {`.

Значение `h1` определяет элемент, к которому браузер должен применить последующий стиль.

Фигурную скобку после `h1` называют *открывающей*. Она обозначает начало определения CSS-свойств для данного стиля. Иначе говоря, за ней начинается самое интересное. Надо отметить, что, как и в случае с закрывающими тегами, желательно указывать *закрывающую скобку* стиля до непосредственного добавления каких-либо свойств этого стиля.

4. Дважды нажмите клавишу **Enter** и укажите одну закрывающую скобку `}`.

Ответная закрывающая скобка, соответствующая введенной в предыдущем шаге открывающей, должна сообщить браузеру, что этот CSS-стиль здесь заканчивается.

5. Перейдите к пустой строке между двумя скобками, нажмите клавишу **Tab** и введите код `color: #6A94CC;`.

Это текст того же свойства стиля, что и во встроенном варианте, — свойство `color` со значением `#6A94CC`. Точка с запятой обозначает окончание объявления свойства.

ПРИМЕЧАНИЕ

Исходя из синтаксиса языка CSS, не обязательно размещать каждое свойство стиля на отдельной строке, но это в ваших же интересах. Построчный набор свойств облегчает просмотр таблицы стилей и позволяет визуально выделить каждое свойство. Кроме того, есть еще правило оформления, рекомендуемое для лучшего представления структуры CSS-кода, — использование табуляции (вместо нее можно также добавлять несколько пробелов). Такое смещение строк кода обеспечивает быстрый и понятный просмотр таблиц стилей, выстраивая селекторы (в данном примере в качестве селектора выступает `h1`) в одну линию вдоль левого края страницы и располагая свойства на одинаковом уровне со смещением вправо.

6. Снова нажмите клавишу **Enter** и добавьте дополнительно два свойства, как показано ниже:

```
font-size: 3em;
margin: 0;
```

Убедитесь в том, что вы не забыли указать точку с запятой в конце каждой строки, иначе код CSS некорректно отобразится в браузере.

Каждое из этих свойств придает заголовку определенный визуальный эффект. Первое свойство назначает размер и шрифт текста, в то время как второе удаляет воздух (пустое пространство) вокруг заголовка. Более подробно об этих свойствах читайте во второй части книги.

Поздравляю — вы только что создали внутреннюю таблицу стилей! Код, который вы добавили на веб-страницу, должен выглядеть так, как выделенный ниже полужирным шрифтом:

```
<title>Большая книга CSS – Глава 2</title>
<style>
h1 {
  color: #6A94CC;
  font-size: 3em;
  margin: 0;
}
</style>
</head>
```

7. Сохраните страницу и просмотрите ее в браузере.

Вы можете сделать это так, как описано в третьем шаге, или, если страница все еще открыта в окне браузера с предыдущего раза, нажмите кнопку **Обновить** (Reload).

Теперь мы добавим другой стиль.

ПРИМЕЧАНИЕ

Никогда не забываете добавлять закрывающий тег `</style>` в конце внутренней таблицы стилей. Если вы не сделаете этого, браузер отобразит на экране код таблицы стилей, за которым последует сама веб-страница без всякого форматирования, а может быть и такое, что браузер вообще не покажет содержимого веб-страницы.

8. Переключитесь обратно в HTML-редактор, установите курсор после закрывающей фигурной скобки стиля `h1`, который вы только что создали, нажмите клавишу **Enter** и добавьте следующий стиль:

```
p {
  font-size: 1.25em;
  color: #616161;
  line-height: 150%;
  margin-top: 10px;
  margin-left: 60px;
}
```

Этот стиль форматирует все абзацы веб-страницы. Не переживайте, что пока не знаете, что делает каждое из описываемых свойств CSS. В последующих главах эти свойства будут подробно описаны. А пока просто потренируйтесь правильно набирать код и прочувствуйте, каково это — добавлять CSS на страницу.

9. Просмотрите страницу в браузере.

Страница выглядит так, как показано на рис. 2.3. Вы видите, как изменился стиль абзаца под первым заголовком? Вы можете посмотреть окончательную версию этой части примера, открыв файл `internal-stylesheet.html` из папки `02_finished`.

Все то, чем вы занимались в практикуме, можно назвать «CSS в двух словах»: начать с HTML-страницы, добавить таблицу стилей, создать прочие CSS-стили, чтобы заставить страницу прилично выглядеть. В следующей части практикума вы увидите, как можно более эффективно работать, используя внешние таблицы стилей.

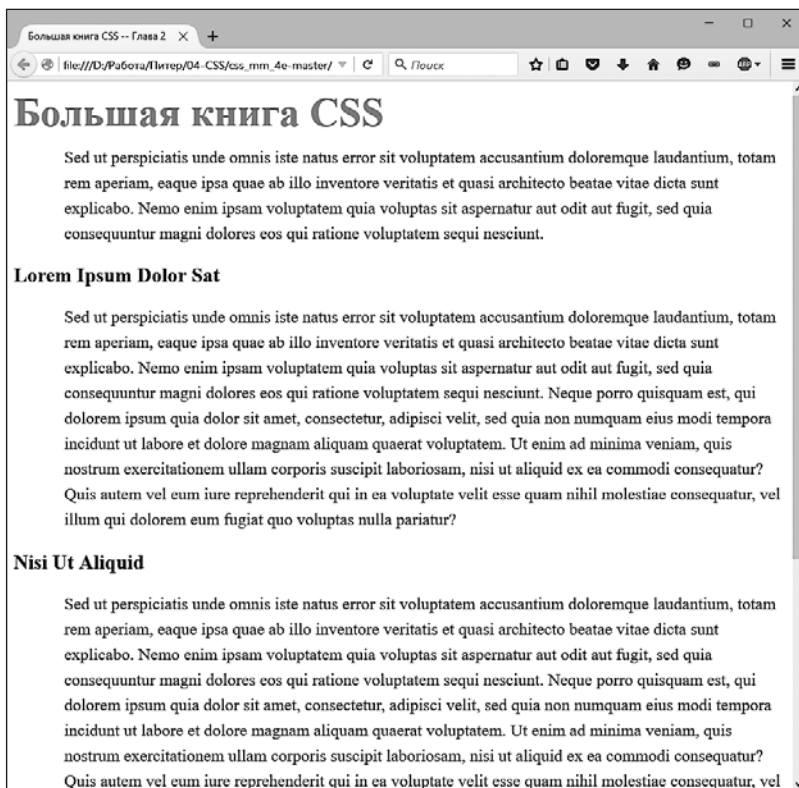


Рис. 2.3. Каскадные таблицы стилей легко справляются с форматированием текста, позволяя изменять начертание, размер, цвет шрифтов текста и даже добавлять декоративные рамки и подчеркивание и многое другое, о чем вы узнаете в главе 6

Создание внешней таблицы стилей

Поскольку во внутренних таблицах все стили сгруппированы в начале веб-страницы, создавать и редактировать их намного проще и удобнее, чем встроенные стили, с которыми вы имели дело до этого. Кроме того, внутренние таблицы стилей позволяют форматировать любое количество экземпляров элементов веб-страницы одновременно (как в примере с элементом `h1`), создав один простой стиль (правило). Внешние таблицы стилей не только наследуют преимущества внутренних таблиц, но и имеют дополнительные плюсы: в них можно хранить все стили для всех страниц сайта. Редактирование одного стиля во внешней таблице обновляет стиль целого сайта. В этом разделе вы возьмете стили, созданные в предыдущем уроке, и поместите их во внешнюю таблицу стилей.

1. В редакторе HTML-кода создайте новый файл и сохраните его под именем `styles.css` в той же самой папке, где находится веб-страница, над которой вы сейчас работаете.

Файлы внешних таблиц стилей должны заканчиваться расширением `.css`. Имя файла `styles.css` указывает на то, что стили, содержащиеся в файле, используются глобально для всего сайта (вы, конечно, можете использовать любое понравившееся имя).

Приступим к добавлению нового стиля к таблице стилей.

2. Наберите в файле `styles.css` следующее правило:

```
html {  
    padding-top: 25px;  
    background-image: url(images/bg_page.png);  
}
```

Это правило касается `html` — элемента, окружающего все остальные HTML-элементы, имеющиеся на странице. Свойство `padding-top` добавляет пространство между верхней частью элемента и помещаемым внутри него содержимым. Иными словами, набранный код приведет к добавлению 25 пикселей пустого пространства между верхней частью окна браузера и содержимым страницы. Свойство `background-image` добавляет к странице фоновое изображение. Свойство `background-image` может отобразить графику множеством различных способов, но в данном случае изображение будет выведено в виде неперекрывающейся мозаики слева направо и сверху вниз. Дополнительные сведения о свойствах фоновых изображений будут даны в главе 8.

3. Добавьте второе правило сразу же после только что введенного в файле `styles.css`:

```
body {  
    width: 80%;  
    padding: 20px;  
    margin: 0 auto;  
    border-radius: 10px;  
    box-shadow: 10px 10px 10px rgba(0,0,0,.5);  
    background-color: #E1EDEB;  
}
```

Это правило относится к телу веб-страницы (элементу `body`), которое включает в себя всю информацию, видимую в окне браузера. В этом определении стиля задается множество различных свойств, каждое из которых еще будет рассмотрено более подробно в книге. Но если дать стилю краткое описание, он создает прямоугольную область (блок) для содержимого страницы, составляющую 80 % от ширины окна браузера, имеющую внутри себя небольшое пустое пространство, сдвигающее текст от краев блока (свойство `padding`), и производит центрирование блока на странице (свойство `margin`), в частности конкретный прием центровки содержимого страницы, рассматриваемый в практикуме главы 3. И наконец, блок получает светло-синий цвет фона и прозрачную темную тень.

Вместо повторного набора стилей, созданных в предыдущем уроке, просто скопируем стили, созданные ранее, и вставим их в эту внешнюю таблицу стилей.

4. Откройте страницу `index.html`, над которой работали, и скопируйте весь текст, содержащийся внутри тегов элемента `style` (не копируйте сами теги элемента `style`).

Скопируйте код стилей тем же самым способом, которым скопировали бы любой текст. Например, с помощью меню Правка ▶ Копировать (Edit ▶ Copy) или нажатием сочетания клавиш `Ctrl+C`.

5. В файл `styles.css` вставьте этот код стилей либо посредством меню Правка ▶ Вставить (Edit ▶ Paste), либо нажатием сочетания клавиш `Ctrl+V`.

Внешняя таблица стилей никогда не должна содержать HTML-код, именно поэтому вы и не копировали теги элемента `style`.

6. Сохраните файл `styles.css`.

Теперь нужно из HTML-файла удалить CSS-код и связать новую таблицу стилей с этим файлом.

7. Вернитесь к файлу `index.html` в своем текстовом редакторе и удалите теги элемента `style` и все CSS-стили, определенные в нем ранее.

Вам больше не нужны эти стили внутренней таблицы, поскольку они перенесены во внешнюю таблицу стилей, которую сейчас нужно присоединить к HTML-файлу. В этом уроке вы окунетесь в захватывающий мир веб-шрифтов. Все, что касается веб-шрифтов, будет рассмотрено в главе 6, но основная идея заключается в том, что вы можете использовать на веб-страницах практически любые шрифты, даже те, которые посетители вашего сайта не установили на своих компьютерах, предоставив просто ссылку на файл с этим шрифтом. Существует множество различных способов применения веб-шрифтов, но в данном примере будет использована служба веб-шрифтов Google.

8. В позиции HTML-файла, где находилась встроенная таблица стилей (между закрывающими тегами `</title>` и `<head>`), введите следующий код:

```
<link href='http://fonts.googleapis.com/css?family=Kurale' rel='stylesheet'>
```

Как и прежде, вам пока не стоит вдаваться в подробности. На этом этапе нужно лишь знать, что, встретившись с данной ссылкой, браузер загружает с сервера Google шрифт, который называется Kurale и который могут свободно использовать ваши CSS-стили.

Затем вы укажете ссылку на созданную ранее внешнюю таблицу стилей.

9. После только что добавленного элемента `link` наберите следующий код:

```
<link href="styles.css" rel="stylesheet">
```

Элемент `link` определяет местонахождение внешней таблицы стилей. Атрибут `rel` оповещает браузер о том, что ссылка ведет на таблицу стилей.

ПРИМЕЧАНИЕ

В данном примере файл внешней таблицы стилей расположен в той же самой папке, что и веб-страница, так что использование имени файла в качестве значения `href` предполагает простой путь относительно документа. А если бы он находился в любой другой папке, путь был бы иным. В таком случае для указания местонахождения файла нужно задавать путь либо относительно самого документа, то есть веб-страницы, либо относительно корневого каталога сайта. Применяется такая же методика, как и при указании гиперссылки на другую веб-страницу в HTML-теге `` (информацию по этой теме смотрите по адресу tinypurl.com/oe6j3sr).

10. Сохраните файл и просмотрите его в браузере.

Вы увидите контент с теми же стилями для элементов `h1` и `p`, которые были созданы ранее во внутренней таблице стилей. Кроме того, теперь есть пятнистый желто-коричневый фон (фоновое изображение, примененное в элементе `html`), а также светлый сине-зеленый прямоугольник. Он является элементом `body`, и его ширина составляет 80 % от ширины окна браузера. Попробуйте изменить размер окна браузера и обратите внимание, что размер прямоугольника также изменяется. Прямоугольник также отбрасывает тень, но сквозь нее можно увидеть пятнистый фон. Все это получается благодаря цветовой модели `rgba`, включающей установки прозрачности (которая будет рассмотрена далее).

ПРИМЕЧАНИЕ

Если на получившейся веб-странице отсутствует форматирование (к примеру, заголовок отображается шрифтом малого размера и никак не выделен), то, вероятно, в шаге 6 вы набрали код с ошибкой или сохранили файл `styles.css` в папке, отличной от той, в которой находится файл `index.html`. В этом случае просто переместите файл `styles.css` в ту же самую папку.

Теперь вы увидите веб-шрифт, ссылка на который была добавлена в шаге 8.

11. Запустите текстовый редактор и вернитесь к файлу `styles.css`. Добавьте к стилю селектора `h1` следующие две строки:

```
font-family: 'Kurale', 'Arial Black', serif;
font-weight: normal;
```

В конечном результате стиль должен приобрести следующий вид (добавления выделены полужирным шрифтом):

```
h1 {
  font-family: 'Kurale', 'Arial Black', serif;
  font-weight: normal;
  color: #6A94CC;
  font-size: 3em;
  margin: 0;
}
```


Теперь при просмотре страницы в браузере вы увидите заголовок, отформатированный шрифтом Kurale.

ПРИМЕЧАНИЕ

Если вы не видите новый шрифт с тонкими линиями (засечками) на концах букв, показанный на рис. 2.4, значит, какое-то из двух обстоятельств не сложилось. Может отсутствовать подключение к Интернету, что мешает загрузить шрифт с сервера Google, а кроме того, вы можете допустить опечатку либо в коде тега `<link>` (при выполнении шага 8), либо в объявлении `font-family` (во второй строке показанного выше кода).

Чтобы продемонстрировать пользу от хранения ваших стилей в их собственных отдельных файлах, вы можете прикрепить таблицу стилей к другой веб-странице.

12. Откройте файл `02\another_page.html`.

Эта веб-страница содержит те же самые HTML-элементы: `h1`, `h2`, `p` и др., с которыми вы уже работали.

13. Установите курсор после закрывающего тега `</title>` и нажмите клавишу `Enter`.

Сейчас нужно добавить к этой веб-странице ссылку на веб-шрифт и уже созданную внешнюю таблицу стилей.

14. Наберите код элемента `link`, который применялся в шагах 8 и 9.

Код веб-страницы должен выглядеть следующим образом (код, который вы только что набрали, отмечен полужирным шрифтом):

```
<title>Другая страница</title>
  <link href='http://fonts.googleapis.com/css?family=Kurale' rel='stylesheet'>
  <link href="styles.css" rel="stylesheet">
</head>
```

15. Сохраните страницу и просмотрите ее в браузере.

Достаточно всего двух строк кода, чтобы мгновенно преобразить внешний вид веб-страницы. Чтобы показать, насколько просто обновить стиль, описанный во внешней таблице стилей, попробуйте отредактировать один из стилей или добавить другие.

16. Откройте файл `styles.css` и добавьте объявление свойства `font-family`: `"Palatino Linotype", Baskerville, serif`; — в начале стиля элемента `p`.

Код должен выглядеть следующим образом (добавленный код выделен полужирным шрифтом):

```
p {
  font-family: "Palatino Linotype", Baskerville, serif;
  font-size: 1.25em;
  color: #616161;
  line-height: 150%;
  margin-top: 20px;
```

```
margin-left: 60px;
}
```

В данном случае веб-шрифт не используется и все зависит от наличия перечисленных шрифтов на машине посетителя сайта (все вопросы об использовании шрифтов будут рассмотрены позже). Затем создайте новый стиль для элемента h2.

17. Установите курсор после заключительной фигурной скобки } стиля элемента p, нажмите клавишу Enter и добавьте такое правило:

```
h2 {
  color: #B1967C;
  font-family: 'Kurale', 'Arial Black', serif;
  font-weight: normal;
  font-size: 2.2em;
  border-bottom: 2px white solid;
  background: url(images/head-icon.png) no-repeat 10px 10px;
  padding: 0 0 2px 60px;
  margin: 0;
}
```

С большинством этих CSS-свойств вы уже знакомы, однако некоторые из них новые, например border-bottom, используемое для добавления линии под заголовком. Свойство background вообще предоставляет возможность комбинировать различные свойства — в данном случае это background-image и background-repeat — в одно. Не беспокойтесь о назначении этих свойств, вы подробно изучите их в последующих главах. О свойствах шрифта читайте в главе 6, о свойствах, устанавливающих отступы и границы, — в главе 7, а о свойствах, устанавливающих параметры фона, — в главе 8.

Стили, которые вы создавали до сих пор, работают с основными элементами h1, h2 и p, изменяя оформление каждого их экземпляра. Другими словами, стиль p, который вы создали, форматирует каждый абзац на странице. Но если вы хотите воздействовать на конкретный абзац, нужно использовать другой вид стиля.

18. Перейдите к концу стиля h2, нажмите клавишу Enter после закрывающей скобки } и наберите следующий код:

```
.intro {
  color: #666666;
  font-family: 'Kurale', Helvetica, sans-serif;
  font-size: 1.2em;
  margin-left: 0;
  margin-bottom: 25px;
}
```

Если вы просмотрите файл index.html в браузере, то пока не заметите никаких изменений. Этот тип стиля, называемый *селектором класса*, форматирует только отдельные элементы, к которым вы примените класс. Для того чтобы этот новый стиль работал, придется немного отредактировать HTML-код.

19. Сохраните файл `styles.css` и перейдите к файлу `index.html` в своем текстовом редакторе. Найдите открывающий тег `<p>`, следующий за элементом `h1`, и добавьте код `class="intro"`. Открывающий тег должен выглядеть так:

```
<p class="intro">
```

Вам не нужно добавлять точку перед словом `intro`, как вы это делали, создавая стиль в шаге 18 (почему так, вы узнаете в следующей главе). Дополнительный HTML-код применяет стиль к первому абзацу (и только к первому).

Повторите этот шаг для файла `another_page.html` — добавьте `class="intro"` к первому тегу `<p>` на этой странице.

20. Сохраните все файлы и просмотрите страницы `index.html` и `another_page.html` в браузере.

Обратите внимание, что вследствие простого редактирования CSS-файла внешний вид обеих веб-страниц изменился. Закройте глаза и представьте, что ваш сайт содержит тысячи страниц. Мощные средства изменения дизайна, не правда ли?

Вам осталось выполнить еще одно небольшое изменение. Если вы посмотрите в нижнюю часть страницы в браузере, то увидите информацию об авторских правах. Форматирование этой строки отличается от стиля абзацев, расположенных над ней. Страница смотрелась бы гораздо лучше, если бы их стили совпадали.

21. Вернитесь к файлу `styles.css` в текстовом редакторе. Найдите стиль с селектором `p`. Добавьте запятую, пробел и слово `address`.

Стиль должен выглядеть следующим образом:

```
p, address {
  font-family: "Palatino Linotype", Baskerville, serif;
  font-size: 1.25em;
  color: #616161;
  line-height: 150%;
  margin-top: 20px;
  margin-left: 60px;
}
```

Вы изменили не свойства стиля, а его селектор. Фактически вы создали групповой селектор, который очень эффективен в случаях, когда необходимо применить один и тот же стиль ко множеству элементов страницы. Подробнее об этом способе вы узнаете позже. В данном случае стиль применялся к двум элементам: `p` и `address`.

22. Сохраните все файлы и просмотрите страницу `index.html`.

На рис. 2.4 представлен окончательный вид страницы (вы можете найти ее в папке `02_finished`).

Для практики попробуйте внести в файл `styles.css` какие-либо изменения. Поэкспериментируйте с различными значениями свойств. Можно, например, изменить значение свойства `width` стиля `body` или изменить размер шрифта.

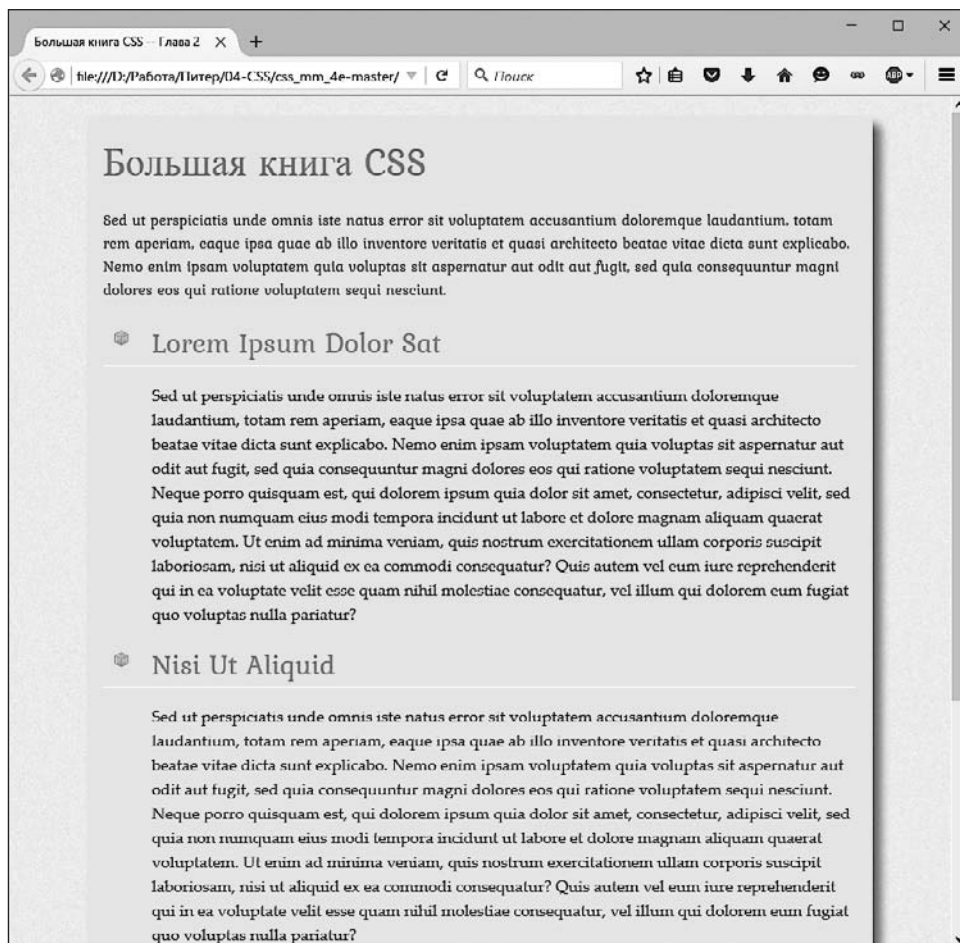


Рис. 2.4. Применение внешних таблиц стилей позволяет обновить дизайн всех страниц сайта за один прием посредством редактирования единственного CSS-файла. Полное исключение CSS-кода из HTML-документа и вынесение его в отдельный файл может значительно уменьшить размер самих веб-страниц, следовательно, они будут загружаться намного быстрее

3 Селекторы: выбор форматируемых элементов

Каждый CSS-стиль состоит из двух частей: селектора и блока объявления (о нем говорилось в предыдущей главе), который, в свою очередь, содержит правила форматирования — свойства цвета, размера шрифта текста и т. д. Они относятся лишь к оформлению. Возможность фокусировки каскадных таблиц стилей на отдельных элементах заключается как раз в самых первых символах, начинающих определение любого стиля, — селекторах. Именно селектор контролирует дизайн веб-страницы (рис. 3.1), определяя элемент, который вы хотите изменить. Другими словами, он используется для форматирования множества элементов одновременно. Если дать *более* подробное описание, то селекторы позволяют выбрать один или несколько схожих элементов для их последующего изменения с помощью свойств в блоке объявления. Селекторы CSS — большой потенциал для создания дизайна веб-страниц.

```
h1 {  
    font-family: Arial, sans-serif;  
    color: #CCCCFF;  
}
```

Рис. 3.1. Здесь первая часть стиля — селектор — определяет элементы, подлежащие форматированию. В данном случае h1 означает «все заголовки первого уровня (элементы h1) веб-страницы»

ПРИМЕЧАНИЕ

Если вы хотите немного попрактиковаться на примерах, прежде чем изучать теоретический материал по теме селекторов, обратитесь к практикуму в конце этой главы.

Селекторы тегов

Селекторы, которые используют для применения стилей к определенным HTML-элементам, называются *селекторами тегов* (или *селекторами элементов*). Они

представляют собой весьма эффективное средство проектирования дизайна веб-страниц, поскольку определяют стиль всех экземпляров конкретного HTML-элемента. С их помощью можно быстро изменять дизайн веб-страницы с небольшими усилиями. Например, если надо отформатировать все абзацы текста, используя шрифт одного начертания, размера, а также цвета, то вам просто нужно создать стиль с селектором `p` (применительно к элементу `p`). Он определяет, каким образом браузер отобразит отдельно взятый элемент (в данном случае `p`).

До появления CSS, чтобы отформатировать текст, вам пришлось бы заключить его по всем абзацам в элемент `font` многократно. Этот процесс занял бы много времени, не говоря о том, что код HTML-страниц при этом увеличится в объеме до невероятных размеров, страницы будут загружаться медленнее, их обновление будет занимать много времени. С селекторами тегов вам фактически ничего не нужно делать с HTML-кодом, вы создаете CSS-стили и позволяете браузеру сделать все остальное.

Селекторы исключительно просто определить в CSS-стилях, так как они наследуют имя формируемых элементов — `p`, `h1`, `table`, `img` и т. д. Например, на странице, представленной на рис. 3.2, селектор `h2` (*вверху*) определяет представление каждого заголовка второго уровня страницы (*внизу*), которых в данном случае три (см. метки в левой части окна браузера).

```
h2 {
  font-family:"Century Gothic", "Gill Sans", sans-serif;
  color:#000000;
  margin-bottom:0;
}
```

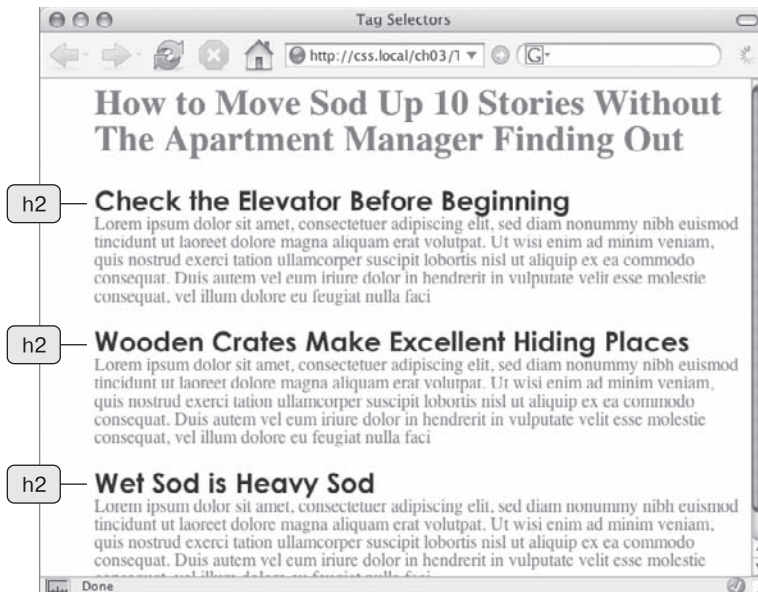


Рис. 3.2. Селектор тега воздействует на все экземпляры указанного элемента веб-страницы

ПРИМЕЧАНИЕ

Как видно из рис. 3.2, селекторы тегов указываются без символов < и >, которые окружают теги HTML-элементов. Поэтому при написании правила, например, для тега <p> просто вводите его имя: p.

Однако и здесь есть свои недостатки. Как сделать так, чтобы не все абзацы веб-страницы выглядели одинаково? Селекторами тегов этого добиться не удастся, потому что они не предоставляют достаточную информацию браузеру. Например, нужно задать различия между элементом p, выделенным определенным цветом и кеглем, и элементом p, для которого вы хотите оставить шрифт черного цвета. Каскадные таблицы стилей предоставляют сразу несколько способов решения данной проблемы, самый простой из которых — использование *классов (селекторов классов)*.

Классы: точное управление

Если вы хотите, чтобы какие-то элементы выглядели не так, как другие родственные им элементы на той же веб-странице, например, хотите задать для одного или двух рисунков красную рамку, оставив все остальные изображения без нее, то можете использовать селектор классов. Если вы привыкли работать со стилями в текстовых редакторах, таких как Microsoft Word, то классы покажутся вам хорошо знакомыми. Вы создаете селектор, назначая ему имя, а затем применяете его лишь к тем HTML-элементам, которые хотите отформатировать. Например, вы можете создать класс .copyright и с его помощью выделить абзац, содержащий информацию об авторских правах, не затрагивая остальные абзацы.

Классы позволяют указать конкретный элемент веб-страницы, независимо от тегов. Предположим, вы хотите отформатировать одно или несколько слов абзаца. В данном случае применяется форматирование не ко всему элементу p, а лишь к фрагменту абзаца. Таким образом, вам нужно использовать класс для выделения определенного текста. Можно применить изменения к множеству элементов, окруженных различными HTML-тегами.

Например, вы можете придать какому-то абзацу и заголовку второго уровня (элемент h2) одинаковый стиль, как показано на рис. 3.3. В отличие от селекторов тегов, которые ограничивают вас существующими на веб-странице HTML-элементами, с помощью этого метода вы можете создать любое количество классов и поместить их в выбранные позиции.

Посмотрите на рис. 3.3. Вы можете отформатировать один экземпляр заголовка h2 (с текстом Wet Sod is Heavy Sod). Класс .special сообщает браузеру о необходимости применения стиля к единственному элементу h2. Создав его один раз, вы можете пользоваться им и в дальнейшем для любых элементов. В примере такой стиль применен к верхнему абзацу.

ПРИМЕЧАНИЕ

Если вы хотите применить класс всего к нескольким словам текста, содержащегося в произвольном элементе HTML-кода (подобно среднему абзацу на рис. 3.3), используйте элемент span. Для более детального ознакомления с ним смотрите врезку «В курс дела!» далее в этой главе.



Рис. 3.3. Классы позволяют целенаправленно изменять дизайн фрагментов веб-страниц

Вы, вероятно, обратили внимание на точку, с которой начинается имя каждого класса: `.copyright`, `.special`. Это одно из правил, которые необходимо иметь в виду при именовании классов.

- Все имена классов должны начинаться с точки. С ее помощью браузеры находят классы в каскадной таблице стилей.
- При именовании классов разрешается использование только латинских букв, цифр, дефисов и знаков подчеркивания.
- Имя после точки всегда должно начинаться с латинской буквы. Например, `.9lives` — неправильное имя класса, а `.crazy8` — правильное. Можно называть классы, скажем, именами `.copy-right` и `.banner_image`, но не `.-bad` или `._as_bad`.

○ **Имена классов чувствительны к регистру.** Например, `.SIDEBAR` и `.sidebar` рассматриваются языком CSS как различные классы.

Классы создаются так же, как селекторы тегов. После имени указывается блок объявления, содержащий все необходимые свойства:

```
.special {  
  color:#FF0000;  
  font-family:"Monotype Corsiva";  
}
```

Поскольку селекторы тегов распространяются на все типы веб-страницы, их достаточно определить в таблице стилей и они начинают работать. Форматируемые HTML-элементы уже присутствуют в коде веб-страницы. Чтобы воспользоваться преимуществами, которые обеспечивают классы, требуется выполнить еще несколько действий, ведь использование классов — двухэтапный процесс. После того как вы создали класс, надо указать, где вы хотите его применить. Для этого добавьте атрибут `class` к HTML-элементу, который следует отформатировать.

Допустим, вы создали класс `.special` с целью выделения определенных элементов веб-страницы. Чтобы применить этот стиль к абзацу, добавьте атрибут `class` в тег `<p>`, как показано ниже:

```
<p class="special">
```

ПРИМЕЧАНИЕ

Вы не должны указывать точку перед именем класса в HTML-коде (в атрибуте `class`). Она требуется только в имени селектора таблицы стилей.

Таким образом, когда браузер сталкивается с этим элементом, он знает, что правила форматирования, содержащиеся в стиле `.special`, необходимо применить к данному тексту. Вы можете также использовать класс только в части абзаца или заголовка, добавив элемент `span`. Например, чтобы выделить только одно слово в абзаце, используя стиль `.special`, можно написать следующее:

```
<p>Добро пожаловать в ресторан <span class="special">Медуза</span>,  
  который удивит вас изысками морской кухни.</p>
```

Создав класс, можно применить его практически к любому элементу веб-страницы. Вообще, вы можете применять один и тот же класс к разным элементам, создав, к примеру, стиль `.special` с особым шрифтом и цветом для элементов `h2`, `p` и `ul`.

Один элемент, несколько классов. Вы можете применять не только один и тот же класс к различным элементам, но и несколько классов к одному элементу. Хотя сначала может показаться, что создание нескольких классов и добавление их имен к одному элементу — лишняя работа, это лишь общий подход.

Приведу примеры, когда бы вы могли применить несколько классов к одному элементу: представьте себе, что вы разрабатываете интерфейс для управления корзиной покупок посетителя сайта. Интерфейс требует разнообразных кнопок, каждая из которых выполняет какое-либо действие. Одна кнопка может быть использована для удаления товара из корзины, другая позволяет добавлять товар в нее, а третья служит для изменения количества товаров.

Будучи педантичным дизайнером, вы хотите, чтобы кнопки не только обладали некоторым сходством, например имели закругленные углы и одинаковый шрифт, но и различались: кнопка для удаления товара из корзины была красной, а кнопки для добавления — зелеными и т. д. Для достижения согласованности дизайна вы можете создать два класса. Один класс будет применяться ко всем кнопкам, а второй — к их определенным типам.

Для начала создадим класс `.btn` (сокращение от английского слова `button` — «кнопка»):

```
.btn {  
  border-radius: 5px;  
  font-family: Arial, Helvetica, serif;  
  font-size: .8 em;  
}
```

Далее необходимо создать дополнительные классы для каждого типа кнопок:

```
.delete {  
  background-color: red;  
}  
.add {  
  background-color: green;  
}  
.edit {  
  background-color: grey;  
}
```

Затем, применяя несколько классов к элементу, вы можете комбинировать стили, тем самым соблюдая согласованность между кнопками, но сохраняя уникальный вид каждого типа.

```
<button class="btn add">Добавить</button>  
<button class="btn delete">Удалить</button>  
<button class="btn edit">Изменить</button>
```

Браузеры и язык HTML не испытывают никаких проблем в случаях, когда к одному элементу применено несколько классов. Все, что нужно сделать, — добавить атрибут `class` к HTML-элементу и в качестве его значения — имя каждого класса, разделяя их запятой. Браузер объединит свойства из различных классов, применив законченные стили к элементам. В данном примере для текста каждой из кнопок будет использоваться шрифт Arial размером 0,8 em, а углы самих кнопок будут закруглены. Тем не менее цвета каждой из кнопок будут различны: кнопка **Добавить** — зеленого цвета, кнопка **Удалить** — красного, а кнопка **Изменить** — серого.

Преимущество этого подхода заключается в том, что, если вы решите, что углы кнопок больше не должны быть закруглены или что для текста на кнопках должен использоваться другой шрифт, вам нужно лишь изменить стиль `.btn`, чтобы обновить внешний вид всех кнопок. С другой стороны, если вы решите, что кнопка **Изменить** должна быть желтой, а не серой, изменение стиля `.edit` повлияет только на нее и не затронет другие кнопки.

Идентификаторы: отдельные элементы веб-страницы

В языке CSS идентификаторы предназначены для *идентификации* уникальных частей веб-страниц, таких как шапка, панель навигации, область основного контента и т. д. Как и при использовании классов, вы создаете идентификатор (ID-селектор), придумав ему имя, а затем применяете его к HTML-коду своей веб-страницы. Так в чем же различие? Как описано во врезке «В курс дела!», идентификаторы имеют дополнительное специфическое применение. Например, в веб-страницах с использованием JavaScript-сценариев или в очень объемных страницах. Другими словами, существует несколько вынужденных причин для использования идентификаторов.

ПРИМЕЧАНИЕ

В сообществе веб-разработчиков наметился тренд к отказу от идентификаторов в каскадных таблицах стилей. Причины этого явления требуют более глубокого знания языка CSS, чем то, которое может быть получено при чтении данной книги на текущий момент, поэтому здесь идентификаторы будут встречаться редко, а все причины того, почему применение идентификаторов нельзя признать удачным решением, будут изложены позже.

В КУРС ДЕЛА!

HTML-элементы `div` и `span`

В главе 1 были кратко описаны `div` и `span` — два универсальных HTML-элемента, которые можно использовать в любых ситуациях. Когда нет HTML-элемента, который бы однозначно указывал, где следует разместить класс или идентификатор форматирования, для заполнения промежутков используйте `div` и `span`.

Логическое *деление* страницы на такие фрагменты, как шапка, панель навигации, боковые панели и колонтитул, обеспечивает элемент `div`. Вы также можете его использовать, чтобы охватить любой фрагмент, включающий несколько последовательных элементов веб-страницы, в том числе заголовки, маркированные списки, абзацы (программисты называют эти элементы *блочными*, потому что они формируют логически законченный блок контента с разрывами строк до и после такого блока). Элемент `div` функционирует как элемент абзаца: указываем открывающий тег `<div>`, далее добавляем некоторый текст, рисунок или какой-либо другой контент, а затем закрываем код тегом `</div>`.

Элемент `div` имеет уникальную способность включать в себя *несколько* блочных элементов, являясь средством группировки (к примеру, логотип и панель навигации или блок новостей, составляющий боковую панель). После

группировки содержимого веб-страницы таким образом вы можете применить определенное форматирование исключительно к элементам, находящимся внутри этого фрагмента `div`, или переместить (позиционировать) весь отмеченный фрагмент содержимого в конкретное место веб-страницы, например в правую часть окна браузера. О компоновке макетов средствами каскадных таблиц стилей рассказывается в части III книги.

Например, вы добавили на веб-страницу фотографию, у которой есть подрисованная подпись. Вы можете для группировки обернуть оба объекта в элемент `div` (с применением к нему класса):

```
<div class="photo">  
  
<p>Я, мама и папа в отпуске в Антарктике.</p>  
</div>
```

В зависимости от того, как вы объявите стиль, класс `.photo` может добавить декоративную рамку, цвет фона и прочее сразу ко всему блоку, включающему и фотографию, и подпись к ней. В части III описываются еще более мощные способы применения `div`, включая вложенные конструкции из этих элементов.

В КУРС ДЕЛА!

Последние версии языка HTML содержат множество блочных элементов, которые работают схожим с `div` образом, но предназначены для контента определенного типа. Например, для отображения картинки или подписи вместо элемента `div` вы можете использовать элемент `figure`. Тем не менее, поскольку Internet Explorer 8 не поддерживает HTML5-элементы (см. врезку «Как заставить браузер Internet Explorer 8 поддерживать HTML5-элементы» в главе 1), для группировки нескольких HTML-элементов в одно целое многие дизайнеры до сих пор используют элементы `div`.

Кроме того, новые HTML5-элементы предназначены добавлять «смысл» вашему HTML-коду. Например, элемент `article` используется для автономных блоков текста, таких как статья в журнале. Тем не менее не вся разметка имеет смысл, поэтому в своей рабо-

те для простой группировки элементов вы по-прежнему будете прибегать к элементам `div`.

С другой стороны, элемент `span` позволяет применять классы и идентификаторы к фрагменту — *части* HTML-элемента. С его помощью вы можете выхватить из абзаца отдельные слова и фразы (которые часто называются *строчными* элементами), чтобы форматировать их отдельно друг от друга.

В данном примере класс форматирования с именем `.companyName` форматировал строчные элементы "CosmoFarmer.com", "Disney" и "ESPN":

```
<p>Добро пожаловать на сайт <span class="companyName">CosmoFarmer.com</span>, дочерней компании таких корпораций, как <span class="companyName">Disney</span> и <span class="companyName">ESPN</span>...шутка.</p>
```

Хотя многие веб-разработчики уже не пользуются идентификаторами так часто, как раньше, все же следует знать, что они собой представляют и как работают. Применение идентификаторов не представляет сложности. Если в классах перед именем указывается точка (`.`), то в случае с идентификаторами сначала должен быть указан символ решетки (`#`). Во всем остальном руководствуйтесь теми же правилами, что и для классов (см. выше). Следующий пример устанавливает цвет фона, ширину и высоту элемента:

```
#banner {
  background: #CC0000;
  height: 300px;
  width: 720px;
}
```

Применение идентификаторов в HTML схоже с использованием классов, но требует другого атрибута с соответствующим именем — `id`. Например, для применения показанного выше стиля к элементу `div` нужно написать такой HTML-код:

```
<div id="banner">
```

Точно так же для указания того, что последний абзац страницы — единственный с информацией об авторских правах, вы можете создать идентификатор с именем `#copyright` и применить его к тегу этого абзаца:

```
<p id="copyright">
```

ПРИМЕЧАНИЕ

Символ решетки (`#`) используется только при описании стиля в таблице. При вставке же имени идентификатора в HTML-тег символ `#` указывать не нужно: `<div id="banner">`.

ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ**Специальное применение идентификаторов**

У идентификаторов имеется несколько преимуществ перед классами. Эти особенности фактически не имеют никакого отношения к каскадным таблицам стилей, поэтому могут вам и не понадобиться. Но если вам это интересно, приведу их.

- Одним из простых для JavaScript-программистов способов определения частей страницы и манипуляции ими является применение идентификатора к элементу страницы с последующим использованием JavaScript-сценариев со ссылками на этот идентификатор. Специалисты часто применяют идентификаторы к заполняемым элементам форм (например, к текстовым полям) для получения имени посетителя сайта и т. д. Это позволяет JavaScript получить доступ к полям форм и совершать с ними разные манипуляции, например, когда посетитель нажимает кнопку отправки формы, убеждаться в том, что поля заполнены.
- Идентификаторы также дают возможность создавать ссылки на определенные части веб-стра-

ниц, при этом быстро перемещаясь по ним. Если у вас есть алфавитный словарь терминов, можно использовать идентификатор для создания указателя по буквам алфавита. При щелчке кнопкой мыши на букве «З» посетители сразу же переходят к словам, начинающимся с этой буквы. Для этого вам не придется использовать CSS — достаточно и HTML.

Сначала добавьте атрибут `id` в позицию на странице, на которую вы хотите ссылаться. Например, в указателе вы можете добавить элемент `h2` с буквой из алфавита. Добавьте соответствующий идентификатор к каждому элементу `h2`: `<h2 id="Z">3</h2>`. Чтобы создать ссылку в HTML, добавьте символ `#` и имя идентификатора в конец URL-адреса — `index.html#Z`. Эта ссылка указывает непосредственно на элемент с идентификатором `Z` на странице `index.html` (использование идентификатора таким способом оказывает действие, аналогичное применению элемента привязки `a` в языке HTML: `3`).

Форматирование групп элементов

Иногда необходимо быстро применить одинаковое форматирование сразу к нескольким различным элементам веб-страницы. Например, нужно, чтобы все заголовки имели один и тот же цвет и шрифт. Создание отдельного стиля для каждого заголовка определенного уровня — слишком объемная работа. А если вы потом захотите изменить цвет всех заголовков одновременно, то придется все и обновлять. Для решения этой задачи лучше использовать *групповой селектор*, который позволяет применить стиль, описав его лишь один раз, одновременно ко всем элементам.

Создание групповых селекторов

Для работы с групповым селектором создайте список, в котором один селектор отделен от другого запятыми. Таким образом, получаем:

```
h1, h2, h3, h4, h5, h6 { color: #F1CD33; }
```

Здесь приведены только селекторы тегов, но можно использовать любые типы селекторов (и их сочетания). Например, стиль группового селектора, который определяет одинаковый цвет шрифта для элементов `h1`, `p` и любых других,

принадлежащих классу `.copyright`, а также для элемента с идентификатором `#banner`, выглядит так:

```
h1, p, .copyright, #banner { color: #F1CD33; }
```

ПРИМЕЧАНИЕ

Если вам нужно применить к нескольким элементам веб-страницы схожие и в то же время несколько различающиеся стили, то можно создать групповой селектор с общими командами и отдельные стили с уникальным форматированием для каждого индивидуального элемента. Другими словами, два (или больше) стили могут форматировать один и тот же элемент. Это и является мощной особенностью языка CSS (по этой теме см. главу 5).

Универсальный селектор

Групповой селектор можно рассматривать как подручное средство для применения одинаковых свойств различным элементам. Каскадные таблицы стилей предоставляют *универсальный селектор* `*` для выборки всех элементов веб-страницы вместо указания каждого тега *по отдельности*. Например, если вы хотите, чтобы все отображалось полужирным шрифтом, нужно добавить следующий код:

```
a, p, img, h1, h2, h3, h4, h5 ...и т. д... { font-weight: bold; }
```

Использование символа `*` — более быстрый способ сообщить CSS о выборке всех HTML-элементов веб-страницы:

```
* { font-weight: bold; }
```

Кроме того, вы можете использовать универсальный селектор в составе селектора потомков (также называемого вложенным селектором): применяете стиль ко всем элементам-потомкам, подчиненным определенному элементу веб-страницы. Например, `.banner *` выбирает все элементы внутри элемента, имеющего атрибут `class` со значением `.banner` (более подробно о селекторах потомков вы узнаете чуть позже).

Универсальный селектор не определяет тип элементов, поэтому трудно описать его воздействие на HTML-элементы сайта. По этой причине дизайнеры со стажем используют для форматирования различных элементов такую особенность языка CSS, как *наследование*. Она описана в следующей главе.

Тем не менее некоторые веб-дизайнеры используют универсальный селектор как способ очистки *всего* пространства (воздуха) вокруг блочных элементов. Как вы увидите позже, с помощью свойства `margin` можно добавить пространство вокруг элемента, а свойства `padding` — пространство между границей элемента и его содержимым. Для разных элементов браузеры автоматически добавляют воздух различного размера, поэтому один из способов начать с чистого листа заключается в том, чтобы удалить все пространство вокруг элементов с определенным стилем:

```
* {  
  padding: 0;  
  margin: 0;  
}
```

Форматирование вложенных элементов

Выбор типа селектора в каждом случае обусловлен конкретной целью. Селекторы тегов можно быстро и просто создать, но они придадут всем экземплярам форматируемого элемента одинаковый внешний вид. Это бывает необходимо, например, чтобы все заголовки второго уровня вашей веб-страницы выглядели одинаково. Классы и идентификаторы обеспечивают большую гибкость независимого форматирования отдельных элементов страницы, но создание стилей для классов или идентификаторов также требует добавления атрибута `class` или `id` с соответствующим значением к HTML-элементам, которые нужно отформатировать. Это не только усложняет работу, но и увеличивает объем кода HTML-файла. Все, что нужно в этом случае, — объединить простоту использования селекторов тегов с точностью классов и идентификаторов. И такое средство в CSS есть — это *селекторы потомков* (также называемые *вложенными селекторами*).

Они используются для того, чтобы согласованно отформатировать целый набор элементов, но только когда те расположены в определенном фрагменте веб-страницы. Их работу можно описать так: «Эй вы, элементы привязки на панели навигации, прислушайтесь. У меня есть для вас указания по форматированию. А все остальные элементы привязки, проходите мимо, вам здесь нечего смотреть».

Селекторы потомков позволяют форматировать элементы в зависимости от их связей с другими элементами. Чтобы разобраться, как они работают, придется немного покопаться в языке HTML. Понимание работы селекторов потомков позволит получить представление о функционировании других типов селекторов, работа которых описана далее.

ПРИМЕЧАНИЕ

Селекторы потомков на первый взгляд могут показаться немного сложными, однако это одна из наиболее важных технологий для эффективного и тонкого применения CSS. Запаситесь терпением, и вы скоро освоите их.

Дерево HTML

HTML-код, на котором написана любая веб-страница, похож на генеалогическое дерево. Первый используемый HTML-элемент — `html` — похож на главного прародителя всех остальных. Из него выходят `head` и `body`, следовательно, `html` является *предком* названных элементов.

Элемент, расположенный внутри другого, — его *потомок*. Элемент `title` в следующем примере — потомок `head`:

```
<html>
  <head>
    <title>Простой документ</title>
  </head>
  <body>
    <h1>Заголовок</h1>
    <p>Абзац с <strong>важным </strong>текстом.</p>
  </body>
</html>
```

Представленный выше HTML-код можно изобразить в виде схемы (рис. 3.4). Здесь показаны отношения между элементами веб-страницы. Сначала располагается `html`; от него отходят два раздела, представленные `head` и `body`. Они содержат и другие, которые, в свою очередь, могут включать еще элементы. Таким образом, рассматривая, какие из них являются вложенными, можно схематически изобразить любую веб-страницу.

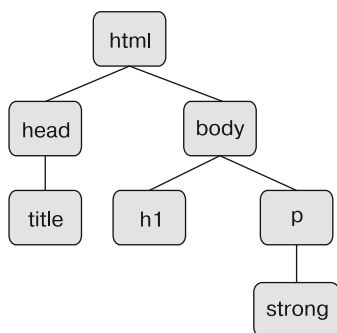


Рис. 3.4. HTML-код состоит из вложенных элементов. Связь между элементами, отображающая способ вложения одного в другой, представляет своего рода генеалогическое дерево

Схемы в форме дерева помогают выяснить и проследить, как CSS «видит» взаимодействие элементов веб-страницы. Функционирование многих селекторов, описанных в настоящей главе, включая селектор потомков, основывается на их родственных отношениях. Рассмотрим самые важные из них.

- **Предок.** Как описано в начале раздела, HTML-элемент, который заключает в себе другие элементы, — это предок. На рис. 3.4 это `html`, в то время как `body` — предок для всех содержащихся в нем элементов: `h1`, `p` и `strong`.
- **Потомок.** Элемент, расположенный внутри одного или более типов, — потомок. На рис. 3.4 `body` — потомок `html`, в то время как `p` — потомок одновременно и для `body`, и для `html`.
- **Родительский элемент.** Он связан с другими элементами *более низкого* уровня и находится выше в дереве. На рис. 3.4 `html` является родительским только для `head` и `body`. Элемент `p` — родительский по отношению к `strong`.
- **Дочерний элемент.** Элемент, непосредственно подчиненный другому элементу более высокого уровня, является дочерним. На рис. 3.4 оба элемента — `h1` и `p` — дочерние по отношению к `body`, но `strong` не является дочерним для `body`, так как он расположен непосредственно внутри элемента `p`, он является дочерним для этого элемента.
- **Родственный элемент.** Элементы, являющиеся дочерними для одного и того же родительского элемента, называются *родственными*, как братья или сестры. На рис. 3.4 они расположены рядом друг с другом. Элементы `head` и `body` — элементы одного уровня, как и `h1` и `p`.

На этом в CSS «родственные отношения» заканчиваются.

Создание селекторов потомков

Селекторы потомков позволяют использовать дерево HTML, форматируя элементы по-разному, в зависимости от того, где они расположены. Например, на веб-странице имеется элемент `h1` и вы хотите выделить слово внутри этого заголовка с помощью элемента `strong`. Проблема в том, что большинство браузеров отобразит весь заголовок полужирным шрифтом, поэтому посетитель страницы не сможет увидеть разницы между выделенным словом и другими частями заголовка. Создание селектора тега — не очень хорошее решение: так вы измените цвет *всех* вхождений элемента `strong` веб-страницы. Селектор же потомков позволит вам изменить цвет элемента *только в том случае*, если он расположен внутри заголовка первого уровня.

Решение проблемы выглядит следующим образом:

```
h1 strong { color: red; }
```

В данном случае *любой* текст, окруженный тегами элемента `strong`, находящегося внутри элемента `h1`, будет выделен красным цветом, но на другие экземпляры элемента `strong` на веб-странице этот стиль не повлияет. Вы могли добиться того же результата, создав класс, например `.StrongHeader`. Но в таком случае понадобилось бы вносить изменения в HTML, добавляя новый класс к элементу `strong` внутри заголовка. Подход, основанный на селекторах потомков, позволяет обойтись без лишней работы при создании стилей.

Селекторы потомков форматируют вложенные элементы веб-страницы, следуя тем же правилам, которым подчиняются элементы-предки и элементы-потомки в дереве HTML.

Вы создаете селектор потомков, объединяя селекторы вместе (согласно ветви дерева, которую нужно отформатировать), помещая самого старшего предка слева, а форматируемый элемент — справа. На рис. 3.5 все элементы веб-страницы — потомки элемента `html`. Элемент может происходить от множества других. Например, первый элемент `a` диаграммы является потомком `strong`, `p`, `body` и `html`. Обратите внимание на три ссылки (`a`) — элементы маркированного списка и еще одну, расположенную внутри абзаца. Чтобы отформатировать их иначе, вы можете создать стиль с селектором потомков:

```
li a { font-family: Arial; }
```

Это правило гласит: «Нужно отформатировать все ссылки (`a`), которые расположены в элементах списка (`li`), используя шрифт Arial». Вообще, селекторы потомков могут включать более двух элементов. Ниже представлены правильные определения для элементов `a`, находящихся в маркированных списках (см. рис. 3.5):

```
ul li a
body li a
html li a
html body ul li a
```

Все четыре селектора, представленные выше, имеют один и тот же эффект, что лишний раз демонстрирует отсутствие необходимости полностью описывать всех предков элемента, который вы хотите отформатировать. Например, во втором примере (`body li a`) определение `ul` не требуется. Этот селектор выполняет свои функции,

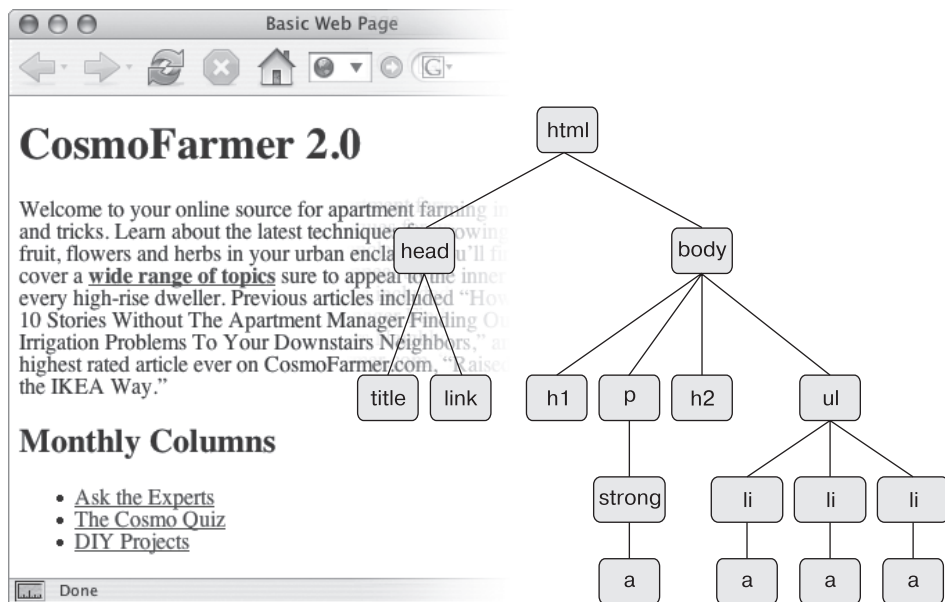


Рис. 3.5. Простейшая диаграмма в виде дерева, представляющая структуру веб-страницы

если есть `a` — потомок `li`. Его одинаково просто применить как к элементу `a`, расположенному внутри `em`, так и к `a` в `strong` или `li` и т. д.

В общем случае вы должны использовать кратчайший селектор потомка, который выполнит необходимое действие. Поскольку все элементы находятся внутри элементов `html` и `body`, включать их в селектор потомков нет причин.

ПРИМЕЧАНИЕ

Количество селекторов, включаемых вами в селектор потомков, влияет на то, как стиль взаимодействует с другими, конфликтующими стилями. Данное поведение называется специфичностью и описывается в главе 5.

Вы не ограничены использованием только селекторов тегов. Можно комбинировать различные типы селекторов в селекторе потомков. К примеру, нужно, чтобы ссылки были окрашены в желтый цвет, когда они находятся во вводимом элементе (который определен классом с именем `intro`). Этого можно добиться с помощью следующего селектора:

```
.intro a { color: yellow; }
```

Краткий перевод звучит так: «Нужно применить данный стиль ко всем ссылкам (`a`) — потомкам другого элемента, относящегося к классу с именем `intro`».

Создание модулей

Селекторы потомков часто используются для форматирования *модуля* кода, то есть коллекции HTML-элементов, выполняющих на странице конкретную функцию. Предположим, что на странице присутствует `div`-контейнер, используемый для

вывода перечня корпоративных новостей. HTML-код может иметь следующий вид:

```
<div>
  <h2>Наша компания великолепна!</h2>
  <p>Дополнительная информация о достижениях нашей компании</p>
  <h2>Другая новость</h2>
  <p>Текст другой новости...</p>
  <h2>...и т. д...</h2>
  <p>...и т. д... </p>
</div>
```

Если вставить атрибут `class` в открывающий тег `<div>` — `<div class="news">`, вы можете создать селекторы потомков, по-разному форматирующие HTML-элементы внутри раздела новостей. Например:

```
.news h2 { color: red; }
.news p { color: blue; }
```

Теперь содержимое элементов `h2` внутри раздела новостей будет отформатировано красным цветом шрифта, а абзацев — синим. Можно даже создать (что часто и делается) селекторы потомков с несколькими именами классов. Предположим, что вы создаете страницу, предоставляющую каталог адресов сотрудников какой-нибудь организации. Вы можете заключить каждую контактную информацию в ее собственный `div`-контейнер, а затем улучшить внешний вид элементов внутри этого контейнера с помощью атрибута `class`:

```
<div class="contact">
  <p class="name">Василий Теркин</p>
  <p class="phone">495-555-1234</p>
  <p class="address">Советская, 5</p>
</div>
```

Затем можно создать несколько селекторов потомков для назначения стиля только этим элементам контактной информации:

```
.contact .name { font-weight: bold; }
.contact .phone { color: blue;}
.contact .address { color: red; }
```

ПРИМЕЧАНИЕ

Иногда в таблице стилей можно увидеть следующий код:

```
p.intro
```

Может показаться, что это похоже на селектор потомка, поскольку в нем присутствует как имя HTML-элемента, так и имя класса, но это не так. Между `p` и `.intro` нет пробела, значит, чтобы этот стиль работал, класс `intro` должен быть применен конкретно к элементу `p` (`<p class="intro">`). Если добавить пробел, вы получите другой эффект:

```
p .intro { color: yellow; }
```

Это несколько иной вариант, выбирающий любой элемент, форматируемый под класс `.intro`, который, в свою очередь, является потомком элемента `p`. Иными словами, он выбирает не абзац, а другой элемент внутри абзаца. В общем, чтобы максимально сохранить гибкость стилей вашего класса, лучше оставить в покое HTML-элемент (иными словами, использовать только `.intro` вместо `p.intro`).

Псевдоклассы и псевдоэлементы

Иногда требуется выбрать фрагмент веб-страницы, в котором вообще нет элементов, но в то же время его достаточно просто идентифицировать. Это может быть первая строка абзаца или ссылка, на которую наведен указатель мыши. Каскадные таблицы стилей предоставляют для этих целей псевдоклассы и псевдоэлементы.

Форматирование ссылок

Ниже представлены четыре псевдокласса, которые позволяют форматировать ссылки в зависимости от того, какое действие над ними выполняет посетитель веб-страницы.

- `a:link` — обозначает любую ссылку, по которой посетитель веб-страницы еще не переходил и на которую не наведен указатель мыши. Это обычный стиль непосещенных гиперссылок.
- `a:visited` — обозначает любую ссылку, по которой посетитель веб-страницы уже переходил. Она сохраняется в истории браузера. Можно создать для этого типа стиль, отличный от обычного, чтобы сообщить посетителю, что он уже посетил эту страницу. (Ограничения, связанные с этим селектором, будут рассмотрены во врезке в начале главы 9.)
- `a:hover` — позволяет изменять вид ссылки, на которую посетитель навел указатель мыши. Вы можете добавить визуальные эффекты трансформации (ролловер-эффект), которые служат для улучшения визуального восприятия, например, на кнопки панели навигации.

Кроме того, можно использовать псевдокласс `:hover` для применения стилей к элементам веб-страниц, отличным от ссылок. Например, вы можете применить его для выделения фрагмента текста, заключенного в теги элемента `p` или `div`, каким-либо эффектом форматирования в тот момент, когда посетитель веб-страницы перемещает указатель мыши поверх этого фрагмента. В этом случае вместо `a:hover` для добавления эффекта наведения указателя поверх ссылки вы можете создать стиль с именем `p:hover`, добавляющий эффект при наведении указателя мыши поверх абзаца. А если вы хотите добавить стиль к элементам с классом `highlight`, создайте стиль `highlight:hover`.

- `a:active` — позволяет определить, как будет выглядеть ссылка *во время* выбора ее посетителем веб-страницы. Другими словами, это стиль во время кратковременного щелчка кнопкой мыши.

В главе 9 описывается, как спроектировать дизайн ссылок при использовании этих селекторов для хорошего восприятия посетителями сайта (например, элементов навигации для перехода по разделам сайта и т. д.).

ПРИМЕЧАНИЕ

Если вы не собираетесь заниматься веб-дизайном профессионально, то не утруждайтесь чтением последующих разделов о селекторах. Многие специалисты вообще никогда ими не пользуются. Все, что вы уже изучили, — селекторы тегов, селекторы потомков, классы, идентификаторы и т. д. — по-

эволюит создавать красивые, функциональные, легко обновляемые, профессиональные с точки зрения дизайна сайты. Если вы готовы к практической части, то переходите сразу к практикуму данной главы. Вы можете закончить изучение теоретического материала позже в любое удобное для вас время.

Форматирование фрагментов абзаца

На этапе становления Всемирной паутины не стоял вопрос типографического дизайна (в стиле глянцевого журналов) страниц и сайтов, никто и не думал о том, что текст должен выглядеть красиво. Теперь же это важно. В CSS для форматирования текста предусмотрено два псевдоэлемента — `:first-letter` и `:first-line`. Их использование придаст вашим веб-страницам изящное оформление, которым печатные издания обладают уже на протяжении многих лет.

Псевдоэлемент `:first-letter` позволяет создавать *буквицу* — начальный символ абзаца, который выделяется из остального текста, как в начале книжной главы.

Форматирование первой строки с помощью псевдоэлемента `:first-line` отличным от основного абзаца цветом притягивает посетителей веб-страницы изяществом оформления и простотой визуального восприятия содержимого сайта (в главе 6 вы узнаете все о форматировании текста, там же вы найдете подробное описание этих двух псевдоэлементов).

ПРИМЕЧАНИЕ

В последней версии CSS синтаксис для псевдоэлементов претерпел изменения. В CSS 2.1 псевдоэлементы начинались с одинарного двоеточия:

```
:first-letter
```

В версии CSS3, чтобы отличить такие псевдоклассы, как `:hover`, от псевдоэлементов, добавлено еще одно двоеточие. Поэтому `:first-letter` и `:first-line` стали теперь выглядеть как `::first-letter` и `::first-line`. К счастью, чтобы сохранить поддержку более старых сайтов, браузеры продолжают поддерживать версию псевдоэлементов с одинарным двоеточием. Это хорошо, потому что Internet Explorer 8 не поддерживает синтаксис с двумя двоеточиями, поэтому пока можно остановиться на одинарном, поскольку все остальные браузеры используют их как и раньше.

Дополнительные псевдоклассы и псевдоэлементы

В спецификации по языку CSS определены еще несколько мощных псевдоклассов и псевдоэлементов. Эти селекторы очень хорошо поддерживаются во всех браузерах, кроме устаревших.

:focus

Псевдокласс `:focus` подобен `:hover`, с той лишь разницей, что `:hover` применяется, когда посетитель помещает указатель мыши поверх ссылки, а `:focus` — когда нажимает клавишу **Tab** или щелкает кнопкой мыши на текстовом поле (то есть требуется акцентировать внимание посетителя на конкретном (текущем) элементе веб-страницы). Щелчок на заполняемой форме программисты называют *фокусировкой*. Это единственный способ для дизайнера веб-страницы узнать, на чем сосредоточено внимание посетителя, с каким элементом страницы он имеет дело.

Селектор `:focus` полезен в основном для обеспечения обратной связи с посетителями сайта. Например, для смены цвета фона заполняемого поля формы, чтобы указать, где именно нужно вводить данные (текстовые поля, поля ввода пароля, текстовые области — везде можно использовать `:focus`). Этот стиль задает светло-желтый цвет любому текстовому полю, на котором посетитель щелкает кнопкой мыши или к которому переходит нажатием клавиши `Tab`:

```
input:focus { background-color: #FFFFCC; }
```

Селектор `:focus` задает эффект стиля только на время, пока элемент находится в фокусе. Если посетитель переходит к другому полю или в другую позицию веб-страницы, то свойство CSS прекращает форматировать элемент страницы.

СОВЕТ

Информация о поддержке браузерами селекторов приведена на сайте caniuse.com.

:before

Псевдоэлемент `:before` выполняет такую функцию, которая не присуща ни одному другому селектору: он позволяет добавлять сообщение, предшествующее определенному элементу веб-страницы. Допустим, вы хотите поместить слово **ПОДСКАЗКА!** перед абзацами, чтобы визуально выделить их (по аналогии с названиями врезок в этой книге). Вместо многократного набора текста в HTML-коде вашей веб-страницы вы можете сделать это с помощью селектора `:before`. Кроме того, такое решение позволит уменьшить объем кода веб-страницы. Так, если вы решите изменить сообщение с **ПОДСКАЗКА!** на **ЭТО НУЖНО ЗНАТЬ!**, можно быстро отформатировать его на всех страницах сайта, один раз изменив текст в таблице стилей. Однако недостаток состоит в том, что сообщение невидимо для браузеров, которые не поддерживают CSS или псевдоэлемент `:before`.

Для работы псевдоэлемента нужно создать класс (скажем, с именем `.tip`) и применить его к абзацам, которым должно предшествовать данное сообщение, например `<p class="tip">`. Добавьте текст сообщения в таблицу стилей:

```
.tip:before {content: "ПОДСКАЗКА!" }
```

Всякий раз, когда браузеру встречается класс `.tip` внутри элемента `p`, он будет добавлять перед абзацем сообщение **ПОДСКАЗКА!**.

Текст, который добавляется этим селектором, называют *сгенерированным контентом*, поскольку браузер создает его на лету. В исходном коде HTML-страницы этой информации не содержится. Браузеры все время генерируют контент, например маркеры в маркированных списках или цифры в нумерованных. Для этого также можно использовать селектор `:before`.

Селектор `:before` поддерживается браузером Internet Explorer 8 и выше, а также другими основными браузерами (как и селектор `:after`).

:after

Селектор `:before` добавляет сгенерированное содержимое перед определенным элементом, а `:after` — после. Например, им вы можете пользоваться для добавления заключительных кавычек (") после процитированного материала.

ПРИМЕЧАНИЕ

Псевдоэлементы `:before` и `:after` схожи с `:first-line` и `:first-letter`. Как уже упоминалось, в CSS3 к псевдоэлементам добавилось двойное двоеточие, поэтому `:before` и `:after` в CSS3 теперь называются как `::before` и `::after`. К счастью, браузеры поддерживают и более старую нотацию, поэтому вы можете продолжать использовать `:before` и `:after`, что добавляет преимуществ при работе в Internet Explorer 8.

::selection

Этот селектор, появившийся в CSS3, ссылается на элементы, которые посетитель выбрал на странице. Например, когда посетитель, нажав и удерживая кнопку мыши над текстом, перетаскивает указатель мыши, браузер выделяет этот текст и посетитель сможет скопировать выделенный фрагмент. Обычно браузеры добавляют за текстом синий фон. Internet Explorer изменяет при этом цвет шрифта текста на белый. Но вы можете управлять цветом фона и текста, указав описываемый селектор. Например, если нужно сделать выбранный текст белым на фиолетовом фоне, можно добавить в таблицу CSS следующий стиль:

```
::selection {
  color: #FFFFFF;
  background-color: #993366;
}
```

С помощью данного селектора можно установить лишь свойства `color` и `background-color`, поэтому вы не сможете изменить кегль, шрифт, поля и внести другие визуальные изменения.

ПРИМЕЧАНИЕ

Версии с одинарным двоеточием для псевдоэлемента `selection` не существует, поэтому вы должны использовать двойное двоеточие. Иными словами, `::selection` работает, а `:selection` — нет.

Этот селектор работает в браузерах Internet Explorer 9, Opera, Chrome и Safari, но не поддерживается в Internet Explorer 8 или Firefox. Однако вы можете добавить поддержку для Firefox, дополнив имя селектора так называемым *вендорным префиксом*:

```
::-moz-selection {
  color: #FFFFFF;
  background-color: #993366;
}
```

Для обеспечения работоспособности в Firefox и других браузерах в таблице стилей нужно указывать оба стиля, которые можно просто расположить друг за другом. (Подробности, касающиеся вендорных префиксов и необходимости их использования, изложены во врезке в главе 7.)

Если вы действительно хотите кого-нибудь поразить, можно указать другой цвет фона для текста, выделенного внутри конкретного элемента. Например, чтобы сделать красным на розовом фоне текст, который находится только внутри абзацев, нужно добавить селектор элемента `p` перед кодом `::selection`:

```
p::selection {
  color: red;
  background-color: pink;
}
```

Селекторы атрибутов

Каскадные таблицы стилей обеспечивают возможность форматирования элементов на основе выборки любых содержащихся в них атрибутов. Например, вы хотите выделить рамкой важные изображения веб-страницы, при этом не форматировать логотип, кнопки и другие небольшие изображения, в коде которых также присутствует элемент `img`. Вы должны понимать, что если у всех рисунков есть описания с атрибутом `title`, то это способствует использованию *селектора атрибутов* для выделения из массы изображений только нужных.

С помощью селекторов атрибутов вы можете выбрать элементы с конкретными свойствами. Вот пример, в котором выделены все элементы `img` с атрибутом `title`:

```
img[title]
```

Первая часть селектора — имя элемента (`img`); атрибут указывается далее в квадратных скобках: `[title]`.

Каскадные таблицы стилей не ограничивают использование селекторов атрибутов именами элементов: вы можете комбинировать их с классами. Например, селектор `.photo[title]` выбирает все элементы класса `.photo` с HTML-атрибутом `title`.

Если необходима более детальная выборка, то есть возможность найти элементы, которые имеют не только определенный атрибут, но и нужное значение. Например, если вы хотите подсветить ссылки, указывающие на определенный URL-адрес, создайте следующий селектор атрибута:

```
a[href="http://www.cafesoylentgreen.com"]{
  color: green;
  font-weight: bold;
}
```

Уточнение селектора конкретным значением очень полезно при работе с заполняемыми формами. Многие элементы форм имеют одинаковые названия, даже если выглядят и функционируют по-разному. Будь то флажок, текстовое поле, кнопка отправки данных или какие-то другие элементы формы — все они содержат `input`. Значение атрибута `type` — вот что позволяет наделить тот или иной элемент формы соответствующими функциональными возможностями. Например, код `<input type="text">` создает текстовое поле, а `<input type="checkbox">` — флажок.

Чтобы выбрать только текстовые поля в форме веб-страницы, используйте следующее выражение:

```
input[type="text"]
```

Селектор атрибута очень разносторонний. Он не только позволяет находить элементы с определенным значением атрибута (например, все элементы формы со

значением `checkbox` атрибута `type`), но и выбирать элементы со значением атрибута, *начинающимся* с какого-либо значения, *заканчивающимся* им или *содержащим* его. Хотя эта возможность на первый взгляд может показаться излишней, на самом деле она достаточно полезная.

Представьте, что вы хотите создать стиль, который бы выделял внешние ссылки (ведущие за пределы вашего сайта), сообщая пользователю: «Ты покинешь этот сайт, если перейдешь по ссылке». Принимая во внимание то, что абсолютные ссылки внутри собственного сайта не используются, мы определяем, что любая внешняя ссылка будет начинаться со значения `http://` — первой части любой абсолютной ссылки.

Тогда селектор будет выглядеть так:

```
a[href^="http://"]
```

Символы `^=` означают «начинается на», так что вы можете использовать этот селектор для форматирования любой ссылки, начинающейся со значения `http://`. С его помощью вы легко форматируете ссылку, указывающую на `http://www.google.com` либо `http://www.piter.com`, то есть любую внешнюю ссылку.

ПРИМЕЧАНИЕ

Этот селектор не будет работать при использовании защищенного SSL-соединения, то есть когда ссылка начинается со значения `https://`. Чтобы создать стиль, учитывающий данную проблему, вам понадобится группа селекторов:

```
a[href^="http://"], a[href^="https://"]
```

Встречаются также ситуации, когда вам необходимо выбрать элемент с атрибутом, *заканчивающимся* определенным значением. И снова для этой задачи пригодны ссылки. Скажем, вы хотите добавить небольшой значок после ссылок, указывающих на PDF-файлы. Поскольку они имеют расширение PDF, вы знаете, что ссылка на эти документы будет заканчиваться также этими символами, например ``. Значит, чтобы выбрать только такие ссылки, нужен следующий селектор:

```
a[href$=".pdf"]
```

А стиль целиком будет выглядеть так:

```
a[href$=".pdf"] {
  background: url(doc_icon.png) no-repeat;
  padding-left: 15px;
};
```

Не беспокойтесь о том, что не знаете конкретные свойства этого стиля — вы прочтете о них далее в книге. Только обратите внимание на один интересный селектор: `$=` означает «заканчивается на». Вы можете использовать его для форматирования ссылок на документы Word (`[a href$=".doc"]`), фильмы (`[a href$=".mp4"]`) и т. д.

И наконец, вы можете выбирать элементы с атрибутами, содержащими конкретное значение. Например, вам нужно выделить фотографии сотрудников повсюду на сайте. Вы хотите, чтобы у всех фотографий был общий стиль, допустим зеленая рамка и серый фон. Один из способов сделать это — создать класс, например `.photo`,

и добавлять вручную атрибут класса к соответствующим элементам `img`. Однако, если вы задали для фотографий последовательные названия, это не самый быстрый метод.

Например, каждую из фотографий вы называли, используя при этом слово `photo`: `ivanov_photo`, `petrov_photo` и т. д. В каждом файле встречается слово `photo`, поэтому и атрибут `src` тега `` содержит это слово. Вы можете создать селектор специально для этих изображений:

```
img[src*="photo"]
```

Выражение переводится как «выберите все изображения повсюду, атрибут `src` которых содержит слово `photo`». Это простой и изящный способ форматирования фотографий сотрудников.

Дочерние селекторы

Подобно применению селекторов потомков, описанных ранее, в CSS можно форматировать вложенные элементы с помощью *дочернего* селектора, который использует дополнительный символ — угловую скобку (`>`) — для указания отношения между двумя элементами. Например, `body > h1` выбирает любой элемент `h1`, дочерний по отношению к `body`.

В отличие от селектора потомков, который применяется ко *всем* потомкам (то есть вложенным элементам), дочерний селектор позволяет определить конкретные дочерний и родительский элемент. На рис. 3.6 вы можете видеть два элемента `h2`. Использование селектора потомков `body h2` привело бы к выбору обоих, так как они являются вложенными по отношению к `body`. Но только второй — дочерний элемент `body`. Первый `h2` непосредственно вложен в `div`, который и является родительским. Поскольку у `h2` различные родительские элементы, то для того, чтобы добраться до каждого из них отдельно, можно использовать дочерний селектор. Для выбора только первого элемента `h2` используйте код `body > h2`, а для выбора второго — `div > h2`.

Для выбора дочерних элементов в CSS3 также включены некоторые весьма специфические псевдоклассы. Они позволяют точнее настроить селекторы под многие различные компоновки HTML-кода.

:first-child

Возвращаясь на время к аналогии с семейным деревом HTML, вспомним, что такое дочерний элемент: это любой элемент, непосредственно заключенный в другой элемент. (Например, на рис. 3.6 элементы `h1`, `div`, `h2` и `ul` являются дочерними по отношению к `body`.) Псевдоэлемент `:first-child` позволяет выбрать и форматировать только первый из них, независимо от того, сколько их вообще может быть.

Если нужно выбрать *первый* элемент `h1` на странице, показанной на рис. 3.6, создайте следующий селектор:

```
h1:first-child
```

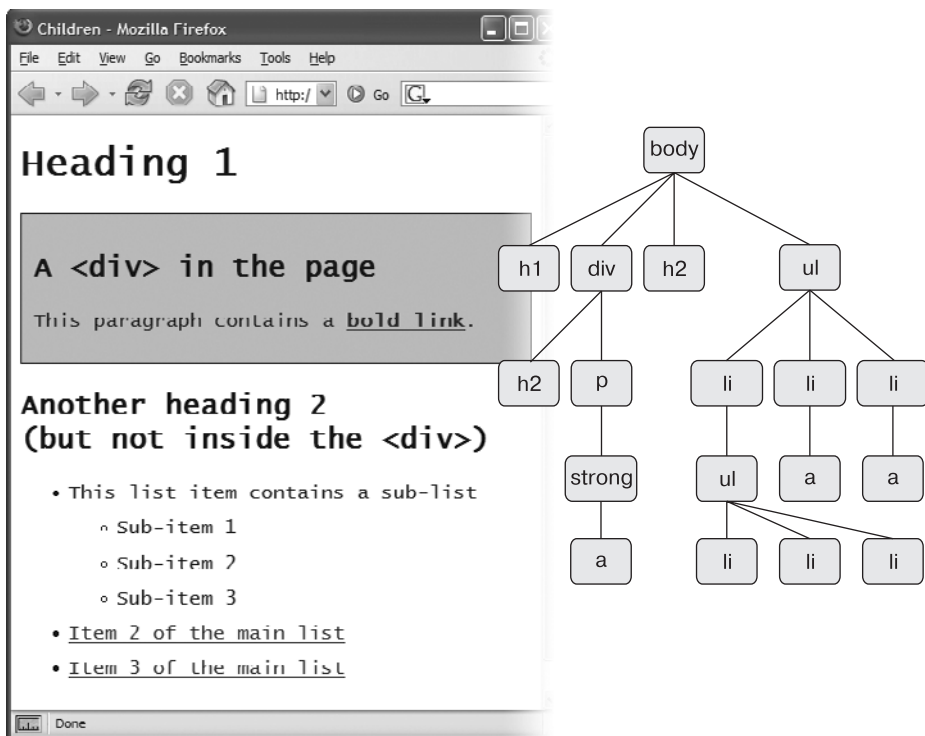


Рис. 3.6. Диаграмма в виде дерева (справа) показывает отношения между HTML-элементами (слева)

Этот селектор применяется к любому элементу `h1`, являющемуся первым дочерним элементом. На рис. 3.6 результат будет очевиден: там только один элемент `h1` и он является первым элементом на странице. Следовательно, он является дочерним для элемента `body`. Но `:first-child` может вызвать путаницу. Например, если вы измените элемент `h2` внутри `div`, показанного на рис. 3.6, на элемент `h1`, то `h1:first-child` выберет оба заголовка `h1`: тот, который непосредственно находится внутри элемента `body`, и тот `h1`, который находится внутри `div`-контейнера (поскольку этот заголовок `h1` является первым дочерним элементом `div`).

:last-child

Этот псевдоэлемент похож на рассмотренный ранее `:first-child`, но выбирает последний дочерний элемент. Например, для форматирования последнего элемента списка нужно воспользоваться селектором `li:last-child` (рис. 3.7).

:only-child

Существует также селектор для элемента, который является единственным дочерним для другого элемента. Допустим, в вашей таблице стилей есть такой стиль:

Child Selectors

li:first-child

one
two
three
four
five
six

li:last-child

one
two
three
four
five
six

li:nth-child(odd)

one
two
three
four
five
six

li:nth-child(even)

one
two
three
four
five
six

li:nth-child(2)

one
two
three
four
five
six

li:nth-child(5)

one
two
three
four
five
six

li:nth-child(3n)

one
two
three
four
five
six

li:nth-child(2n)

one
two
three
four
five
six

li:nth-child(3n+1)

one
two
three
four
five
six

li:nth-child(4n+2)

one
two
three
four
five
six

li:nth-child(n+3)

one
two
three
four
five
six

li:nth-child(-n+3)

one
two
three
four
five
six

Рис. 3.7. Многочисленные дочерние селекторы в CSS предоставляют вам различные способы выбора дочерних элементов. Эти селекторы хорошо подходят для выделения первого, последнего или чередующихся элементов списка

```
p:only-child {
  color: red;
}
```

Он указывает, что цвет текста должен быть красным, но только в том случае, если внутри элемента находится лишь один абзац. Например, если в вашем элементе `div` находится три абзаца, то стиль не будет применен, поскольку внутри элемента `div` находится три дочерних элемента. Тем не менее, если из элемента `div` вы удалите два абзаца, оставшийся абзац станет красным.

Все несколько запутанно, но следует помнить, что данный стиль работает только в том случае, когда конкретный элемент является единственным дочерним для другого элемента. Иными словами, недостаточно, чтобы элемент являлся единственным в своем роде. Если у него существует другой родственный элемент, то селектор не сработает. Если вы добавите элемент `ul` внутрь `div` вместе с `p`, то абзац больше не будет единственным дочерним объектом. Элемент `ul` будет таким же дочерним, поэтому селектор `p:only-child` не будет применен.

:nth-child

Этот комплексный селектор очень полезен. К примеру, с его помощью можно легко и просто форматировать каждую вторую строку в таблице, каждый третий элемент в списке или придать свой стиль любому сочетанию чередующихся дочерних элементов (см. рис. 3.7). Для определения того, какой из дочерних элементов нужно выбирать, этот селектор требует значение. Проще всего указать ключевое слово — либо `odd`, либо `even`, — позволяющее выбрать чередующиеся нечетные или четные дочерние элементы соответственно. Например, если нужно предоставить один фоновый цвет для каждой четной строки таблицы и другой фоновый цвет для каждой нечетной, можно создать два следующих правила:

```
tr:nth-child(odd) { background-color: #D9F0FF; }
tr:nth-child(even) { background-color: #FFFFFF; }
```

Теперь у вас есть действительно простой способ для окрашивания чередующихся строк таблицы (рис. 3.8). Но у псевдоэлемента `:nth-child()` в рукаве спрятано еще немало козырей.

Можно также выбрать определенный дочерний элемент, указав его номер. Например, если вы хотите выделить пятый элемент в списке, задайте цифру 5 в селекторе `:nth-child()`:

```
li:nth-child(5)
```

Этот стиль выделяет только один дочерний элемент. Если вы хотите выделить, скажем, каждый третий элемент в списке, используйте цифру (в данном примере это 3) и букву `n`:

```
li:nth-child(3n)
```

Здесь `n` — это множитель, поэтому запись `3n` означает каждый третий дочерний элемент, начиная с третьей строки (см. рис. 3.7).

Но как быть, если вы хотите выделить каждый третий дочерний элемент списка, начиная со второго дочернего элемента? Представим, что вам нужно выделить каждую третью ячейку таблицы (элемент `td`) внутри строки, начиная со второй ячейки таблицы (см. рис. 3.8). Для этого нужно применить следующий стиль:

```
td:nth-child(3n+2) { background-color:#900; }
```

Число перед `n` — множитель: запись `3n` означает каждый третий элемент, `4n` — каждый четвертый элемент и т. д. Символ `+` и следующее за ним число (`+2` в данном примере) указывают, с какого элемента нужно начинать выделение, следовательно, `+2` означает, что нужно начинать со второго дочернего элемента, а `+5` говорит о том, что нужно начинать выделение с пятого дочернего элемента. Таким образом, использование псевдоэлемента `:nth-child(5n+4)` приведет к выделению каждого пятого дочернего элемента, начиная с четвертого дочернего элемента.

Вы также можете использовать отрицательные значения `n`, что приведет к выбору всех предшествующих дочерних элементов *в обратном порядке*. Например, последний список на рис. 3.7 использует следующий селектор:

```
li:nth-child(-n+3)
```

Alternating Table Rows

Рис. 3.8. Простой способ окрашивания ячеек в таблице. Вы даже можете форматировать таблицу, выделяя чередующиеся столбцы, нацеливаясь на каждый второй элемент `td` в строке или, как в данном случае, на каждый третий столбец, начиная со второго

Он означает «начать с третьего элемента в списке и выделить каждый предшествующий ему элемент». Как можно увидеть, селектор `nth-child` сложный, но достаточно мощный инструмент, который позволяет выделять бесконечное разнообразие дочерних элементов.

Дочерние псевдоклассы

Каскадные таблицы стилей включают селектор, который работает во многом похоже на дочерний, рассмотренный в предыдущем разделе, но применяется к дочерним элементам с *HTML-элементом* определенного типа. Предположим, что вам нужно определенным образом отформатировать первый абзац на боковой панели, но только на тех страницах, где эта боковая панель начинается с элемента `h2`, и на других страницах, где она начинается с элемента `p`. Псевдоэлемент `:first-child` для выбора этого абзаца применять нельзя, потому что в некоторых случаях абзац является *вторым* дочерним элементом (который следует за `h2`). Тем не менее он всегда является первым абзацем (элементом `p`) на этой боковой панели, даже если перед ним идут какие-нибудь другие элементы, следовательно, его можно выбрать с помощью селектора `first-of-type`.

ПРИМЕЧАНИЕ

Псевдоклассы `:last-child`, `:first-of-type` и `:nth-child()` поддерживаются всеми современными браузерами, включая Internet Explorer 9 и выше. Но в Internet Explorer 8 они не работают.

`:first-of-type`

Селектор работает так же, как и `:first-child`, но применяется к дочернему элементу, имеющему определенный элемент. Предположим, что у вас есть боковая панель с классом `sidebar`. Для форматирования первого абзаца этой боковой панели используется следующий селектор:

```
.sidebar p:last-of-type
```

Обратите внимание на букву `p` в коде `p:first-of-type`. Она обозначает элемент, который вы собираетесь отформатировать.

:last-of-type

Селектор работает так же, как и `:last-child`, но применяется к последнему экземпляру элемента определенного типа. Например, если нужно на боковой панели `div` определенным образом отформатировать последний абзац, но вы не уверены, что за абзацем нет каких-либо других элементов (например, неупорядоченного списка, заголовка или рисунка). Этот стиль имеет следующий вид:

```
.sidebar p:last-of-type
```

ПРИМЕЧАНИЕ

Следует помнить, что указанные селекторы тегов также должны быть дочерними по отношению к конкретному элементу. Следовательно, код `p:first-of-type` означает «первый дочерний элемент, являющийся элементом абзаца».

:nth-of-type

Работает так же, как и `:nth-child()`, но применяется к чередующимся дочерним элементам, имеющим определенный элемент. Этот селектор может пригодиться при наличии больших абзацев текста, усеянных фотографиями. Элемент `img` является строчным, поэтому у вас может быть абзац `p` с несколькими изображениями `img` внутри. И если вам захочется чередовать появление изображений то слева, то справа, как показано на рис. 3.9, это можно сделать с помощью таких правил:

```
img:nth-of-type(odd) { float: left; }
img:nth-of-type(even) { float: right; }
```

Как видите, для `:nth-of-type()` используются такие же ключевые слова (`odd` или `even`) и формулы (например, $2n+1$), как и для `:nth-child()`.

Между прочим, `:nth-of-type()` можно также использовать для чередующихся строк таблицы:

```
tr:nth-of-type(odd) { background-color: #D9F0FF; }
tr:nth-of-type(even) { background-color: #FFFFFF; }
```

Что касается CSS-селекторов, то здесь всегда имеется несколько способов добраться до HTML-элемента. Обычно их набирается более пяти!

ЧАВО

Придание спискам привлекательного внешнего вида

Когда нужно использовать дочерние селекторы? Исходя из текста главы существует достаточно количество селекторов для выборки практически любого элемента веб-страницы. Так для чего нужны остальные селекторы?

На самом деле существуют некоторые сложности дизайна веб-страниц, где дочерние селекторы просто незаменимы и никакие другие не смогут с этим спра-

виться. Такая ситуация встречается на большинстве сайтов. В любом маркированном списке присутствует несколько пунктов — элементов списка (см. рис. 3.6). Здесь можно использовать дочерние селекторы, чтобы визуально упорядочить данные в категориях (пунктах) и подкатегориях (подпунктах). Вы форматируете элементы первого уровня списка одним способом, а второго — другим. Содержимое, представленное

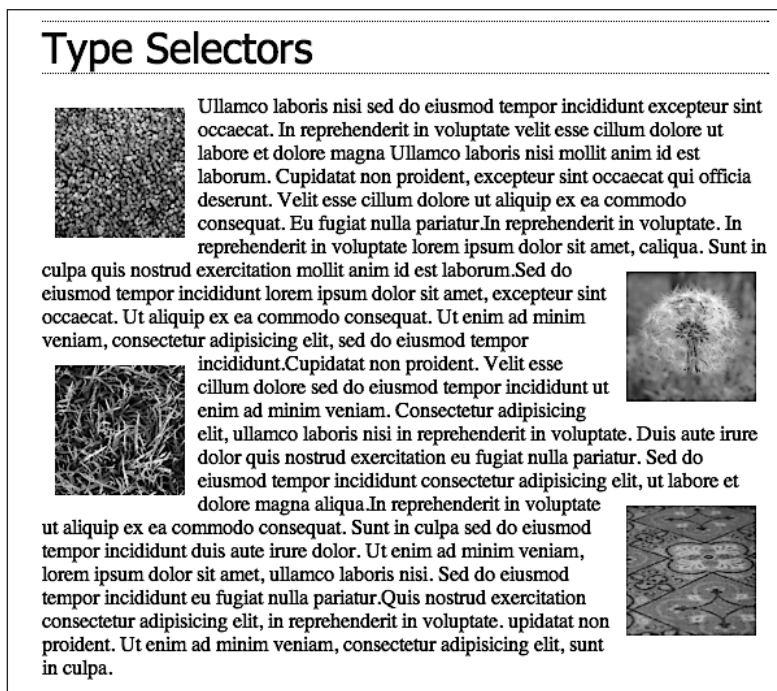


Рис. 3.9. Используя селектор `:nth-of-type()`, можно легко выбрать любые изображения внутри элемента, чередуя их расположение то справа, то слева

ЧАВО

таким способом, выглядит четко, профессионально и читабельно.

Сначала создайте класс для самого верхнего — внешнего — уровня вложенности элементов списка и назовите его, скажем, `.mainList`. Для первого уровня вы можете использовать шрифт `sans-serif`, имеющий немного больший размер по сравнению с основным текстом веб-страницы, возможно, в другом цвете. Последующие категории могут быть представлены Times для лучшего восприятия. При большом объеме текста форматирование каждого уровня подкатегорий с небольшим отличием позволяет посетителям веб-страниц визуально ориентироваться в материале.

Теперь примените класс `.mainList` к первому элементу `ul`: `<ul class="mainList">`. Затем используйте дочерний селектор (`ul.mainList > li`) для выбора элементов списка только первого уровня и придания пунктам необходимого форматирования. Данный

стиль будет применен только к элементу `li`, являющемуся дочерним по отношению к `ul` и принадлежащему к классу `.mainList`. Затем для задания стиля дочерним элементам `li` любых последующих вложенных элементов `ul` воспользуйтесь таким селектором: `ul.mainList > li > ul > li`. (Селекторы потомков, как `ul li`, в отличие от этого, выбирают элементы списка всех неупорядоченных списков на странице: и вложенных, и невложенных.)

Вам нужно будет обратить внимание и на понятие наследования, которое будет рассмотрено в следующей главе. Как правило, конкретные CSS-свойства, примененные к одному элементу, наследуются элементами, находящимися внутри него. Поэтому, если даже вы используете дочерний селектор, нацеленный на чей-либо дочерний элемент, свойства могут перейти на другие элементы внутри этого дочернего элемента. Один из способов избежать такого развития событий заключается в применении селектора `:not()`.

Родственные селекторы

Родительско-дочерние отношения — не единственная форма родственных связей в дереве HTML. Иногда требуется выбрать элемент, относящийся к группе родственных элементов одного уровня с общим родителем. Элемент, который следует сразу же за другим элементом, в HTML называется *смежным родственным элементом* того же уровня. На рис. 3.6 элемент `div` является смежным по отношению к `h1`, а элемент `p` — по отношению к `h2` и т. д.

Используя смежный родственный селектор, можно, к примеру, придать первому абзацу после каждого заголовка форматирование, отличное от следующих абзацев. Предположим, вы хотите удалить отступ, который автоматически появляется перед элементом `p`, чтобы между заголовком и абзацем не было промежутка. Или хотите придать абзацу, как небольшому вводному описанию, другой цвет и размер шрифта.

Смежный родственный селектор использует знак `+` для соединения одного элемента с другим. Поэтому, чтобы выбрать все первые абзацы, следующие за любым заголовком `h2`, используйте селектор `h2 + p` (пробелы необязательны, так что `h2+p` также будет прекрасно работать). Последний элемент в селекторе (в данном случае `p`) — элемент, который нужно отформатировать, но только при условии, что он следует сразу за смежным для него элементом `h2`.

Есть и другой родственный селектор, который называется *общим родственным сборным селектором*. Он обозначается знаком тильды (`~`) и означает следующее: «Нужно выбрать все родственные элементы этого типа». Например, если селектором `h2 + p` задается выбор отдельного абзаца `p`, который следует сразу за заголовком `h2`, то селектором `h2 ~ p` задается выбор *всех* элементов `p`, родственных (то есть находящихся на одном уровне) по отношению к заголовку `h2`. Честно говоря, вы можете так и не найти этому селектору достойного применения, но в CSS весьма разнообразный синтаксис.

Селектор `:target`

Селектор `:target()` забавен. С его помощью можно создавать на самом деле интересные эффекты, например применять стиль к элементу страницы *после* того, как будет выполнен щелчок кнопкой мыши на другом элементе. Этот элемент интерактивности обычно выполняется с помощью сценариев JavaScript. Селектор зависит от использования конкретных идентификаторов, как это показано во врезке «HTML-элементы `div` и `span`» в разделе «Идентификаторы: отдельные элементы веб-страницы» этой главы. Идентификаторы используются для связи с определенным местом на странице.

Например, вы работаете с веб-страницей `index.html`. На ней находится элемент `div` с идентификатором `signupForm`. Если на этой странице (или на другой) расположена ссылка, которая указывает на объект `index.html#signupForm`, то после щелчка кнопкой мыши на ней браузер перейдет к этому элементу `div`. Если объект `div` находится в нижней части очень длинной страницы, то браузер прокрутит

страницу, отобразив его. Такие селекторы иногда называют *внутренними ссылками* и используют в качестве предметных указателей на веб-страницах. Щелкнув на такой ссылке, посетитель перейдет к области страницы, в которой встречается искомое слово.

Но вы не должны использовать эту функцию для перехода к другой области веб-страницы. Всякий раз, когда в URL в адресной строке браузера встречается символ # и следующий за ним идентификатор, элемент с указанным идентификатором становится селектором `target`. Поэтому вы можете применить конкретный стиль к элементу только в том случае, когда его идентификатор присутствует в URL-адресе.

Ниже приведен пример работы селектора `:target`. Представьте, что следующий код находится в верхней части веб-страницы:

```
<button>
  <a href="#signupForm">Подпишитесь на нашу рассылку</a>
</button>
<form id="signupForm">
  <label for="email">Укажите свой адрес электронной почты</label>
  <input type="email" id="email">
  <input class="btn" type="submit" value="Подписаться">
</form>
```

Когда посетитель сайта щелкнет кнопкой мыши на ссылке (элемент `a`), форма станет целевой. Другими словами, вы можете использовать один стиль для формы в ее обычном состоянии и другой после того, как посетитель выполнит щелчок на ссылке. Например, вы могли бы сначала скрыть форму (о свойствах CSS, отвечающих за отображение объектов, вы прочитаете в разделе «Принципы работы свойств позиционирования» главы 14) с помощью такого правила:

```
#signupForm {
  display: none;
}
```

Оно скрывает форму, поэтому, когда страница будет загружена, посетитель не увидит ее. Но, когда он щелкнет на ссылке `Подпишитесь на нашу рассылку`, форма станет целевой и вы сможете отформатировать ее с помощью следующего кода:

```
#signupForm:target {
  display: block;
}
```

Другими словами, если в адресной строке браузера отображается только URL, например `index.html`, то браузер использует первое правило, скрывая форму. Но если адрес выглядит, к примеру, следующим образом: `index.html#signupForm`, то браузер использует селектор `target` и форма отображается.

ПРИМЕЧАНИЕ

В галерее по адресу tinyurl.com/ltqzifu представлены поистине замечательные примеры использования селектора `:target`.

Селектор `:not()`

Селектор `:not()`, также известный как *псевдокласс отрицания*, позволяет выбрать что-либо отличное от другого. Например, можно применить класс к абзацу — `<p class="classy">` — и создать селектор, позволяющий отформатировать этот абзац:

```
.classy { color: red; }
```

А что делать, если понадобится выбрать все абзацы, *за исключением* тех, которым присвоен класс `classy`? Именно здесь пригодится селектор `:not()`. Чтобы указать, что *не* нужно выбирать, селектор помещается в круглые скобки. Например:

```
p:not(.classy) { color: blue; }
```

Этот стиль форматирует текст синим цветом во всех абзацах, к которым не применен класс `classy`.

Селектор `:not()` может быть полезен при использовании селекторов атрибутов. Например, ранее было показано, что селектор атрибутов можно использовать, чтобы выбрать все ссылки, указывающие за пределы вашего сайта:

```
a[href^="http://"]
```

Как вы уже, наверное, заметили, этот селектор не выбирает конкретно все ссылки, указывающие за пределы вашего сайта, он просто выбирает все ссылки, использующие абсолютные URL-адреса, то есть начинающиеся со значения `http://`. Для многих сайтов это одно и то же, поскольку они используют для указания на другие страницы этого же сайта ссылки относительно документа или главной страницы сайта, а для указания ссылок на другие сайты применяются абсолютные URL. Но в некоторых случаях абсолютные URL-адреса могут использоваться для указания на страницу внутри вашего сайта.

Например, некоторые системы управления контентом (в частности, WordPress) используют для указания на публикации блогов внутри сайта абсолютные URL-адреса. В этом случае, если нужно придать стиль ссылкам, которые ведут за пределы вашего сайта, следует усовершенствовать селектор атрибута, задействовав также селектор `:not()`. Предположим, доменное имя вашего сайта `mysite.com`. Для выбора ссылки, указывающей за пределы вашего сайта, нужно выбрать все абсолютные ссылки, *не* указывающие на домен `mysite.com`. Вот как это можно сделать:

```
a[href^="http://"]:not([href^="http://mysite.com"])
```

Если перевести на нормальный язык, этот селектор говорит: «Нужно выбрать все ссылки, значение атрибута `href` которых начинается с текста `http://`, но не те, которые начинаются с `http://mysite.com`». Если вспомнить предыдущий материал, в селекторе атрибута символы `^=` означают директиву «начинается с». То же самое можно написать еще короче:

```
a[href^="http://"]:not([href*="mysite.com"])
```

В селекторе атрибута символы `*=` означают правило «содержит», тем самым любой абсолютный URL-адрес, содержащий значение `mysite.com`, будет исключен. В том числе адреса `http://www.mysite.com` и `http://mysite.com`.

В отношении селектора `:not()` действуют несколько ограничений.

- С селектором `:not()` можно использовать только *простые селекторы*. Другими словами, можно применять селекторы тегов (такие как `html` или `p`), универсальный селектор (`*`), классы (например, `.footer`), идентификаторы (например, `#banner`) или псевдоклассы (`:hover`, `:checked`, `:first-child` и т. д.). Таким образом, все следующие селекторы можно считать правильными:

```
.footnote:not(div)
img:not(.portrait)
div:not(#banner)
li:not(:first-child)
```

- Нельзя использовать селекторы потомков (такие как `div p a`), псевдоэлементы (такие как `::first-line`), групповые селекторы или комбинации (такие как родственный смежный селектор `h2 + p`).
- Нельзя в одной строке указывать несколько селекторов `:not()`. Например, следующий код будет неправильным:

```
a[href^="http://"]:not([href*="google.com"]):not([href="yahoo.com"])
```

Другими словами, значение `:not()` в селекторе можно использовать только один раз.

Практикум: использование селекторов

В оставшейся части этой главы вы потренируетесь в создании разных селекторов и увидите, как они влияют на дизайн веб-страницы. Практикум начнем с представления основных типов, а затем перейдем к более современным стилям.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 03.

ВОЗМУЩЕННОЕ ЧАВО

Используйте внутренние таблицы стилей

Почему в этом практикуме мы пользуемся внутренними таблицами стилей? Ведь в главе 2 книги рекомендуется применять внешние!

Да, внешние каскадные таблицы стилей используются для создания быстро загружаемых, «производительных» сайтов. Однако внутренние таблицы стилей упрощают разработку одиночных веб-страниц, таких, как в этом практикуме. В данном случае гораздо удобнее работать с одним файлом, вместо того чтобы переключаться между файлом внешней таблицы стилей и веб-страницей.

Кроме того, вы можете пользоваться предварительным просмотром результатов своей работы без постоянного обновления кэша браузера.

Ввиду вышесказанного я рекомендую использовать внешние таблицы стилей для сайтов. Если вы собираетесь использовать стили, созданные в этом уроке, в дальнейшем, а не только в учебных целях — на здоровье. Но в рамках текущего упражнения, чтобы выполнять задания быстро и легко, вы будете пользоваться одиночным HTML-файлом и внутренней таблицей стилей.

1. Откройте файл `selector_basics.html`, расположенный в папке 03, в редакторе HTML-кода.

Страница состоит из основных HTML-элементов (рис. 3.10). В этом практикуме мы попытаемся придать ей изящный вид. Сначала вы свяжете страницу со шрифтом Google, который уже использовали в предыдущей главе.

2. Введите следующий код в пустой строке сразу после элемента `</title>`:

```
<link href='http://fonts.googleapis.com/css?family=Kurale' rel='stylesheet'>
```

Этот код связывается с внешней таблицей стилей, размещенной на сервере Google. Она загружает шрифт Kurale и тем самым дает вам возможность использовать его на своей странице (подробнее об использовании веб-шрифтов читайте в разделе «Использование веб-шрифтов» главы 6). Теперь необходимо добавить внутреннюю таблицу стилей.

3. Сразу после добавленного на предыдущем шаге элемента `link` нажмите клавишу **Enter** для перехода на новую строку и введите тег `<style>`. Дважды нажмите клавишу **Enter** и введите `</ >`.

Эти открывающий и закрывающий теги элемента `style` нужны для размещения внутренней таблицы стилей. Очень полезно набирать их сразу парой, чтобы случайно не забыть о закрывающем теге. Теги сообщают браузеру, что между ними находятся команды языка CSS. HTML-код теперь должен выглядеть следующим образом (код, который вы только что добавили, выделен полужирным шрифтом):

```
<title>Селекторы</title>
<link href='http://fonts.googleapis.com/css?family=Kurale' rel='stylesheet'>
<style>

</style>
```

Теперь нужно добавить селектор тега (если вы выполнили все упражнения практикума в прошлой главе, то уже умеете это делать).

4. Перейдите к строке между открывающим и закрывающим тегами элемента `style`, а затем наберите код `body {`. Дважды нажмите клавишу **Enter** и введите `}`.

Как я уже отмечал, желательно указывать закрывающую скобку сразу же, как только вы добавили открывающую. Чтобы создать селектор тега, просто укажите имя HTML-элемента, который следует отформатировать. Он будет применен ко всем абзацам текста, заключенным в элемент `body`. Теперь необходимо изменить фоновый цвет и поле вокруг страницы.

5. Щелкните кнопкой мыши между скобками `{` и `}` стиля `body` и добавьте три правила форматирования — цвет, отступ и поле:

```
body {
  background-color: rgb(50,122,167);
  padding: 0 20px 20px 20px;
  margin: 0;
}
```

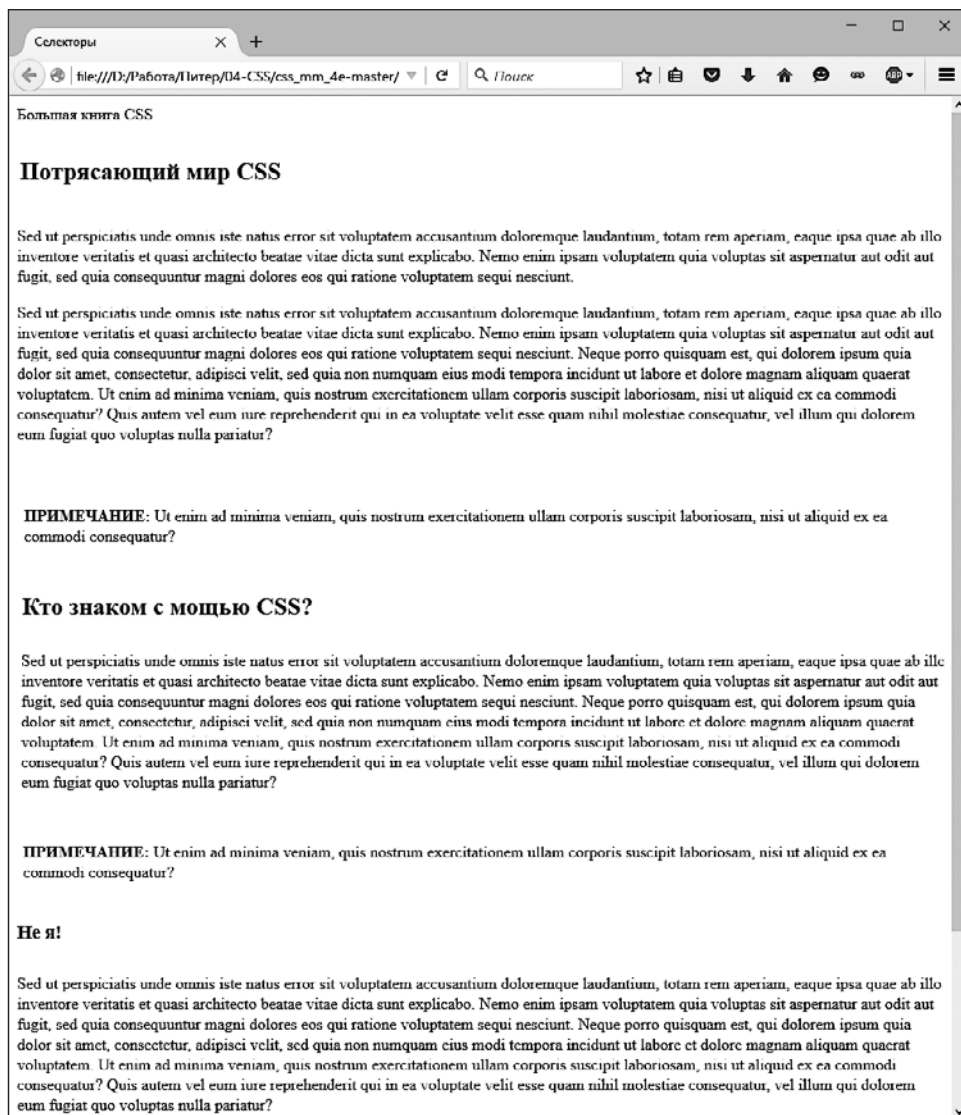


Рис. 3.10. Простой HTML-текст смотрится в браузере чересчур аскетично. Но с помощью каскадных таблиц стилей вы можете из серой страницы, показанной на этом рисунке, получить потрясающе красивую (рис. 3.11), выполнив три десятка простых шагов

Нажимайте клавишу **Enter**, чтобы поместить каждое CSS-свойство на отдельную строку. Кроме того, полезно сделать отступы клавишей **Tab**, чтобы улучшить визуальное восприятие кода CSS (некоторые разработчики вместо табуляции используют два пробела, а вы можете выбрать то, что вам больше нравится).

Свойство цвета фона, которое мы только что изменили, обозначается как `rgb()` — это один из способов указать значения красного, зеленого и синего цветов. В дан-

ном случае цвет фона темно-синий. Он затрудняет чтение текста, поэтому необходимо изменить цвет текста элементов абзаца.

ПРИМЕЧАНИЕ

Если вы новичок в веб-дизайне, то имена свойств и их значения вам пока незнакомы. Так что просто набирайте их в том виде, как показано в практических шагах. Обо всех этих свойствах вы узнаете в главе 6.

6. Добавьте другой стиль под только что созданным стилем `body`.

```
p {  
  color: rgba(255,255,255,.6);  
  font-size: 1em;  
  font-family: "Kurale", Arial, Helvetica, sans-serif;  
}
```

С помощью трех свойств CSS мы изменили форматирование всех абзацев (всех элементов `p`) — определили цвет, размер и шрифт текста. На этот раз, чтобы указать цвет, мы использовали значение `rgba()`. Дополнительная буква `a` в свойстве `rgb` позволяет создать частично прозрачный цвет. В данном примере для абзаца мы использовали белый цвет (его значения — `255, 255, 255`) с непрозрачностью 60 % (значение `.6`). Сквозь текст проступает синий цвет, поэтому он кажется светло-синим.

Самое время посмотреть, что у нас получилось.

7. Откройте страницу в браузере для предварительного просмотра.

До изменения настроек текст веб-страницы отображался шрифтом с засечками (таким как `Times New Roman`). Теперь, если каскадные таблицы стилей функционируют должным образом, вы увидите семь абзацев, для которых задан светло-синий цвет текста и шрифт `Kurale`.

Создание группового селектора

Нередко бывает так, что несколько различных элементов веб-страницы должны иметь одинаковый внешний вид. Вероятно, и вы хотите, чтобы все заголовки отображались шрифтом одного вида и цвета. Вместо того чтобы создавать отдельные стили и дублировать одни и те же атрибуты и параметры для каждого элемента `h1`, `h2` и т. д., вы можете собрать и сгруппировать несколько элементов в единственный селектор.

1. Вернитесь к своему HTML-редактору с открытым файлом `selector_basics.html`.

Сейчас мы добавим новый стиль сразу после только что созданного стиля элемента `p`.

2. Щелкните кнопкой мыши после закрывающей фигурной скобки селектора тега `p`, нажмите клавишу `Enter` для начала новой строки и наберите код:

```
h1, h2, h3 {  
  
}
```

Как описано в этой главе ранее, групповой селектор — это просто список. Данный стиль создает одинаковое форматирование элементов h1, h2 и h3 веб-страницы.

- Щелкните на пустой строке между открывающей { и закрывающей } фигурными скобками и добавьте пять CSS-свойств:

```
color: rgb(255,255,255);
font-family: Arial, "Palatino Linotype", Times, serif;
border-bottom: 2px solid rgb(87,185,178);
padding-top: 10px;
padding-bottom: 5px;
```

Здесь вы задаете цвет и тип шрифта для заголовков, добавляете линию границы под заголовками, устанавливаете отступы снизу и сверху, используя свойство padding. Оно добавляет дополнительное пространство от краев элемента без воздействия на фон или рамку, то есть вы добавляете немного пространства под заголовком и между нижней границей текста и рамкой под ним.

- Сохраните файл и просмотрите его работу в браузере. Заголовок h1 в начале веб-страницы и заголовки h2 и h3 ниже на странице имеют одинаковое начертание и цвет шрифта, а также зеленую рамку вверху (см. рис. 3.11). Элемент h1 выглядит немного мелковатым, но мы легко можем увеличить его.
- Вернитесь к файлу selector_basics.html в текстовом редакторе. Под только что созданным групповым селектором добавьте еще один стиль:

```
h1 {
  font-size: 2em;
}
```

Этот стиль увеличит размер шрифта. Em — это размер шрифта, который используется в браузерах по умолчанию, поэтому запись 2em означает двойной стандартный размер шрифта. Обратите внимание, как удобно применять сразу несколько стилей к одному элементу: правило h1, h2, h3 и h1 в данном случае. В этом примере групповой селектор и новый селектор тега применяются к элементу h1. Такой процесс в языке CSS называется *каскадом*. Подробнее об этом вы прочитаете в главе 5.

Создание и применение идентификаторов

Идентификаторы предназначены для изменения стиля определенного элемента. Вы создаете стиль и добавляете атрибут id к открывающему HTML-тегу на странице, а затем применяете свойства созданного стиля к этому одиночному элементу. Часто идентификаторы используются для определения элементов формы, создания ссылок на страницах (см. врезку «HTML-элементы div и span» в данной главе), а также помогают применять JavaScript-сценарии для управления элементами на странице. Хотя многие веб-дизайнеры в настоящее время стараются не использовать идентификаторы (почему, вы узнаете в разделе «Управление каскадностью» главы 5), желательно знать, как с ними работать.

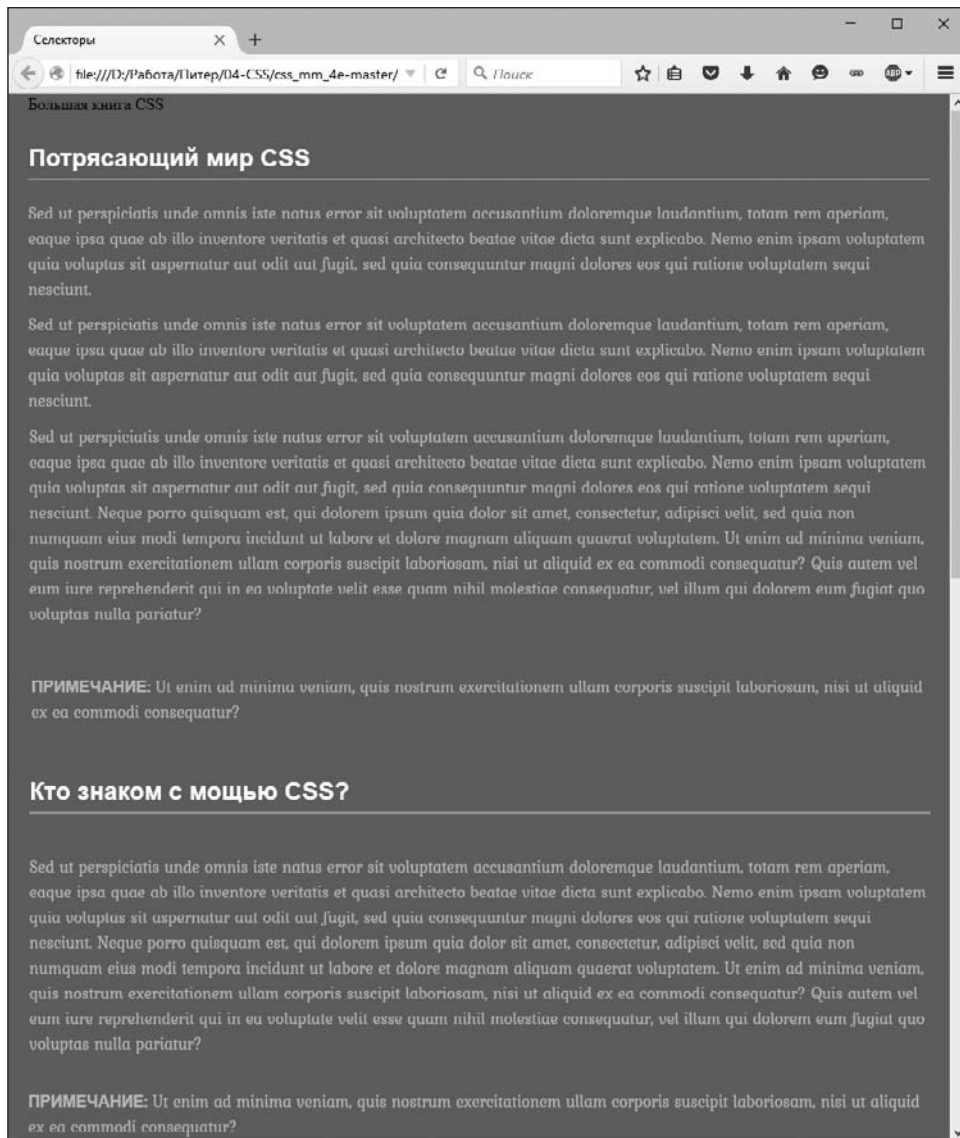


Рис. 3.11. Простой селектор тега может коренным образом преобразовать внешний вид всех входящих элементов, выполнив форматирование текста веб-страницы. В данном случае групповой селектор делает даже больше, изменяя формат каждого экземпляра заголовков трех различных уровней

В текущем примере мы создадим стиль, который определяет вид текста *Большая книга CSS* (CSS: The Missing Manual) в самом верху страницы. Этот текст играет роль логотипа, и вы создадите специальный идентификатор для его форматирования.

1. Вернитесь к HTML-редактору с открытым файлом `selector_basics.html`.
Добавим после последнего созданного класса `h1` новый стиль.

- Щелкните кнопкой мыши после закрывающей фигурной скобки предыдущего стиля, нажмите клавишу **Enter** для создания новой строки и введите код `#logo {`. Идентификаторы всегда начинаются с символа решетки (**#**). Имя стиля указывает на тип элемента страницы, который является логотипом сайта.

- Нажмите клавишу **Enter** еще раз и введите следующий код:

```
font-family: Baskerville, Palatino, sans-serif;
font-size: 2em;
color: rgba(255,255,255,.8);
font-style: italic;
text-align: center;
margin-bottom: 30px;
background-color: rgb(191,91,116);
border-radius: 0 0 10px 10px;
padding: 10px;
```

Этот код похож на длинный список свойств, но он лишь устанавливает некоторые свойства шрифта, фоновый цвет страницы и добавляет отступы для текста логотипа.

- Завершите определение стиля, введя закрывающую фигурную скобку. Код стиля полностью должен выглядеть так:

```
#logo {
  font-family: Baskerville, Palatino, sans-serif;
  font-size: 2em;
  color: rgba(255,255,255,.8);
  font-style: italic;
  text-align: center;
  margin-bottom: 30px;
  background-color: rgb(191,91,116);
  border-radius: 0 0 10px 10px;
  padding: 10px;
}
```

Если вы сохраните файл и просмотрите его в браузере, то не увидите никаких изменений. Это обусловлено тем, что данный стиль не делает *ничего*, пока вы его не примените. Для этого нужно добавить атрибут `id` к HTML-коду веб-страницы, обозначая фрагмент, который следует отформатировать.

- Найдите на веб-странице открывающий тег `<div>` с текстом Большая книга CSS. Он расположен после тега `<article>`. Добавьте в тег `<div>` код `id="logo"` следующим образом:

```
<div id="logo">
  Большая книга CSS
</div>
```

Теперь элемент `div` будет отформатирован в соответствии со стилями `#logo`. Как это часто случается при работе с CSS, существует много способов добиться одного и того же результата. Вы могли бы использовать класс и применить его к элементу `div`. Но в данном случае вы используете идентификатор, по-

сколькx назначение стиля — идентификация логотипа на странице — соответствует предназначению идентификаторов.

6. Сохраните страницу и просмотрите ее в браузере.

Теперь текст Большая книга CSS (CSS: The Missing Manual) находится в небольшом прямоугольнике в верхней части страницы, располагается по центру и выделен цветом (рис. 3.12).

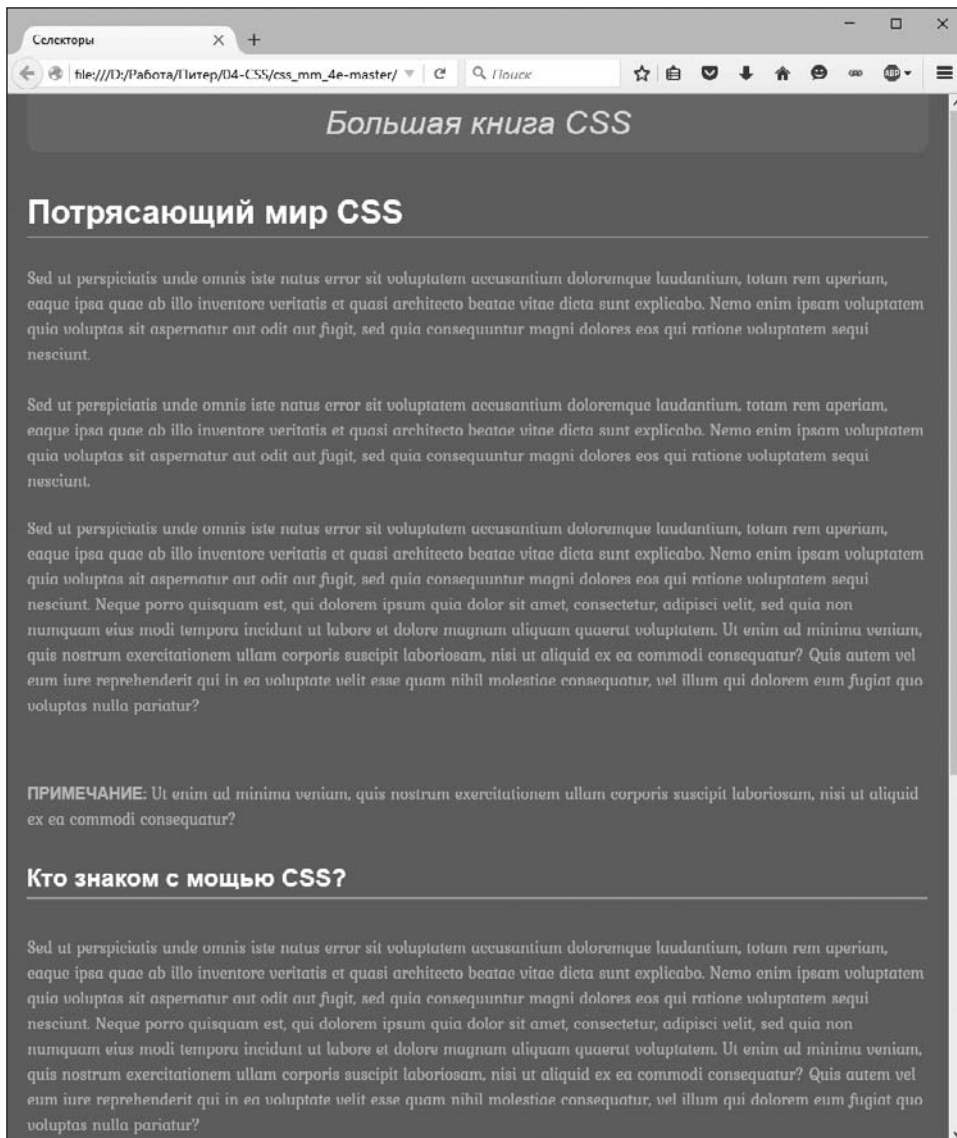


Рис. 3.12. Идентификаторы не единственный способ форматирования отдельных элементов, таких как логотип в верхней части страницы

Создание и применение классов

Селекторы тегов выполняют свои функции быстро и эффективно, но они, можно сказать, совсем неразборчивы в деталях. Что же делать, если вы хотите отформатировать один элемент р на веб-странице иным способом, чем все остальные такие элементы? Решение проблемы — использование классов.

1. Вернитесь к текстовому редактору с открытым файлом `selector_basics.html`. Добавьте вслед за последним созданным стилем еще один.
2. Щелкните кнопкой мыши после закрывающей фигурной скобки селектора `#logo`, нажмите клавишу **Enter** и введите код:

```
.note {  
  
}
```

Имя `note` (примечание) для стиля выбрано не случайно. Оно соответствует его функциям: стиль выделяет абзацы, содержащие дополнительные примечания для посетителей вашего сайта. Создав класс один раз, вы можете применить его ко всем примечаниям веб-страницы (сайта), например к третьему абзацу.

3. Щелкните на пустой строке между открывающей `{` и закрывающей `}` фигурными скобками и добавьте к стилю следующий перечень свойств:

```
color: black;  
border: 2px solid white;  
background-color: rgb(69,189,102);  
margin-top: 25px;  
margin-bottom: 35px;  
padding: 20px;
```

Обратите внимание, что для определения цвета шрифта и границ вы не использовали параметр `rgb()`. В языке CSS существует несколько способов определения цвета, включая ключевые слова, такие как `white` (белый), `black` (черный) или `orange` (оранжевый). Вы узнаете об этом подробнее в разделе «Форматирование текста цветом» главы 6.

Когда вы просмотрите эту веб-страницу, вы не увидите изменений. Как и в случае с идентификаторами, классы не возымеют никакого эффекта на веб-странице, пока стиль не будет применен к HTML-коду.

4. В HTML-коде веб-страницы найдите абзац `p`, текст которого начинается со слова **ПРИМЕЧАНИЕ**, окруженного тегами элемента `strong`.

Чтобы применить класс к этому элементу, добавьте атрибут `class`, за которым должно следовать имя класса, в данном случае `note`.

5. Щелкните кнопкой мыши сразу за именем открывающего тега `p`, нажмите клавишу **Пробел**, а затем введите код `class="note"`. HTML-код теперь должен иметь такой вид (только что набранный код отмечен полужирным шрифтом):

```
<p class="note"><strong>ПРИМЕЧАНИЕ:</strong>
```

Убедитесь, что *не* ввели атрибут с точкой: `class=" .note"`. Точка требуется только во время определения имени класса в таблице стилей; в HTML-коде она не нуж-

на. Повторите этот шаг для второго абзаца (он расположен перед элементом h3 с текстом Не я!).

ПРИМЕЧАНИЕ

Несмотря на имя, которое вы присвоили классу, вы можете применить его к любым другим элементам, а не только к p. Если данное форматирование относится, например, к заголовку h2, то HTML-код должен выглядеть следующим образом:

```
<h2 class="note">
```

6. Сохраните файл и просмотрите веб-страницу в браузере.
Абзац с примечанием красиво подсвечен на странице (рис. 3.13).

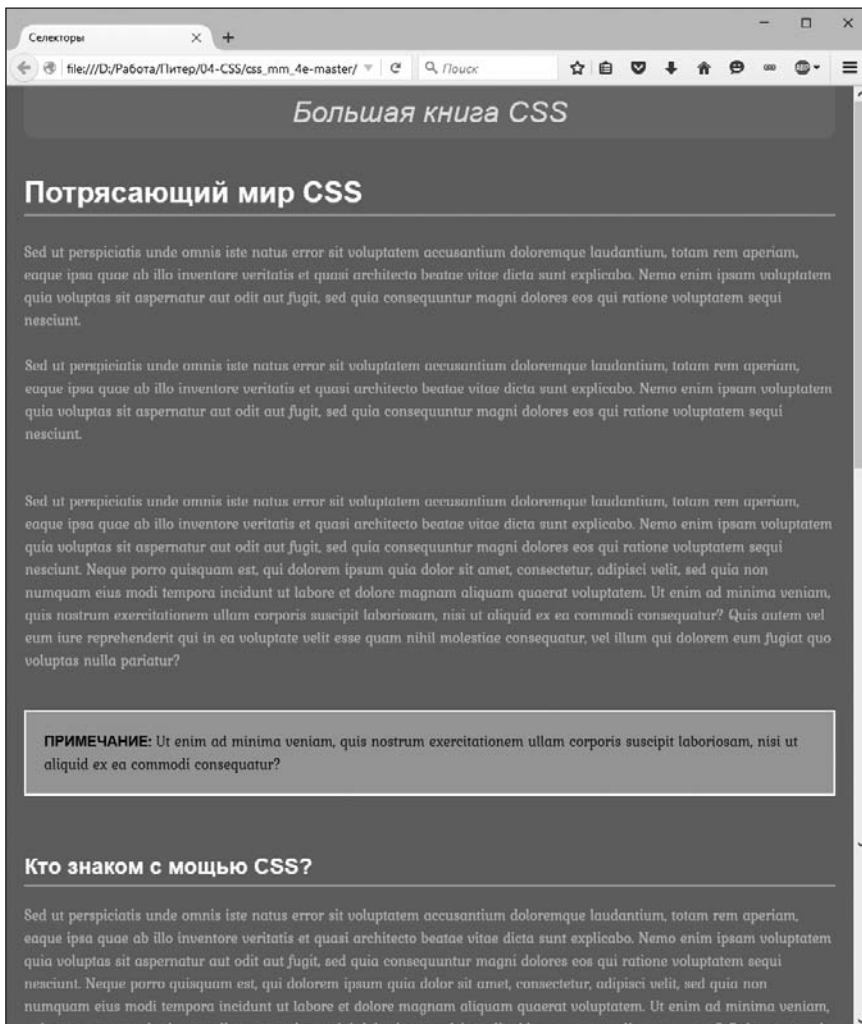


Рис. 3.13. С помощью классов вы можете выполнить ювелирное форматирование элементов веб-страницы

ПРИМЕЧАНИЕ

Если ваша веб-страница не похожа на изображенную на рис. 3.13, возможно, вы указали имя какого-либо свойства или его значение с ошибкой. Проверьте код по шагам. Кроме того, удостоверьтесь в том, чтобы каждая пара «свойство: значение» была завершена точкой с запятой и в самом конце определения стиля присутствовала закрывающая фигурная скобка. Если ваш стиль не работает должным образом, то, скорее всего, в нем не хватает именно этих символов. Это самая распространенная ошибка.

Создание селекторов потомков

На странице `selectors_basics.html` вы применили класс `note` к двум абзацам. Каждый из них начинается словом **Примечание:**, выделенным полужирным начертанием (на самом деле это слово находится внутри элемента `strong`). Но что делать, если вы хотите отформатировать эти слова еще и ярко-оранжевым цветом? Вы могли бы создать стиль для элемента `strong`, но он затронет все эти элементы на странице, в то время как вы хотите изменить только те, которые находятся внутри примечаний. Одним из решений было бы создание класса, например `.noteText`, и применение его к каждому из элементов `strong` внутри примечаний. Но вы наверняка забудете применить класс, если у вас много таких страниц с примечаниями.

Лучший способ решить эту проблему — создать селектор потомков, который относится только к нужным нам элементам `strong`. К счастью, сделать это совсем не сложно.

1. Вернитесь к HTML-редактору и файлу `selector_basics.html`. Создайте новую строку для указания стиля с селектором потомков.

Щелкните кнопкой мыши после закрывающей фигурной скобки стиля `.note` и нажмите клавишу `Enter`.

2. Введите код `.note strong {`.

Последнее слово селектора — `strong` — это и есть элемент, который вы хотите отформатировать. При этом стиль отформатирует элемент `strong` только в том случае, если он расположен внутри другого элемента, к которому применен класс `.note`. Стиль не возымеет никакого эффекта на элементы `strong`, находящиеся, например, в тексте других абзацев, в списках или заголовках первого уровня.

3. Нажмите клавишу `Enter`, введите код `color: #white;`, затем нажмите клавишу `Enter` снова, чтобы создать еще одну новую строку. Закончите стиль символом закрывающей фигурной скобки.

Конечный вариант стиля должен иметь следующий вид:

```
.note strong {  
  color: white;  
}
```

4. Сохраните HTML-файл и просмотрите страницу в браузере.

Слово **Примечание:** должно быть окрашено в оранжевый цвет в каждом из примечаний на странице.

Селекторы потомков — одно из самых мощных средств языка CSS. Профессиональные веб-дизайнеры достаточно интенсивно используют их для целенаправленного форматирования отдельных элементов, при этом не засоряя HTML-код классами. В книге они применяются повсеместно, поэтому у вас будет возможность получить о них более полное представление.

Последние штрихи

Текст на этой странице расширяется, чтобы заполнить окно браузера при изменении его размера. Чтобы увидеть данный эффект, откройте страницу и измените размер окна браузера. Обратите внимание, что по мере растягивания окна строки текста становятся шире. Если ваш монитор достаточно велик, то вы заметите, что при достижении определенной ширины строки текста слишком длинные, чтобы читать их с комфортом. К счастью, вы можете ограничить ширину контента страницы, чтобы она не превышала определенную величину.

1. Вернитесь к HTML-редактору с открытым файлом `selector_basics.html`. Создайте новую строку для нового стиля.

Если вы только что завершили предыдущие шаги, щелкните кнопкой мыши сразу после закрывающей скобки стиля `.note` и затем нажмите клавишу `Enter`.

2. Добавьте еще один стиль:

```
article {  
    max-width: 760px;  
}
```

Это еще один тип селектора. Он применяет HTML5-элемент `article`, который используется чтобы придать контенту вид записи в блоге, как показано на этой странице.

Значение параметра `max-width` определяет максимальную ширину элемента и означает, что максимальная ширина элемента статьи не будет превышать 760 пикселей. Сохраните файл и просмотрите его в браузере. Если вы растянете окно браузера шире 760 пикселей, то увидите, что по бокам от текста появился синий фон, а сам текст больше не расширяется.

С другой стороны, если вы уменьшите окно браузера, сделав его ширину меньше 760 пикселей, то строки текста сожмутся. В этом и заключается преимущество свойства `max-width` — оно позволяет ограничить максимальную ширину, не устанавливая минимальную. Это свойство очень полезно при разработке сайтов для экранов различных размеров, таких как настольные компьютеры, ноутбуки, планшеты и смартфоны. Это важный элемент *адаптивного дизайна*, с которым вы познакомитесь в главе 17.

Теперь при увеличении ширины окна браузера максимальная ширина текста ограничена. Кроме этого, было бы неплохо выровнять его по центру страницы вместо привязки к левому краю.

3. Добавьте еще одно свойство к стилю `article`. Код должен выглядеть следующим образом:


```
article {  
  max-width: 760px;  
  margin: 0 auto;  
}
```

Значение свойства `margin` определяет расстояние между элементом и другими элементами вокруг него. Подробнее о свойстве `margin` вы узнаете в разделе «Управление размерами полей и отступов» главы 7. Отмечу лишь, что в данном примере значения левого и правого поля определяются автоматически, то есть браузер сам вычисляет величину левого и правого поля элемента `article`. Когда ширина окна браузера превысит 760 пикселей, элемент `article` прекратит расширяться, поэтому браузер автоматически добавит пустое пространство слева и справа от элемента, по сути выравнивая его по центру относительно своего окна.

Сейчас вы добавите еще один дополнительный стиль — смежный родственный селектор — для форматирования абзаца, следующего сразу после первого заголовка (этого же эффекта можно достигнуть, создав класс и применив его к этому абзацу, но смежный родственный селектор не требует изменения HTML-кода).

4. Добавьте последний стиль:

```
h1+p {  
  color: rgb(255,255,255);  
  font-size: 1.2em;  
  line-height: 140%;  
}
```

Он будет применен к любому абзацу, следующему *сразу* за элементом `h1`, то есть к первому абзацу после верхнего заголовка страницы. Он не будет применен ко второму или последующим абзацам. С помощью этого селектора можно легко изменить внешний вид вводного абзаца, чтобы выделить его визуально и обозначить начало статьи.

Стиль изменяет цвет и размер шрифта. Свойство `line-height` (о котором вы прочтаете в разделе «Форматирование абзацев» главы 6) определяет пространство между строками в абзацах (параметр, также известный как *интерлиньяж*).

Если вы просмотрите страницу в браузере сейчас, то увидите, что цвет текста верхнего абзаца стал белым, шрифт крупнее, а между строками увеличилось расстояние (рис. 3.14). Если вы удалите этот абзац в HTML-коде, то заметите, что оставшийся абзац также станет белого цвета и с более крупным шрифтом, поскольку теперь он будет смежным родственником элемента `h1`.

Итак, мы ознакомились с различными типами селекторов. Более подробно вы изучите их (и не только их) в практикумах следующих глав этой книги, но на текущий момент вы уже должны понимать, для чего нужны различные селекторы и почему одни должны находиться над другими.

ПРИМЕЧАНИЕ

Окончательную версию созданной в этой главе веб-страницы вы можете найти в папке `03_finished` загруженного архива с примерами.

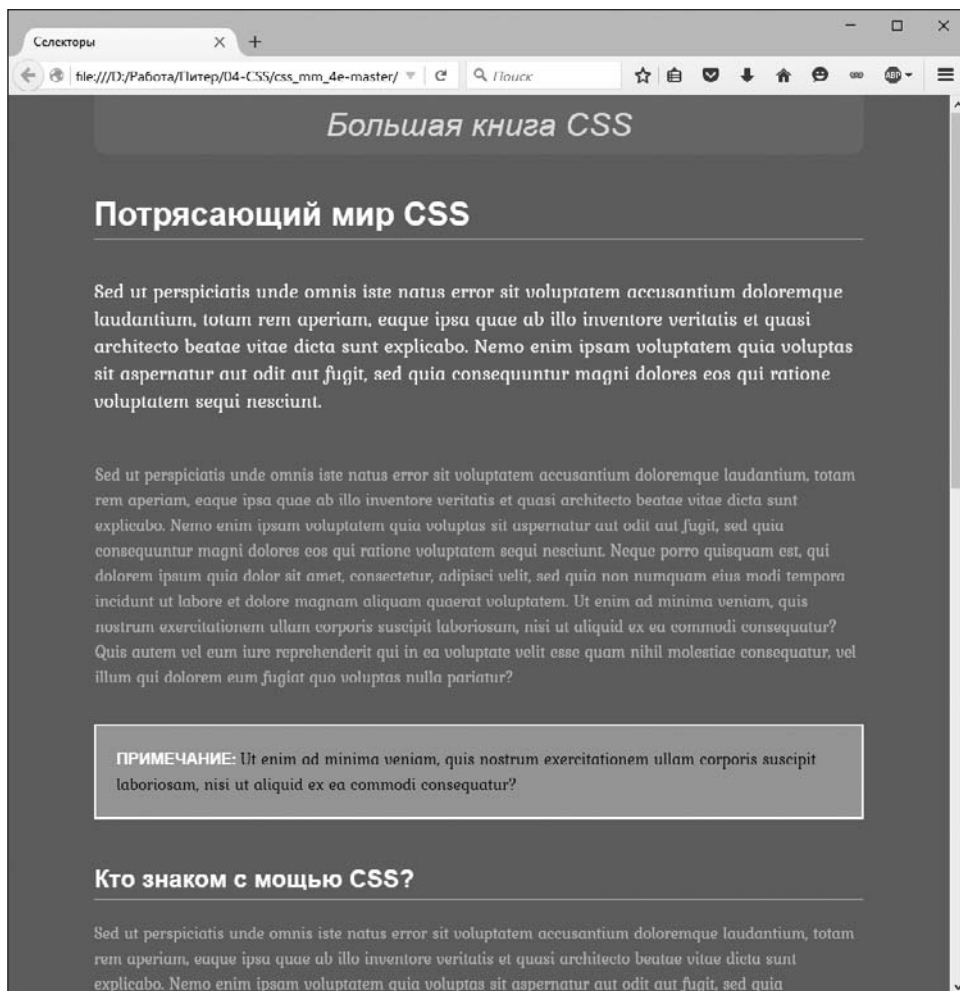


Рис. 3.14. Теперь веб-страница действительно имеет законченный вид. Настройка ширины контента и приемы типографики улучшили аскетичный вид HTML-страницы, показанной в начале практикума этой главы

4 Механизм наследования стилей

Дети наследуют некоторые черты своих родителей, к примеру цвет глаз, рост.

Иногда черты наследуются от более отдаленных предков, например бабушек и дедушек или прародителей. Как вы убедились в предыдущей главе, модель семейных отношений применима и к структуре языка HTML. И точно так же, как люди, элементы могут унаследовать CSS-свойства от своих предков.

Что такое наследование?

Наследование — это прием, с помощью которого CSS-свойства, относящиеся к одному элементу веб-страницы, распространяются и на вложенные элементы. Например, абзац `p` всегда находится внутри тела страницы `body`. Так, атрибуты, применяемые к элементу `body`, наследуются `p`. Допустим, вы создали селектор тега (см. главу 3) для элемента `body`, который устанавливает атрибут `color` (например, темно-красный цвет). Производные элементы, являющиеся потомками `body`, то есть расположенные внутри него, наследуют атрибут. Это означает, что любой текст, заключенный в теги элементов `h1`, `h2`, `p` и т. д., будет отображен тем же темно-красным цветом.

Механизм наследования многоуровневый, то есть его эффект не только распространяется на прямых потомков (дочерние элементы), но и переносится на все вложенные элементы. Если, например, элементы `em` и `strong` расположены внутри абзаца `p`, то они также унаследуют атрибуты любого стиля, применяемого к `body`.

ПРИМЕЧАНИЕ

Как описано в главе 3, любой элемент, вложенный в другой, является его потомком. Так, элемент `p`, находящийся внутри `body`, — потомок. В то же время `body` — предок. Потомки (по аналогии с детьми и внуками) наследуют атрибуты своих предков (по аналогии с родителями и прародителями).

Это может казаться немного непонятным и запутанным, но механизм наследования *на самом деле* экономит очень много времени. Представьте, что ни один атрибут не наследуется вложенными элементами и у вас есть абзац текста, который содержит другие элементы, такие как `strong` и `em`, выделяющие фрагмент текста, или `a`, добавляющий гиперссылку. Если вы создали стиль, форматировующий этот абзац шрифтом `Varela Round` размером 32 пиксела фиолетового цвета,

было бы странно, если бы внутри `em`, `strong` и `a` отобразился прежний стиль, нарушая внешний вид страницы (рис. 4.1). Вам пришлось бы создавать другой стиль для форматирования элемента `em`, чтобы добиться внешнего вида, как у элемента `p`.

Наследование работает не только со стилями элементов (тегов), но и с любыми другими. Когда вы применяете класс (см. раздел «Классы: точное управление» главы 3) к какому-нибудь элементу, то вложенные в него элементы наследуют стили. То же самое справедливо и для идентификаторов, селекторов потомков и других типов стилей, рассмотренных в главе 3.

Упрощение таблиц стилей через наследование

Вы можете использовать преимущества механизма наследования в своих интересах для того, чтобы упростить и ускорить написание таблиц стилей. Предположим, вы хотите отобразить весь текст веб-страницы одинаковым шрифтом. Вместо того чтобы создавать стили для каждого элемента, просто создайте один для `body` (или создайте класс и примените его). Укажите нужный шрифт, и все элементы веб-страницы унаследуют его:

```
body {  
  font-family: Arial, Helvetica, sans-serif;  
}
```

Вы также можете использовать наследование для применения стилей к целому *разделу* веб-страницы. Например, вы можете применять, как и большинство дизайнеров, элемент `div` (см. раздел «Классы: точное управление» главы 3) для определения таких областей страницы, как шапка, панель навигации, колонтитул, или, если используете HTML5-элементы, можно добавить один из элементов деления на разделы, например `header`, `aside`, `footer` или `article`. Применяя стиль к внешнему элементу, вы выделяете специфические CSS-свойства для всех вложенных элементов, находящихся внутри данного раздела веб-страницы. Чтобы весь текст на панели навигации был отображен тем же цветом, можно создать стиль со свойством `color` и применить его к `div`, `header`, `article` или другим элементам деления на разделы. Все элементы, заключенные внутри, в том числе `p`, `h1` и т. д., унаследуют этот цвет шрифта.

Рассмотрим рис. 4.1. *Вверху*: элементу абзаца назначено определенное начертание, размер, цвет шрифта. Абзацы наследуют эти свойства и имеют единообразный стиль. Элементы внутри абзаца (`em`, `strong` и `a` — выделены на рисунке) наследуют эти свойства для сохранения внешнего вида остальных частей абзаца.

Внизу: если бы наследования не существовало, то веб-страница выглядела бы так, как показано в нижней части рисунка. Обратите внимание, что элементы `strong`, `em` и `a` сохранили обычное начертание, размер и цвет шрифта, определенные браузером по умолчанию. Чтобы отформатировать их подобно остальной части абзаца, вам пришлось бы создавать дополнительные стили.

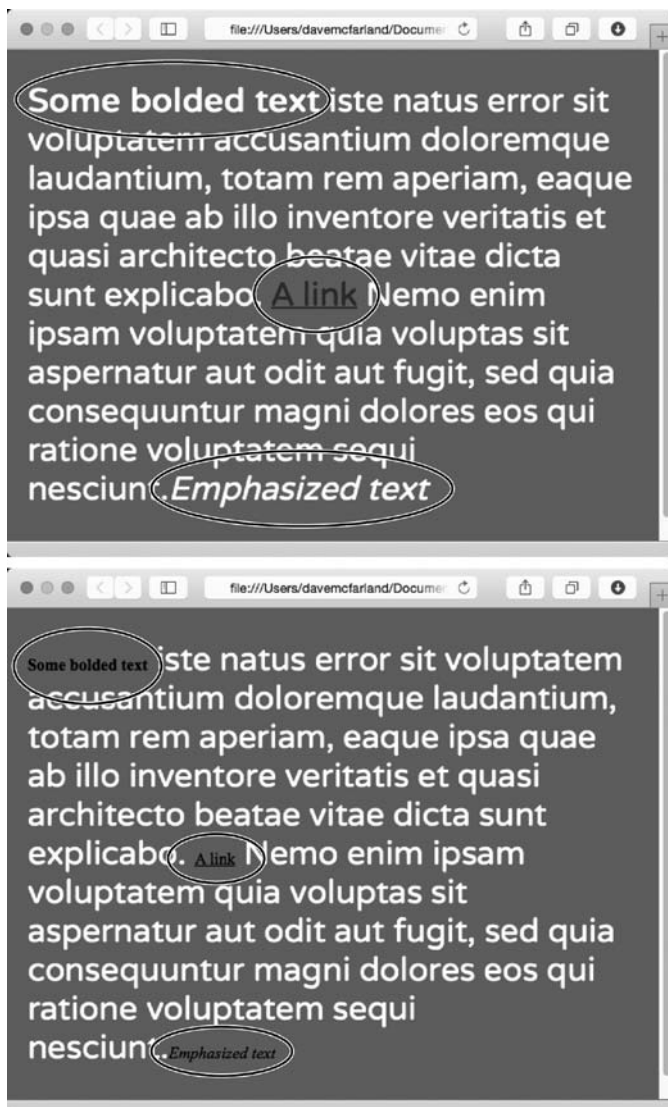


Рис. 4.1. Наследование позволяет копировать свойства из окружающих элементов

Ограничения наследования

Механизм наследования неидеален. Многие CSS-свойства вообще не наследуются, например `border` (позволяющий оформить в рамку элемент веб-страницы). Однако если бы наследование было применимо к этому свойству, то все вложенные элементы были бы одинаковы. Например, если вы добавите рамку к `body`, то она будет во всех маркированных списках (в каждом пункте, подпункте и т. д.) (рис. 4.2).

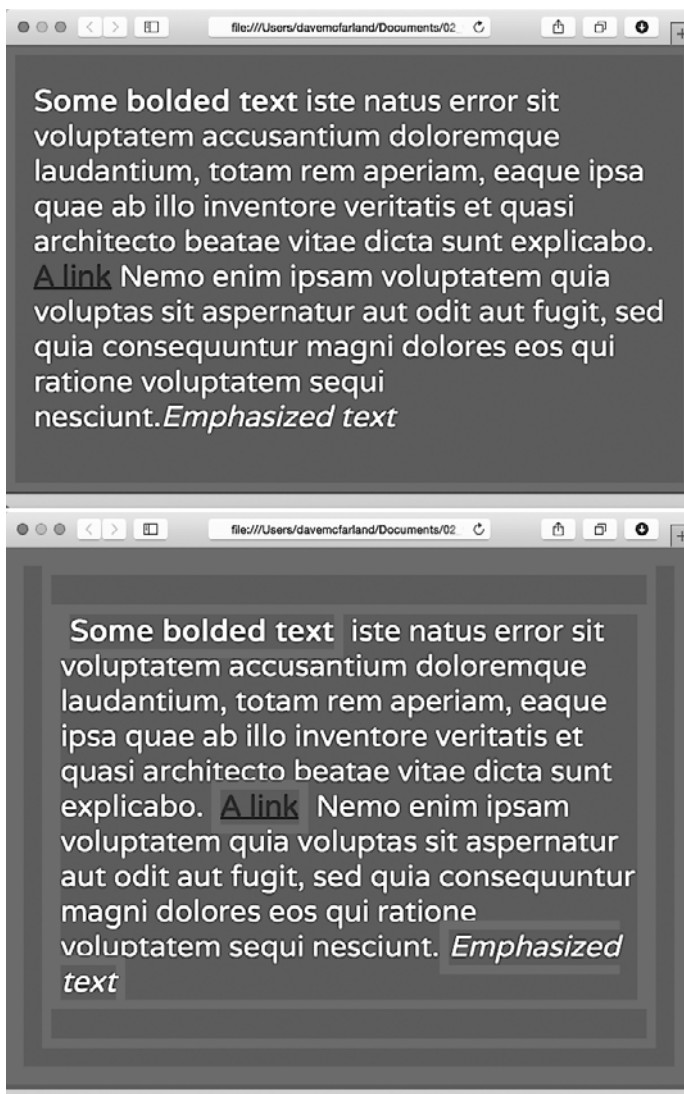


Рис. 4.2. К счастью, не все свойства наследуются. Рамка, относящаяся к абзацу на этой странице (толстая красная линия вокруг текста) (*вверху*), не наследуется элементами, находящимися внутри. Если бы она наследовалась, мы бы получили нагромождение рамок (*внизу*)

ПРИМЕЧАНИЕ

Полный список CSS-свойств, включая их подробное описание, параметры наследования и т. д., приведен в приложении 1.

Ниже приведены конкретные случаи, когда наследование не применяется.

- Как правило, свойства, которые затрагивают размещение элементов на странице (поля, фоновый цвет, границы элементов), не наследуются.

- Браузеры используют собственные встроенные стили для форматирования различных элементов. Заголовки обычно отображаются крупным полужирным шрифтом, ссылки — синим цветом и т. д. Даже если определен конкретный размер кегля для текстового контента веб-страницы и применен к элементу `body`, заголовки будут отображены большим по размеру шрифтом. Элементы `h1` будут крупнее `h2`. Точно так же, когда вы устанавливаете цвет шрифта применительно к `body`, гиперссылки веб-страницы все равно будут отображены синим цветом с подчеркиванием.

ПРИМЕЧАНИЕ

Рекомендуется игнорировать встроенные стили браузеров — это упростит создание сайтов, совместимых с различными типами браузеров. В главе 5 вы узнаете, как добиться этого.

Если возникает конфликт, то побеждает более специфичный стиль. Другими словами, когда вы применяете CSS-свойство к элементу веб-страницы (например, устанавливаете размер шрифта для маркированного списка (элемента `ul`)) и оно конфликтует с наследуемым (например, размером шрифта `body`), браузер использует специфичное свойство, более близко относящееся к форматируемому элементу (в данном случае применяется размер шрифта, определенный для `ul`).

ПРИМЕЧАНИЕ

Такие типы конфликтов между стилями встречаются очень часто, и правила, определяющие, как должен вести себя браузер, называются каскадностью. Подробнее об этом вы узнаете в следующей главе.

Практикум: наследование

В этом практикуме, состоящем из трех частей, вы увидите, как функционирует механизм наследования. Сначала создадим простой селектор тега и понаблюдаем, каким образом он передает свои настройки вложенным элементам. Создадим класс, который использует наследование для изменения форматирования всей веб-страницы. И наконец, рассмотрим случаи отступления от правил, на которые следует обратить внимание.

Перед началом урока нужно загрузить архив с файлами примеров, расположенный по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами. Файлы текущего практикума находятся в папке 04.

Одноуровневое наследование

Для того чтобы увидеть и понять, как работает механизм наследования, добавим стиль к определенному элементу и посмотрим, как он воздействует на вложенные элементы. Все три части этого практикума взаимосвязаны, поэтому сохраняйте свой файл в конце каждого урока для его последующего использования.

1. Откройте файл `inheritance.html` в редакторе HTML-кода.

Файл уже содержит внутреннюю таблицу стилей с одним селектором тега, придающим элементу `body` фоновый цвет.

ПРИМЕЧАНИЕ

Вообще, в сайтах лучше использовать внешние таблицы стилей по причинам, описанным в главе 2. Иногда проще начать разработку CSS-дизайна отдельных веб-страниц, используя внутреннюю таблицу, как в этом примере, а уже потом преобразовать ее во внешнюю.

2. Добавьте еще один стиль после стиля `<body>`:

```
p {
  color: rgb(50,122,167);
}
```

С этим свойством мы работали в практикуме предыдущей главы. Оно устанавливает цвет текста. А ваша таблица стилей уже готова.

3. Чтобы посмотреть на результат работы, откройте страницу в браузере.

Цвет всех четырех абзацев страницы изменился с черного на синий (рис. 4.3).

Обратите внимание, как этот стиль `p` воздействует на *другие* элементы. Они вложены в `p` и также меняют цвет. Например, текст, *заключенный* в `em` и `strong` внутри каждого абзаца, также изменяется на синий, при этом сохраняется выделение полужирным и курсивным начертаниями. В конечном счете устанавливается тот цвет текста абзаца, который вы хотели, *независимо* от любых других элементов.

Без наследования таблицы стилей было бы очень трудно создавать: элементы `em`, `a` и `strong` не унаследовали бы свойство цвета от `p`. Следовательно, пришлось бы создавать дополнительные стили с селекторами потомков, например `p em` или `p strong`, чтобы правильно отформатировать текст.

Но вы увидите, что ссылка в конце первого абзаца не изменила свой цвет, оставшись, как ей и положено, синей. Как уже упоминалось, для определенных элементов у браузеров есть собственные стили, поэтому наследование к ним не применяется. Дополнительные сведения о подобном поведении можно найти в главе 5.

Наследование при форматировании всей веб-страницы

Наследование работает и с классами. Любой элемент с примененным к нему стилем переносит CSS-свойства и на его потомков. Учитывая это, можно пользоваться наследованием для быстрого изменения дизайна всей веб-страницы.

1. Вернитесь к HTML-редактору с открытым файлом `inheritance.html`.

Сейчас мы добавим новый стиль после только что созданного стиля элемента `p`.

2. Щелкните кнопкой мыши сразу за закрывающей скобкой селектора `p`. Нажмите клавишу `Enter` для создания новой строки и введите `.content {`. Теперь дважды нажмите клавишу `Enter` и укажите закрывающую скобку `}`.

Сейчас мы создадим новый класс для `body`, который окружит остальные элементы на странице.

3. Перейдите к строке между двумя скобками и добавьте к стилю следующие свойства:

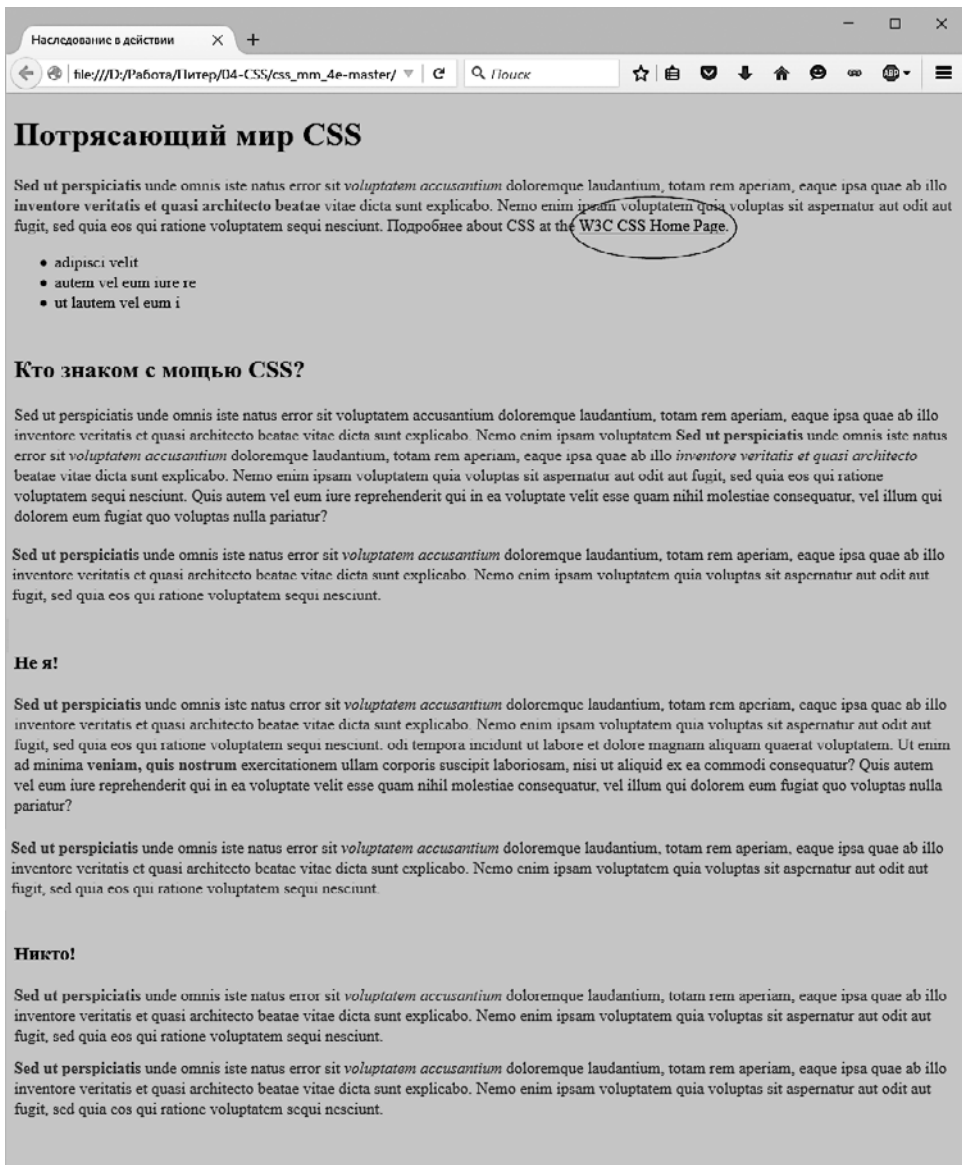


Рис. 4.3. Принцип наследования в действии! Текст, выделенный полужирным шрифтом, приобрел тот же цвет, что и абзац p, окружающий его. Но что это? Ссылка в конце первого абзаца осталась синей (выделена на рисунке). Почему так произошло, вы узнаете в следующей главе

```
font-family: "Helvetica Neue", Arial, Helvetica, sans-serif;
font-size: 18px;
color: rgb(194,91,116);
max-width: 900px;
margin: 0 auto;
```


Законченный стиль должен иметь следующий вид:

```
.content {
  font-family: "Helvetica Neue", Arial, Helvetica, sans-serif;
  font-size: 18px;
  color: rgb(194,91,116);
  max-width: 900px;
  margin: 0 auto;
}
```

Этот класс устанавливает начертание, размер и цвет шрифта. Он также задает ширину и центрирует стиль на странице (мы рассматривали это в практике предыдущей главы).

4. Теперь вернитесь к открывающему тегу `<body>` (расположен строкой ниже закрывающего тега `</head>`) и введите перед закрывающей скобкой через пробел `class="content"`.

Теперь тег `<body>` должен выглядеть следующим образом: `<body class="content">`. Благодаря наследованию все элементы, заключенные внутри `body` (текст которых отображен в окне браузера), наследуют свойства стилей и, соответственно, используют тот же шрифт.

5. Сохраните и просмотрите веб-страницу в браузере.

Как видно на рис. 4.4, наш класс обеспечил всему тексту веб-страницы согласованный внешний вид, без резких переходов, плавно сочетающий все фрагменты содержимого. И заголовки, и абзацы, заключенные в `body`, — все элементы веб-страницы за счет изменения свойств шрифта приобрели новый стиль.

Страница выглядит интересно, но теперь рассмотрим ее более детально: изменение цвета затронуло только заголовки и маркированный список на странице, однако текст заголовка имеет другой размер по сравнению с абзацами, даже несмотря на то, что стиль определяет точный размер шрифта. Каким образом каскадные таблицы стилей узнали, что размер заголовка не должен быть таким же, как размер текста? И почему не применили к вложенным в `body` элементам `p` новый цвет?

ПРИМЕЧАНИЕ

Почему мы используем класс `content` вместо стиля элемента `body`, чтобы изменить вид страницы? В данном примере селектор тега еще хорошо работает. Но применение класса к элементу `body` — это отличный способ настроить по индивидуальному образцу внешний вид нескольких страниц сайта. Например, если они все используют одну и ту же таблицу стилей, то стиль элемента `body` будет применяться к `body` на каждой странице вашего сайта. А создавая различные классы (или идентификаторы), вы можете создавать различные стили элемента `body` для разных разделов сайта или разных типов страниц.

Вы видите, в чем суть каскадности таблиц стилей? В этом примере для элемента `p` образовался конфликт стилей, в частности двух одинаковых атрибутов цвета, — стиля элемента, созданного в шаге 2 в подразделе «Одноуровневое наследование» выше, и класса, созданного только что. Если произошла такая ситуация, то браузер должен выбрать один из стилей. Используется более близкий (специфичный) к элементу стиль, то есть цвет, который вы явно назначили `p`. О правилах каскадности будет рассказано в следующей главе.



Рис. 4.4. Стиль, примененный к элементу `body`, переносит его свойства на все элементы, отображаемые в окне браузера. Благодаря этому очень легко форматировать эффекты на странице

Исключения механизма наследования

Наследование применяется не всегда, и в некоторых случаях это очень хорошо. Для отдельных свойств наследование имело бы исключительно негативное влияние на дизайн веб-страницы. В заключительной части практикума будет приведен пример *исключения (бездействия)* механизма наследования. Вы увидите, что отступы, поля и границы (среди других свойств) не наследуются вложенными элементами. Далее вы узнаете, почему так предусмотрено.

1. Вернитесь в редактор HTML-кода с открытым файлом `inheritance.html`.
Дополним только что созданный стиль `p`.
2. Найдите определение стиля `p`, щелкните кнопкой мыши сразу за свойством цвета (`color: rgb(50,122,167);`) и нажмите клавишу `Enter` для перехода на новую строку.
Сейчас мы создадим отступ слева для всех абзацев веб-страницы.
3. Добавьте два свойства к стилю следующим образом:

```
p {
  color: rgb(50,122,167);
  padding-left: 20px;
  border-left: solid 25px rgba(255,255,255,.5);
}
```

Этот код изменит отступ слева каждого абзаца и сместит текст таким образом, чтобы он не касался границы. Свойство `padding` создает отступ от границы размером 20 пикселей.

4. Сохраните файл и просмотрите его в браузере.

Обратите внимание, что все абзацы `p` приобрели слева толстую светлую границу. Однако у элементов, *вложенных* в абзац `p` (например, `em`), нет такого отступа или границы (рис. 4.5). Это поведение браузера оправданно: веб-страница выглядела бы странно, если бы каждый элемент `em` и `strong` в абзаце имел дополнительную границу и отступ слева размером 20 пикселей!

Если вы хотите увидеть, что бы случилось, если бы эти свойства наследовались, отредактируйте селектор `p` таким образом: `p, p *`, что сделает его групповым. Первая часть — это тот селектор `p`, который вы только что создали. А вторая часть — `p *` — буквально означает следующее: «выберите все элементы внутри абзаца `p` и примените к ним стиль» (`*` — универсальный селектор, был описан в предыдущей главе).

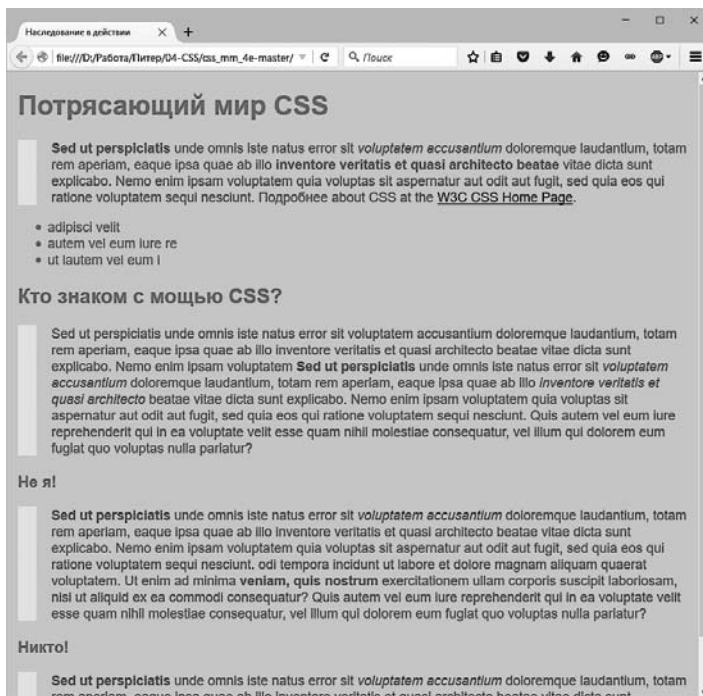


Рис 4.5. Большинство CSS-свойств наследуются вложенными элементами (например, цвет шрифта). Однако поля, отступы и границы являются исключением. Справочная информация относительно наследования CSS-свойств приведена в приложении 1

ПРИМЕЧАНИЕ

Финальный вариант веб-страницы, которую вы создали в этом практикуме, можно найти в папке `04_finished`.

5 Управление сложной структурой стилей: каскадность

По мере того как вы будете создавать все более сложные таблицы стилей, вы все чаще будете задаваться вопросом: почему конкретный элемент веб-страницы выглядит именно так? Из-за особенностей наследования CSS, описанных в предыдущей главе, на любой элемент влияют окружающие его элементы. Стиль становится сложнокомбинированным. Например, элемент `body` может передать форматирование абзацу текста, а тот — гиперссылке, находящейся внутри абзаца. Другими словами, может произойти наследование CSS-свойств и `body`, и `p` *одновременно*.

Кроме того, может возникать конфликт: одно и то же CSS-свойство многократно описывается в различных стилях, которые затем относятся к одному элементу веб-страницы (например, стиль `p` во внешней таблице стилей и другой стиль `p` во внутренней). Вы можете наблюдать такую ситуацию, когда текст отображается ярко-синим цветом, несмотря на то, что установлен красный. Существует особая система, которая управляет взаимодействием стилей и определяет их приоритет при конфликте. Этот механизм называется *правилами каскадности* или просто *каскадностью*.

ПРИМЕЧАНИЕ

В данной главе описываются проблемы, возникающие при построении сложных таблиц стилей, работа которых основана на принципах наследования и использовании более сложных типов селекторов, таких как селекторы потомков (см. раздел «Идентификаторы: отдельные элементы веб-страницы» главы 3). Все правила достаточно логичны и понятны для опытного веб-дизайнера, но у начинающего все-таки может возникнуть множество сложностей. Можно сравнить обучение этому языку с овладением тонкостями налогового кодекса. Если у вас нет желания углубленно изучать все особенности языка CSS, просто пропускайте теоретический материал и переходите к выполнению заданий практикума. Вы всегда сможете вернуться к этой главе после того, как овладеете основами каскадных таблиц стилей.

Каскадность стилей

Каскадность — ряд правил, определяющих, какие именно свойства стилей применяются к элементам веб-страницы, то есть задающих последовательность приме-

нения многократно определенных стилей. Другими словами, каскадность указывает, каким образом браузер должен обработать сложную структуру, относящуюся к одному и тому же элементу, и что делать, если возникает конфликт свойств. Это происходит в двух случаях: из-за механизма наследования (одинаковое свойство наследуется от нескольких родительских элементов-предков) и когда один или более стилей применяются к одному элементу веб-страницы (например, вы применили к абзацу стиль с помощью класса, а также создали стиль для элемента `p` и оба стиля применяются к этому абзацу).

Объединение унаследованных стилей

Как вы узнали из предыдущей главы, наследование каскадных таблиц стилей гарантирует, что однородные, взаимосвязанные элементы веб-страницы (например, все слова в абзаце, даже если они являются вложенными гиперссылками или расположены в другом элементе) получают форматирование родительских элементов. Это избавляет от необходимости создания отдельных стилей для каждого элемента веб-страницы. Поскольку элемент может унаследовать свойства *любого* предка (например, ссылка, наследующая шрифт родительского элемента `p`), определить, почему конкретный элемент отформатирован именно так, может быть сложной задачей. Предположим такую ситуацию: к элементу `body` применен конкретный шрифт, `k p` — размер шрифта, а `k a` — цвет. Таким образом, любой элемент `a` абзаца унаследует стили `body` и `p`. Другими словами, унаследованные стили будут объединены, сформировав один сложный.

Страница, представленная на рис. 5.1, имеет три стиля: один для `body`, второй для `p` и третий для `strong`. CSS-код выглядит следующим образом:

```
body { font-family: Verdana, Arial, Helvetica, sans-serif; }
p { color: #F30; }
strong { font-size: 24px; }
```

Как показано на рисунке, элемент `strong` отформатирован с изменением шрифта, цвета и кегля. Конечный внешний вид элемента определяется сочетанием всех трех стилей (собственным, устанавливающим кегль (размер шрифта) и двумя наследованными от селекторов тегов `body` и `p`).

Элемент `strong` вложен в абзац, который, в свою очередь, вложен в `body`. Стили `strong` наследует у всех элементов-предков, получая форматирование шрифта (`font-family`) от элемента `body` и цвет (`color`) от абзаца `p`. Кроме того, `strong` имеет собственное правило, устанавливающее размер шрифта 24 пиксела. Конечный внешний вид элемента определяется сочетанием всех трех стилей. Другими словами, `strong` выглядит так, будто для него создали следующий стиль:

```
strong {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  color: #F30;
  font-size: 24px;
}
```

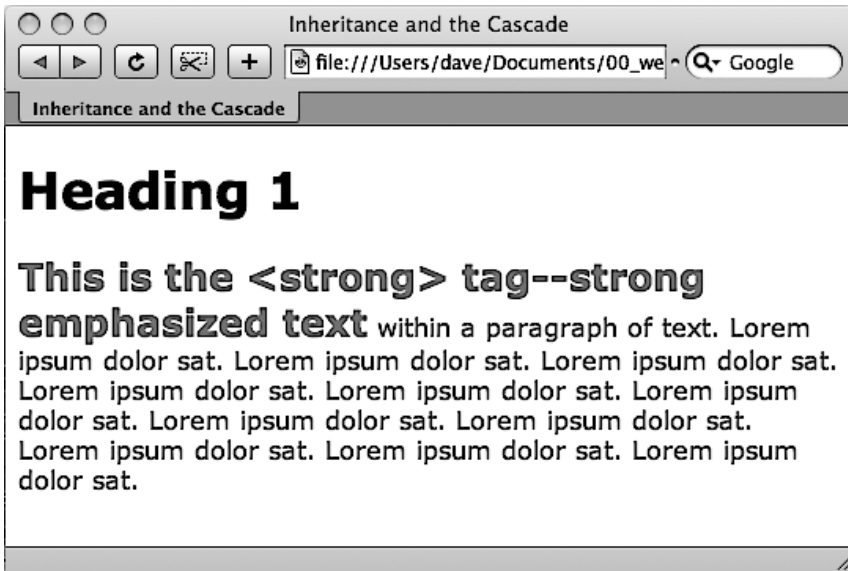


Рис. 5.1. Благодаря наследованию можно отформатировать один элемент несколькими стилями

Превосходство ближайших предков

Как видно из предыдущего примера, комбинирование стилей различных элементов с применением наследования создает полный набор форматирования. Но что произойдет в случае конфликта унаследованных свойств каскадных таблиц стилей? Представьте себе страницу, где вы установили красный цвет шрифта для элемента `body` и зеленый цвет шрифта для элемента `p`. Теперь предположим, что внутри абзаца имеется элемент `strong`. Он наследует стиль как у `body`, так и у элемента `p`. Так какого цвета текст внутри элемента `strong`: красный или зеленый? Правильный ответ: зеленый цвет, унаследованный от стиля абзаца. Браузер применит стиль, который является *самым близким* по отношению к формируемому элементу.

В данном примере любые свойства, унаследованные от `body`, являются скорее общими. Они относятся ко всем элементам веб-страницы. Однако стиль `p` более близок к `strong` и, можно сказать, находится по соседству с ним. Форматирование `p` применяется непосредственно к абзацу и его вложенным элементам.

По сути, если элемент не имеет собственного, явно определенного стиля, то при возникновении конфликтов с унаследованными свойствами одержат победу ближайшие предки (рис. 5.2, 1).

Это еще один пример, позволяющий разобраться в концепции каскадности. Если есть CSS-стиль, определяющий цвет текста для таблицы `table`, и еще один, определяющий *другой* цвет для ячейки `td`, то элементы, заключенные в ячейки данной таблицы (`td`), — элементы абзаца, заголовка, маркированного списка — унаследуют цвет стиля `td`, поскольку он является ближайшим предком.

Преимущества непосредственно примененного стиля

Единственный стиль, обладающий наивысшим приоритетом, является ближайшим предком в «генеалогическом» дереве каскадных таблиц стилей. Это тот стиль, который напрямую применен к элементу. Предположим, что цвет шрифта устанавливается для элементов `body`, `p` и `strong`. Стиль абзаца `p` более специфичен, чем `body`, но `strong` — еще специфичнее, чем все остальные. Он форматирует только элементы `strong`, игнорируя любые конфликтующие свойства, унаследованные от других элементов (см. рис. 5.2, 2). Другими словами, свойства стиля, явно определенного для элемента, отменяют любые унаследованные.

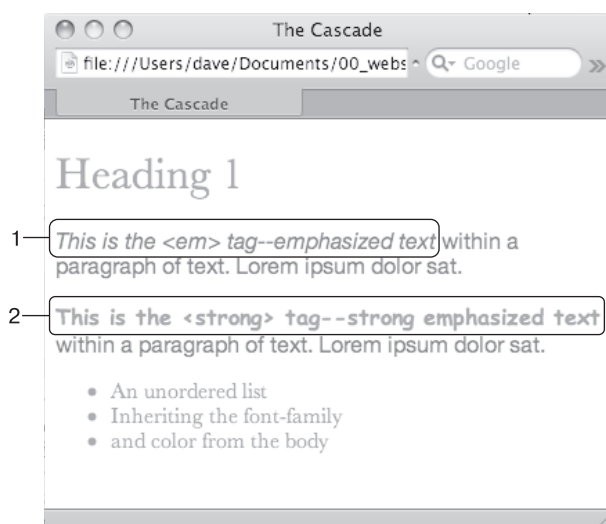


Рис. 5.2. Браузеры определяют, какие свойства применять при возникновении конфликтов с наследуемыми стилями

Это правило объясняет, почему некоторые свойства не применяются. Гиперссылка, находящаяся внутри абзаца, текст которого отформатирован красным шрифтом, по-прежнему будет отображена в окне браузера синей с подчеркиванием. Это происходит потому, что есть собственный предопределенный стиль для элемента привязки `a`. Таким образом, унаследованный цвет текста не будет применен.

ПРИМЕЧАНИЕ

О том, как игнорировать встроенные стили браузера для элемента `a` и установить желаемый стиль для гиперссылки, будет рассказано в главе 9.

Множество стилей для одного элемента

Наследование — один из способов форматирования элемента несколькими стилями. Однако можно определить много стилей, которые будут применены *непосредственно*

к элементу. Рассмотрим такой случай: имеется прикрепленная к веб-странице внешняя таблица стилей, среди прочих содержащая стиль для элемента абзаца `p`. В коде самой страницы есть внутренняя таблица, которая *также* содержит стиль для элемента абзаца `p`. К одному из этих элементов на странице применен еще и класс. Так, для одного элемента абзаца `p` непосредственно определено три различных стиля. Какой (или *какие*) из них должен использовать браузер?

Ответ: все зависит от множества факторов — типов стилей, порядка, в котором они были созданы. Браузер сам выбирает, применить один или несколько стилей одновременно. Рассмотрим ситуации, когда множество стилей может применяться к одному и тому же элементу.

- **К элементу одновременно применен селектор тега и класс.** Например, стиль тега для элемента `h2` и класс `.leadHeadline` могут быть представлены следующим HTML-кодом:

```
<h2 class="leadHeadline">Для вас будущее наступило!</h2>
```

К элементу `h2` будут применены оба стиля.

ПРИМЕЧАНИЕ

Описание того, что произойдет в случае конфликта стилей, следует далее.

- **Одинаковые имена стилей встречаются несколько раз в таблице.** К примеру, это может быть два стиля — групповой селектор (см. раздел «Форматирование групп элементов» главы 3), например `.leadHeadline`, `.secondaryHeadline`, `.newsHeadline` и класс `.leadHeadline` в той же таблице. Форматирование элемента с классом `leadHeadline` будет определяться обоими стилями.
- **К элементу одновременно применены стили класса и идентификатора.** Это может быть идентификатор `#banner` и класс `.news`. HTML-код имеет вид: `<div id="banner" class="news">`. К элементу `div` будут применены оба стиля.
- **С веб-страницей связано несколько таблиц, и в каждой из них содержится стиль с одинаковым именем.** Стили с одинаковыми именами могут использоваться во внешней и во внутренней таблице стилей. Или же один и тот же стиль может появиться в нескольких внешних таблицах стилей, каждая из которых связана с одной и той же страницей.
- **Единственный элемент веб-страницы может быть объектом воздействия сложных селекторов.** Это обычная ситуация, когда вы используете селекторы потомков (см. раздел «Классы: точное управление» главы 3). Допустим, на веб-странице имеется элемент `div` (например, `<div id="mainContent">`) и внутри него заключен абзац с классом: `<p class="byline">`. В этом случае к этому абзацу будут применимы следующие селекторы:

```
#mainContent p
#mainContent .byline
p.byline
.byline
```

Если к конкретному элементу веб-страницы применено несколько стилей, то браузер объединяет их свойства при условии, что они не конфликтуют между со-

бой. Приведенный ниже пример разъясняет этот принцип. Предположим, есть абзац, в котором указаны имя дизайнера веб-страницы и ссылка на адрес его электронной почты. HTML-код может выглядеть следующим образом:

```
<p class="byline">Создано <a href="mailto:jean@cosmofarmer.com">Жорой Ивановым</a></p>
```

Между тем в таблице стилей веб-страницы присутствуют три стиля для форматирования гиперссылки:

```
a { color: #6378df; }  
p a { font-weight: bold; }  
.byline a { text-decoration: none; }
```

Первый стиль окрашивает все элементы `a` в зеленовато-голубой цвет; второй стиль форматирует все элементы `a`, находящиеся в абзаце `p`, полужирным шрифтом; а третий стиль убирает подчеркивание ссылок, вложенных в элементы с классом `byline`.

Воздействие всех трех стилей распространяется на такой часто используемый элемент, как `a`. Ни одно из правил этих стилей не конфликтует с остальными. Ситуация похожа на случай, рассмотренный выше в разделе «Объединение унаследованных стилей»: стили объединяются между собой и образуют один комбинированный «суперстиль», содержащий все три свойства. Таким образом, данная гиперссылка отображается зеленовато-голубым полужирным шрифтом и без подчеркивания.

ПРИМЕЧАНИЕ

Имейте в виду, что на стиль форматирования этой ссылки также могут влиять унаследованные свойства. Например, может быть унаследовано начертание шрифта абзаца. Лучше понять работу механизма каскадности вам помогут несколько инструментов, описанных во врезке «ЧаВо» ниже.

Особенности каскадности: какие стили имеют преимущество

Предыдущий пример слишком прост. Но что получится, если каждый из трех стилей ссылок, приведенных выше, имеет *свое* определение начертания в свойстве `font-family`? Какой из них выберет браузер?

Если вы внимательно читали книгу, то помните, что механизм каскадности устанавливает несколько правил. *Побеждают (имеют преимущество) свойства самого близкого по отношению к формируемому элементу, самого специфичного стиля.* Однако, как и в примере со стилями, не совсем понятно, какой из них является наиболее специфичным. К счастью, CSS предлагает метод *определения приоритетов*. Он основан на присвоении значений в условных единицах каждому типу селекторов: селекторам тегов, классам, идентификаторам и т. д. Система работает так.

- Селектор тегов имеет специфичность, равную **1 условной единице**.
- Класс — **10 условных единиц**.
- Идентификатор — **100 условных единиц**.
- Строчный стиль — **1000 условных единиц**.

ПРИМЕЧАНИЕ

Математические расчеты, используемые для определения приоритетов, на самом деле немного сложнее. Но эта формула работает во всех случаях, кроме самых странных и запутанных. Чтобы узнать о том, как браузеры рассчитывают приоритеты, посетите страницу w3.org/TR/css3-selectors/#specificity.

Чем больше числовое значение, тем выше специфичность (значимость) данного типа селектора. Предположим, вы создали три стиля:

- стиль тега для элемента `img` (специфичность = 1);
- класс с именем `.highlight` (специфичность = 10);
- идентификатор с именем `#logo` (специфичность = 100).

Веб-страница содержит следующий HTML-код: ``. Если определить одинаковый атрибут во всех трех стилях (например, свойство `border`), то будет применен стиль идентификатора (`#logo`), как наиболее специфичного.

ПРИМЕЧАНИЕ

Псевдоэлемент (например, `::first-child`) обрабатывается браузером как селектор тегов и имеет специфичность 1 пункт. Псевдокласс (например, `:link`) рассматривается как класс и имеет специфичность 10 пунктов (см. раздел «Псевдоклассы и псевдоэлементы» главы 3).

Поскольку селекторы потомков состоят из нескольких простых, например `content p` или `h2 strong`, определить их специфичность сложнее: необходимо вычислить суммарное значение их приоритетов (табл. 5.1).

Таблица 5.1. Когда к единственному элементу применяется несколько стилей, браузер должен определить, какой из них будет применен при возникновении конфликта

Селектор	Идентификатор	Класс	Тег	Итого
<code>p</code>	0	0	1	1
<code>.byline</code>	0	10	0	10
<code>p.byline</code>	0	10	1	11
<code>#banner</code>	100	0	0	100
<code>#banner p</code>	100	0	1	101
<code>#banner .byline</code>	100	10	0	110
<code>a:link</code>	0	10	1	11
<code>p:first-line</code>	0	0	2	2
<code>h2 strong</code>	0	0	2	2
<code>#wrapper #content .byline a:hover</code>	200	20	1	221

ПРИМЕЧАНИЕ

Наследуемые свойства вообще лишены специфичности. Так, даже если элемент унаследует, например, селектор `#banner`, то его свойства в любом случае будут заменены теми, что непосредственно относятся к этому элементу.

Разрешение конфликтов: побеждает последний стиль.

Два стиля могут иметь одинаковый приоритет. Конфликт свойств с одинаковой специфичностью может произойти при двойном определении одинаковых селекторов. У вас может быть селектор элемента `p` во внутренней таблице и такой же во внешней. Или два различных стиля могут иметь одинаковый приоритет. В таком случае более значимым будет последний определенный стиль.

Ниже представлен пример HTML-кода.

```
<p class="byline">Создано <a class="email" href="mailto:jean@cosmofarmer.com">Жорой Ивановым</a></p>
```

В таблице для веб-страницы, содержащей вышеприведенные абзац и гиперссылку, присутствует два стиля:

```
p.email { color: blue; }
.byline a { color: red; }
```

Оба стиля имеют специфичность 11 (10 — для класса и 1 — для селектора тега) и относятся к элементу `a`. Конфликт этих стилей очевиден. Какой цвет выберет браузер для окрашивания гиперссылки в приведенном абзаце? Красный, так как он указан последним в таблице стилей.

ЧАВО**Инструменты в помощь**

Существует ли какое-нибудь вспомогательное средство, чтобы представить в понятной форме воздействие, оказываемое механизмом каскадности на конечный дизайн веб-страницы?

Попытки определить все входы и выходы унаследованных свойств и конфликтующие стили сбивали с толку многих исследователей. Более того, применение математических приемов для определения специфики стиля как-то не привлекает обычного веб-дизайнера, особенно при использовании огромных таблиц стилей с большим количеством селекторов потомков.

Во всех современных браузерах имеется встроенная помощь в виде инспектора. Проще всего обследовать элемент на странице и все каскадные таблицы стилей, влияющие на его внешний вид, щелкнув правой кнопкой мыши (щелкнув кнопкой мыши с нажатой клавишей `^` в операционной системе OS X) на элементе (заголовке, ссылке, абзаце или изображении) и выбрав в контекстном меню пункт Исследовать элемент (Inspect Element). Откроется панель (обычно в нижней части веб-страницы), отображающая исходный код страницы с выбранным HTML-элементом. (В браузере Safari

сначала нужно установить флажок Показывать меню "Разработка" в строке меню (Show Develop Menu) на вкладке Дополнения (Advanced) окна, открывающегося выбором команды меню Safari ► Настройки (Preferences).)

В правой части панели вы увидите стили, примененные к элементу. Обычно это «вычисленный» стиль, итоговая сумма всех CSS-свойств, примененных к элементу путем наследования и каскадирования, или некий синтезированный стиль элемента. Чуть ниже будут показаны правила стилей, примененные к элементу, перечисленные в порядке от более специфичных (в верхней части списка) к менее (в нижней части списка).

Вероятнее всего, вы увидите, что в перечне стилей некоторые свойства зачеркнуты. Это свидетельствует о том, что данное свойство либо не применяется к элементу, либо было замещено более специфичным стилем. Чтобы просмотреть два кратких учебных пособия по использованию инструментария разработчика по анализу CSS, посетите сайты tinyurl.com/oetspo9 и tinyurl.com/nb3ls5u.

Теперь представьте, что два стиля поменялись местами и таблица стилей имеет следующий вид:

```
.byline a { color: red; }
p .email { color: blue; }
```

В данном случае гиперссылка будет синей. Стиль с селектором `p .email` расположен в таблице после строки `.byline a`, поэтому его свойства имеют преимущество.

Теперь рассмотрим, что произойдет, если имеются конфликтующие стили (или их свойства) во внешней и внутренней таблицах стилей. В этом случае важна последовательность размещения на веб-странице (в HTML-коде файла). Если вы сначала добавляете внутреннюю таблицу, используя элемент `style` (см. главу 2), а *затем* присоединяете внешнюю таблицу с помощью элемента `link`, то будет применен стиль последней (в сущности, это принцип, который только что был описан: *значение имеет последний из конфликтующих стилей*). Вывод: будьте последовательны в размещении в коде веб-страницы ссылки на внешнюю таблицу стилей. Сначала ее нужно присоединить, а только затем добавлять внутренние таблицы, если абсолютно невозможно обойтись без одного или нескольких стилей, применяемых к одной странице.

УСТРАНЕНИЕ ОШИБОК

Аннулирование специфичности

Каскадные таблицы стилей предоставляют возможность полностью отменить специфичность стилей. Вы можете использовать этот прием, чтобы никакой другой более специфичный стиль не заместил конкретное свойство элемента веб-страницы. Для этого вставьте после нужного свойства значение `!important`.

Рассмотрим пример. Допустим, существует два стиля:

```
.nav a { color: red; }
a { color: teal !important; }
```

При обычном раскладе ссылка, вложенная в элемент с классом `.nav`, была бы окрашена в красный цвет, так как стиль, определенный селектором `.nav a`, специфичнее, чем селектор тега `a`. Однако добавление слова `!important` подразумевает, что данное свойство всегда будет иметь больший приоритет. Так, в вышеприведенном примере все ссылки веб-страницы, в том числе вложенные с классом `nav`, будут отображены зеленовато-голубым цветом.

Обратите внимание, что вы применяете метку `!important` к отдельному свойству, а не ко всему стилю,

поэтому нужно добавить слово `!important` в конце каждого свойства, которое не должно быть замещено. В заключение нужно сказать: когда для двух одинаковых свойств различных стилей указано слово `!important`, опять вступает в силу правило специфичности и приоритет имеет более специфичный атрибут из отмеченных.

Будьте осторожны, применяя метку `!important`. Поскольку этот способ кардинален, при слишком частом его использовании ваши стили не будут соответствовать обычным правилам каскада, что приведет к «эскалации» использования метки `!important`. Другими словами, чтобы скомпенсировать специфичность свойства `!important` в одном стиле, вы будете применять его в другом. Затем, чтобы скомпенсировать действие метки `!important` во втором стиле, вам придется добавить его в третий стиль и т. д. Поэтому не рекомендуется использовать метку `!important` слишком часто. Прежде чем добавлять ее, попробуйте другой способ преодолеть конфликт, например переименуйте или поменяйте местами стили в таблице.

Управление каскадностью

Как вы могли заметить, чем больше CSS-стилей создано, тем больше вероятность запутаться в них. Например, можно создать класс, устанавливающий для шрифта определенное начертание и размер, но применение его к абзацу ни к чему не приводит. Эта проблема обычно связана с механизмом каскадности. Даже когда вы абсолютно уверены в конечном результате, все равно может существовать более специфичный стиль.

Есть несколько вариантов решения этой проблемы. Во-первых, можно использовать слово `!important` (как описано во врезке выше), чтобы *гарантировать* применение конкретного свойства. Этот способ не совсем удобен, так как трудно предугадать, что вы не захотите отменить такую специфичность свойства. Рассмотрим два других метода управления каскадностью.

Изменение приоритетов

На рис. 5.3 и 5.4 приведен пример, когда определенный стиль тега проигрывает в каскадной игре. К счастью, в большинстве случаев можно запросто изменить специфичность одного из конфликтующих стилей и прибегнуть к метке `!important` для по-настоящему безысходных случаев. На рис. 5.3 два стиля форматировать первый абзац. Класс `.intro` не так специфичен, как `#sidebar p`. Таким образом, свойства класса `.intro` не будут применены к абзацу. Чтобы увеличить специфичность класса, добавьте к стилю идентификатор `#sidebar .intro`.

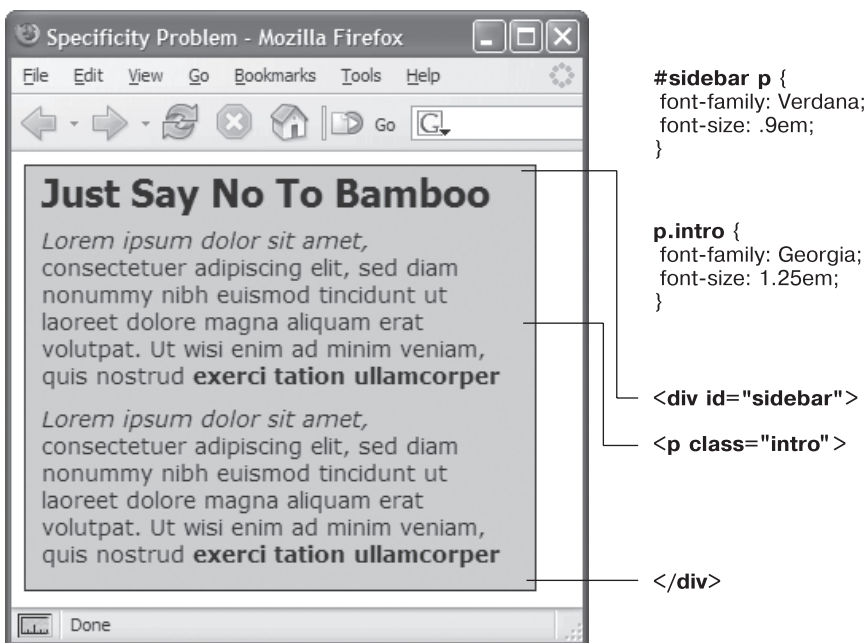


Рис. 5.3. Даже если класс применен к конкретному элементу (см. первый абзац), его свойства не всегда имеют эффект

В данном случае абзац размещен внутри элемента `div` с идентификатором `#sidebar`, поэтому селектор потомков `#sidebar p` специфичнее класса `.intro`. Решение: необходимо повысить значимость класса `.intro`, добавив перед ним идентификатор — `#sidebar p.intro` (рис. 5.4).

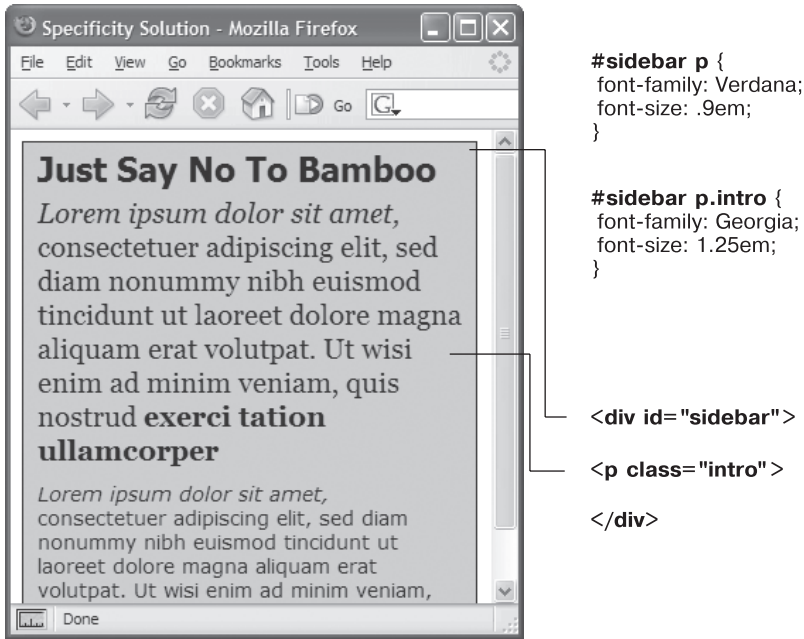


Рис. 5.4. Можно придать большую значимость стилевому классу, добавив перед ним идентификатор

Но простое добавление дополнительных селекторов с целью «победы» свойств стиля может привести к так называемым *войнам специфичности*, когда у вас в конечном итоге получаются таблицы стилей, содержащие очень длинные и запутанные имена стилей наподобие `#home #main #story h1`. Нужно стараться избегать стилей такого типа и использовать максимально короткие селекторы.

Выборочное игнорирование приоритетов

Можно тонко проработать дизайн веб-страниц, *выборочно* отменяя стили. Предположим, вы создали внешнюю таблицу и назвали ее `styles.css`, связав со всеми страницами сайта.

Этот CSS-файл содержит общие определения стилей дизайна страниц: шрифт и цвет для элементов заголовков `h1`, стиль элементов формы и т. д. Возможно, вы хотите, чтобы на главной (домашней) странице заголовков `h1` выглядел иначе, чем он отображен на остальных. Например, был выделен крупным полужирным шрифтом. Или абзац должен быть отформатирован шрифтом меньшего размера, чтобы вместить больший объем информации. Другими словами, вы все еще хотите ис-

пользовать большинство стилей файла `styles.css`, но необходимо отменить несколько атрибутов отдельных элементов (`h1`, `p` и т. д.).

Один из способов — создание внутренней таблицы, содержащей стили, которые вы хотите отменить и переопределить. Предположим, в файле `styles.css` имеется следующий стиль:

```
h1 {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 24px;
  color: #000;
}
```

Вы хотите, чтобы заголовок `h1` главной веб-страницы был отображен крупным шрифтом красного цвета. Для этого добавьте во внутреннюю таблицу следующий стиль:

```
h1 {
  font-size: 36px;
  color: red;
}
```

В данном случае к элементу `h1` на главной странице сайта будет применен шрифт Arial (из внешней таблицы стилей), но в то же время он будет окрашен в красный цвет размером 36 пикселей (стили, определенные во внутренней таблице стилей).

СОВЕТ

Убедитесь, что вы ссылаетесь на внешнюю таблицу стилей перед кодом внутренней в разделе заголовка HTML-страницы. Это гарантирует, что нужные стили будут иметь преимущество в тех случаях, когда специфичность идентична.

Другой метод заключается в создании еще одной внешней таблицы. Например, таблицы `home.css`, которую нужно будет присоединить к главной веб-странице в дополнение к `styles.css`. Файл `home.css` будет содержать те стили `styles.css`, которые вы хотите переопределить. Для правильной работы файл `home.css` должен быть присоединен после `styles.css` в HTML-коде веб-страницы:

```
<link rel="stylesheet" href="css/styles.css"/>
<link rel="stylesheet" href="css/home.css"/>
```

СОВЕТ

Еще один способ выполнить точное постраничное форматирование веб-страниц основан на использовании различных имен класса для элемента `body` веб-страниц разного типа. Например, чтобы изменить дизайн отдельных веб-страниц, применяются идентификаторы `.review`, `.story`, `.home`, а затем создаются селекторы потомков. Эта методика рассматривалась в главе 3.

Как избежать конфликтов приоритетов

Многие веб-дизайнеры предпочитают вместо идентификаторов использовать классы. Одна из причин состоит в том, что идентификаторы обладают очень большой специфичностью, поэтому для их отмены требуется более высокая специфичность. Зачастую это приводит к войнам специфичности, при которых таблицы стилей

загружаются с излишне пространными и сложными селекторами. Суть этой проблемы проще объяснить на примере. Предположим, что в HTML-коде вашей страницы есть следующий фрагмент:

```
<div class="article">
<p>Абзац</p>
<p>Другой абзац</p>
<p class="special">Особенный абзац</p>
</div>
```

Вы решили, что нужно текст абзаца внутри div-контейнера `article` сделать красным, и создали следующий селектор потомков:

```
#article p { color: red; }
```

Но затем вам захотелось, чтобы текст одного абзаца с классом `special` был синим. Если просто создать класс, вы не получите желаемого результата.

```
.special { color: blue; }
```

Как уже говорилось, когда определяется, какое из свойств нужно применить к элементу, браузер использует для разрешения конфликтов стилей простую математическую формулу: браузеры присваивают идентификатору значение 100, классу — значение 10, а селектору тегов — значение 1. Поскольку селектор `#article p` составлен из одного идентификатора и одного тега (суммарный показатель специфичности — 101), он заменяет собой простой стиль класса, заставляя вас изменить селектор:

```
#article .special {color: blue; }
```

К сожалению, это изменение оказывается причиной возникновения еще двух проблем. Во-первых, селектор становится длиннее, и во-вторых, теперь этот синий цвет применяется только тогда, когда класс `special` появляется внутри какого-нибудь элемента с идентификатором `article`. Например, если вы скопируете HTML-код `<p class="special">Особенный абзац</p>` и вставите его в какой-нибудь другой позиции страницы, текст уже не будет синим. Иначе говоря, использование идентификаторов делает селекторы не только длиннее, но и бесполезнее.

А теперь посмотрим, что получится, если просто заменить все идентификаторы классами. Предыдущий код HTML приобретет следующий вид:

```
<div class="article">
<p>Абзац</p>
<p>Другой абзац</p>
<p class="special">Особенный абзац</p>
</div>
```

И код CSS можно заменить следующим:

```
.article p { color: red; }
p.special { color: blue; }
```

Первый стиль, `.article p`, является селектором потомков с уровнем специфичности 11 условных единиц. Второй стиль, `p.special`, также имеет уровень 11 условных единиц (один селектор тега и один класс) и означает «применить следующие свойства к любому абзацу `p` с классом `special`». Теперь, если вырезать этот HTML-

код и вставить его в какую-нибудь другую позицию страницы, вы получите синий текст, обусловленный стилем, к чему, собственно, вы и стремились.

Этот только один из примеров, но не составит труда найти таблицы стилей с абсурдно длинными селекторами наподобие `#home #article #sidebar #legal p` и `#home #article #sidebar #legal p.special`.

Идентификаторы могут быть и полезными. Например, многие системы управления контентом (CMS) используют их для определения уникальных элементов страницы. Их превосходящая значимость может легко превысить специфичность других стилей. Однако не стоит злоупотреблять идентификаторами. Они не дают ничего, что нельзя было бы получить с использованием простого класса или селектора тега, а их высокая специфичность может только привести к ненужному усложнению таблиц стилей.

ПРИМЕЧАНИЕ

Более подробные аргументы, почему следует избегать применения идентификаторов, изложены на странице tinypurl.com/6ge7z86.

С чистого листа

Браузеры применяют к элементам свои собственные стили: например, шрифт заголовков `h1` крупнее `h2`, они оба выделены полужирным начертанием, в то время как шрифт текста абзацев меньше и не выделен полужирным шрифтом; ссылки подчеркнутые и имеют синий цвет, а у маркированных списков есть отступ. В стандарте HTML нет ничего, что бы определяло все это форматирование: браузеры просто добавляют его для того, чтобы простой HTML-код при визуализации был более читабельным. Разные браузеры обрабатывают элементы очень похоже, но все же неодинаково.

Так, например, браузеры Chrome и Firefox используют свойство `padding` для создания отступа в маркированных списках, а Internet Explorer применяет свойство `margin`. Кроме того, вы сможете найти небольшие различия в размерах элементов в разных браузерах и обнаружить вовсе вводящее в заблуждение использование отступов самыми распространенными на сегодняшний день браузерами. Из-за этих несоответствий вы столкнетесь с проблемами, когда, например, Firefox добавит отступ от верхнего края, а Internet Explorer этого не сделает. Такого рода проблемы не ваша вина — они вытекают из различий стилей, встроенных в браузер.

Для предотвращения несоответствия стилей между браузерами лучше всего начинать таблицу стилей с чистого листа. Другими словами, удалить встроенное в браузер форматирование и добавить собственное. Концепция устранения стилей браузера называется *сбросом стилей* (CSS Reset).

В частности, есть базовый набор стилей, который вы должны включить в верхнюю часть своей таблицы стилей. Они устанавливают базовые значения для свойств, которые обычно по-разному обрабатываются во всех браузерах.

Рассмотрим шаблон сброса стандартных стилей:

```
html, body, div, span, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li,
fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed, figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary, time, mark, audio, video {
```

```
margin: 0;
padding: 0;
border: 0;
font-size: 100%;
vertical-align: baseline;
}
```

```
article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav,
section {
  display: block;
}
body {
  line-height: 1.2;
}
ol {
  padding-left: 1.4em;
  list-style: decimal;
}
ul {
  padding-left: 1.4em;
  list-style: square;
}
table {
  border-collapse: collapse;
  border-spacing: 0;
}
```

ПРИМЕЧАНИЕ

Показанный выше код сброса стилей взят из известного и авторитетного источника, составленного Эриком Мейером, который можно найти по адресу tinypurl.com/2amlyf.

Первый стиль — очень длинный групповой селектор, затрагивающий наиболее распространенные элементы и «обнуляющий» их. Он удаляет поля и отступы, устанавливая 100%-ный размер шрифта и убирая полужирное начертание. Благодаря этому шагу ваши элементы смотрятся практически одинаково (рис. 5.5). Но так и нужно — ведь вы хотите начать с чистого листа, а затем добавить собственное форматирование, чтобы все браузеры согласованно отображали ваш HTML-код.

Второй селектор (`article`, `aside`, `details`...) является еще одним групповым селектором, помогающим устаревшим браузерам правильно отображать новые HTML5-элементы. Третий селектор тега (`body`) устанавливает пространство между строками в абзаце (свойство `line-height`). Сведения о свойстве `line-height` будут приведены в следующей главе.

ПРИМЕЧАНИЕ

Вам не нужно вводить этот код самостоятельно. Вы найдете файл с именем `reset.css` в папке 05 архива с примерами с сайта github.com/mrightman/css_4e. Просто скопируйте стили из этого файла и вставьте их в свою таблицу стилей.

Еще один вариант сброса — использовать файл `normalize.css` — бесплатную таблицу стилей с открытым исходным кодом, которая позволяет различным браузерам отображать одни и те же элементы в согласованном виде. Он широко используется веб-дизайнерами. Найти файл `normalize.css` можно по адресу tinypurl.com/oy58gya.

Четвертый и пятый селекторы тегов (для элементов `ol` и `ul`) устанавливают согласованные отступы от левого края и определенное форматирование (в главе 6 вы изучите форматирование списков), а последний стиль упрощает добавление рамок к ячейкам таблицы (польза от установки этого стиля будет рассмотрена в главе 11).

Практикум: механизм каскадности

В этом практикуме вы увидите, как элементы взаимодействуют между собой и конфликтуют, что приводит к неожиданным результатам. Для начала взгляните на простую страницу, упомянутую выше, на которой были сброшены стандартные стили. Кроме того, есть несколько других стилей, обеспечивающих простую разметку. Затем мы создадим два стиля и понаблюдаем за действием механизма каскадности. Мы также рассмотрим, как наследование влияет на элементы веб-страницы и как браузер решает конфликты CSS-стилей. Наконец, вы узнаете, как решаются проблемы механизма каскадности.

Перед началом урока нужно загрузить файлы примеров, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами. Файлы текущего практикума находятся в папке 05.

Сброс стандартных стилей и создание стилей с чистого листа

Для начала взгляните на страницу, с которой будете работать.

1. В браузере откройте страницу `cascade.html` из папки 05 (см. рис. 5.5).

Страница выглядит очень просто: две колонки (одна из них на синем фоне) и много однотипного текста. К этому файлу уже применялись некоторые стили, поэтому откройте CSS-код в текстовом редакторе и просмотрите его.

2. Откройте файл `styles.css` из папки 05 в текстовом либо в HTML-редакторе.

Это внешняя таблица стилей, которую использует файл `cascade.html`. В ней уже есть несколько стилей: первая группа сбрасывает стандартные стили, что мы обсуждали на предыдущей странице. Они устраняют основные стили браузера, поэтому весь текст пока выглядит одинаково. Скоро вы будете создавать собственные стили, чтобы эта страница смотрелась более эффектно.

Последние два стиля — классы `.main` и `.sidebar` — создают две колонки, показанные на рис. 5.5. HTML-код разделен на два элемента `div`, каждый из которых имеет собственный класс. Стили классов здесь, по существу, размещают два элемента `div` так, чтобы они отображались друг рядом с другом в виде колонок (как управлять макетом страницы и создавать колонки, вы узнаете в части III).

Сначала вы добавите пару стилей, чтобы улучшить общий вид страницы и ее верхний заголовок.

3. В файле `styles.css` добавьте два этих стиля в нижней части таблицы стилей после строки с последней скобкой `}` класса `.sidebar`:

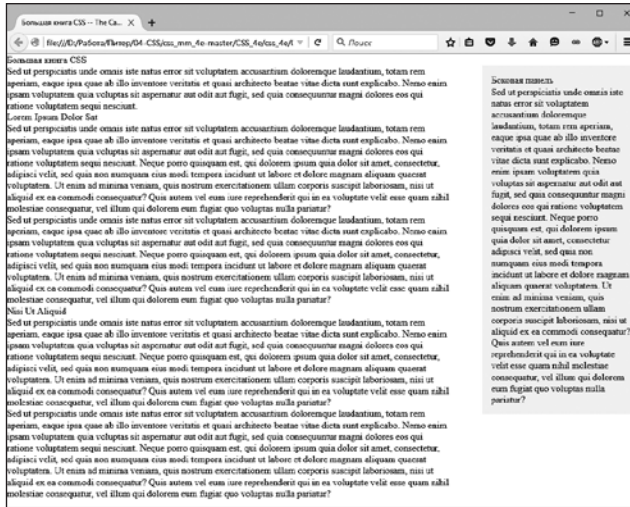


Рис. 5.5. Сброс стандартных стилей на этой странице устраняет небольшие различия в том, как разные браузеры отображают основные HTML-элементы. Устраняются и любые различия в отображении элементов. Ваша задача — взять пустой холст и отформатировать элементы нужным образом

```
body {
  color: #B1967C;
  font-family: "Palatino Linotype", Baskerville, serif;
  padding-top: 115px;
  background: #CDE6FF url(images/bg_body.png) repeat-x;
  max-width: 800px;
  margin: 0 auto;
}
h1 {
  font-size: 3em;
  font-family: "Arial Black", Arial, sans-serif;
  margin-bottom: 15px;
}
```

Первый стиль добавляет фоновое изображение и цвет для страницы, а также устанавливает для нее фиксированную ширину. Сохранив этот файл и просмотрев страницу `cascade.html` в браузере (рис. 5.6), вы заметите, что эти атрибуты не наследуются другими элементами — то же изображение, например, не повторяется за элементами заголовков или абзацев.

Свойства `font-family` и `color`, с другой стороны, наследуются, так что другие элементы на странице теперь используют шрифт `Agial` и имеют коричневый цвет. Тем не менее вы увидите, что, хоть верхний заголовок такого же цвета, как и остальной текст на странице, у него другой шрифт — вот наглядный пример каскадности в действии. Для форматирования элемента `h1` не было назначено никакого цвета, так что заголовок наследует коричневый цвет, который был применен к элементу `body`. Но, поскольку стиль элемента `h1` определяет шрифт, он замещает унаследованный шрифт от стиля `body`.

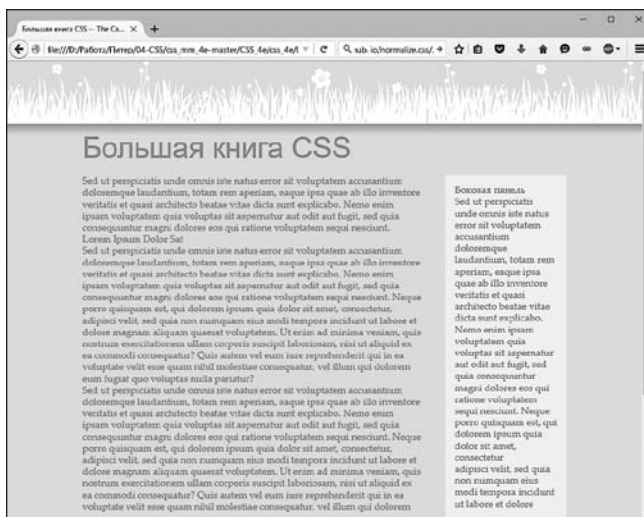


Рис. 5.6. Наследование и каскадность в действии: элемент `h1` в верхней части этой страницы наследует цвет шрифта из форматирования элемента `body`, но получает размер и шрифт из стиля элемента `h1`

Комбинирование стилей

В этом примере мы создадим два стиля. Один стиль будет форматировать все заголовки второго уровня веб-страницы, а другой — более специфичный стиль — будет повторно форматировать те же заголовки, но в крупной, главной колонке веб-страницы.

1. В файле `styles.css` добавьте следующий стиль в конец таблицы стилей:

```
h2 {
  font-size: 2.2em;
  color: #AFC3D6;
  margin-bottom: 5px;
}
```

Этот стиль меняет цвет и увеличивает размер шрифта элемента `h2`, а также добавляет немного пространства под ним. Если вы просмотрите страницу в браузере, то увидите, что заголовки `h2` из основной колонки и те же заголовки из правой колонки похожи друг на друга.

Далее вы создадите стиль для форматирования *только* тех заголовков второго уровня, которые находятся в главной колонке.

2. Вернитесь в HTML-редактор к файлу `styles.css`, щелкните кнопкой мыши сразу после стиля элемента `h2` и нажмите клавишу `Enter`, чтобы перейти на новую строку. Добавьте следующий стиль:

```
.main h2 {
  color: #E8A064;
  border-bottom: 2px white solid;
```

```
background: url(images/bullet_flower.png) no-repeat;
padding: 0 0 2px 80px;
}
```

Вы только что создали селектор потомков, описанный ранее в книге, форматирующий все элементы `h2`, расположенные *внутри* элемента с классом `main`. Две колонки текста на этой странице заключаются в элементы `div` с разными именами классов. У большей, расположенной слева колонки класс носит имя `main`, поэтому такой особый стиль будет применяться только к элементам `h2` внутри этого раздела `div`.

Рассматриваемый стиль похож на тот, который вы создали в практикуме в главе 2 в шаге 17. Он добавляет подчеркивание и значок с изображением цветка к заголовку. Этот стиль также определяет оранжевый цвет шрифта.

3. Сохраните таблицу стилей и снова просмотрите страницу в браузере (рис. 5.7). Вы заметите, что у всех заголовков второго уровня (у двух в основной колонке и у одного в боковой) одинаковый размер шрифта, но у тех двух, которые расположены в основной колонке, также есть подчеркивающая линия и изображение цветка.

Поскольку стиль `.main h2` специфичнее простого стиля `h2`, то при возникновении каких-либо конфликтов между двумя стилями (в данном случае в значении свойства `color`) свойства стиля `.main h2` приоритетнее. Таким образом, хотя шрифту заголовков второго уровня в основной колонке передается синий цвет стиля `h2`, оранжевый цвет более специфичного стиля `.main h2` приоритетнее. Однако, поскольку для стиля `.main h2` не указан размер шрифта или нижнего поля, заголовки в основной колонке получают эти свойства от стиля `h2`.

Преодоление конфликтов

Поскольку свойства каскадных таблиц стилей конфликтуют, когда несколько стилей применяются к одному и тому же элементу, иногда вы можете обнаружить, что ваши страницы выглядят не так, как вы этого ожидали.

Когда это происходит, вам нужно установить, почему это произошло, и внести изменения в селекторы таблиц стилей, чтобы каскадность приводила к нужному результату.

1. Вернитесь к HTML-редактору и открытому файлу `styles.css`. Сейчас вы создадите новый стиль для форматирования исключительно абзацев в основной колонке.
2. Добавьте следующий стиль в конец таблицы стилей:

```
.main p {
  color: #616161;
  font-family: "Palatino Linotype", Baskerville, serif;
  font-size: 1.1em;
  line-height: 150%;
  margin-bottom: 10px;
  margin-left: 80px;
}
```

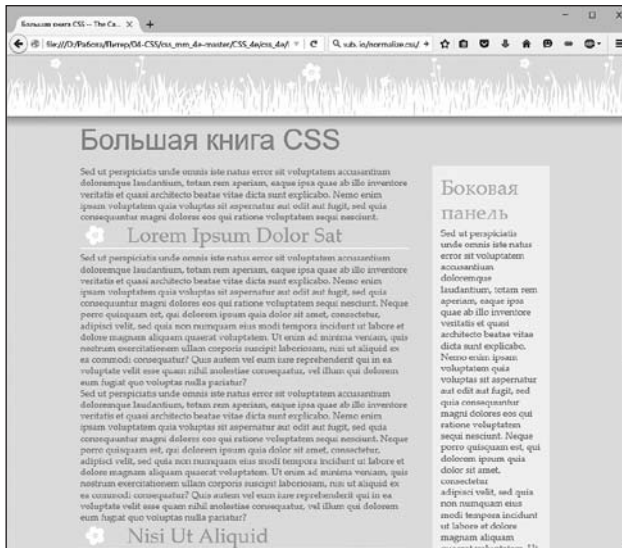


Рис. 5.7. Стили `h2` и `.main h2` применяются к заголовкам второго уровня в левой колонке этой страницы, причем стиль `.main h2` — только к ним

Просмотрев страницу в браузере, вы увидите, что изменился цвет, размер и шрифт текста абзацев, строки растянулись (свойство `line-height`) и изменились отступы абзацев слева и снизу.

Далее вы отформатируете первый абзац более крупным шрифтом с полужирным начертанием, чтобы привлечь к нему внимание. Самый простой способ отформатировать один-единственный абзац — создать класс и применить его к этому абзацу.

3. Добавьте последний стиль в конце таблицы стилей:

```
p.intro {
  color: #6A94CC;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 1.2em;
  margin-left: 0;
  margin-bottom: 15px;
}
```

Этот стиль изменяет цвет, шрифт и размер, а также незначительно регулирует отступы. Все, что вы должны сделать, — применить класс к HTML-коду.

4. Откройте файл `cascade.html` в HTML-редакторе. Найдите элемент `p`, расположенный после заголовка `<h1>Большая книга CSS</h1>` и строки `<div class="main">`, и добавьте следующий атрибут класса:

```
<p class="intro">
```

5. Просмотрите страницу в браузере.

И... абзац никак не изменился. Согласно правилам каскадности, `.intro` — простой класс, а `.main p` — селектор потомков и состоит из имен класса и тега. Они

добавлены для создания более специфичного стиля, поэтому его свойства решают любой конфликт между ним и стилем `.intro`.

Для того чтобы заработал стиль `.intro`, необходимо немного «укрепить» его, наделив этот селектор большей специфичностью.

- Вернемся к файлу `styles.css` в HTML-редакторе и изменим имя стиля с `.intro` на `p.intro`.

Убедитесь в том, что между `p` и `.intro` нет пробела. По сути, вы создали связь: `.main p` — один класс и один селектор тега и `p.intro` — один тег и один класс. Оба имеют уровень специфичности 11, но поскольку `p.intro` появляется в таблице стилей после `.main p`, именно этот селектор побеждает и его свойства применяются к абзацу. (Чтобы преодолеть конфликт, можно создать и еще более специфичный стиль — `.main .intro`.)

- Просмотрите страницу в браузере (рис. 5.8).

Теперь в абзаце цвет шрифта сменился на голубой, шрифт изменился и стал крупнее, а также отсутствует отступ слева. Если бы у вас не было четкого понимания принципа каскадности, вы бы ломали голову над тем, почему этот класс не работал в первый раз.

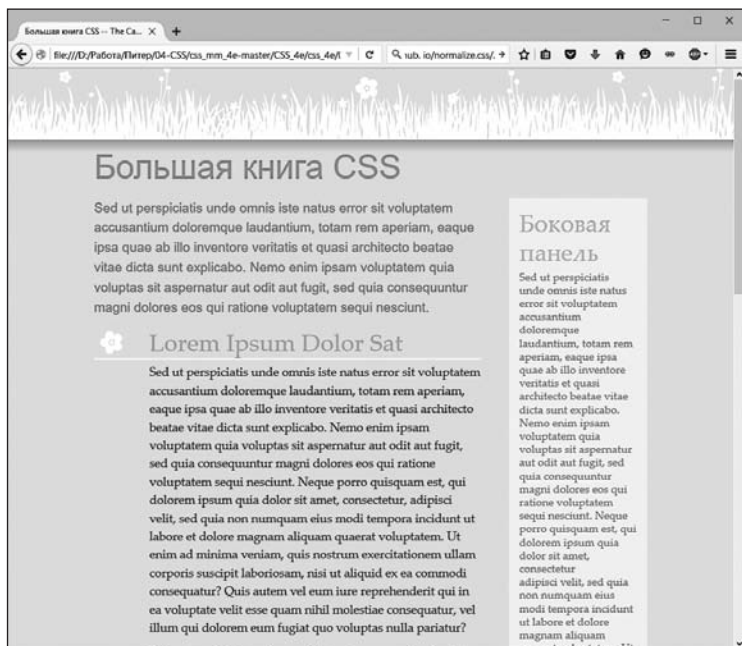


Рис. 5.8. Даже на простой странице с небольшим набором стилей внешний вид элементов зачастую является комбинацией свойств различных стилей

В этой и четырех предыдущих главах мы рассмотрели основы каскадных таблиц стилей. Теперь, в части II, пришло время применить полученные знания для создания настоящих полноценных дизайнов веб-страниц.

ЧАСТЬ II

Применение CSS

Глава 6. Форматирование текста

Глава 7. Поля, отступы, границы

Глава 8. Добавление графики на веб-страницы

Глава 9. Построение навигационной системы сайта

Глава 10. Преобразования, переходы и анимации с помощью CSS

Глава 11. Форматирование таблиц и веб-форм

6 Форматирование текста

Большинство сайтов во Всемирной паутине по-прежнему построены исключительно на текстовом контенте. Несомненно, людям нравится видеть фотографии, видеоклипы, анимацию, но все-таки именно содержательный текстовый материал заставляет их вновь и вновь возвращаться на определенные сайты. Люди жаждут обновлений в социальных сетях типа Facebook, новостей, статей с практическими рекомендациями, рецептов, ответов на актуальные вопросы, полезной информации и новых записей в микроблогах Twitter. С помощью языка CSS вы можете и *должны* придать такой вид заголовкам и абзацам веб-страниц, что они привлекут внимание посетителей не хуже фотографий. Оформление текстового контента называется *типографикой*.

Каскадные таблицы стилей предлагают для этих целей обширный набор мощных команд форматирования, которые позволяют назначать шрифты, цвет, кегль, межстрочный интервал и другие свойства и атрибуты, оказывающие влияние на визуальное восприятие как отдельных элементов веб-страницы (заголовков, маркированных списков, обычных абзацев текста), так и всей веб-страницы, сайта в целом (рис. 6.1). Настоящая глава описывает и показывает все многообразие свойств форматирования текстового содержимого веб-страниц и заканчивается практикумом, позволяющим поупражняться в создании таблиц стилей для текста и применении их к реальным веб-страницам.

Использование шрифтов

Первое, что вы можете сделать для увеличения привлекательности сайта, — применить различные шрифты к заголовкам, абзацам и другим элементам. Для выбора начертания в каскадных таблицах стилей используется свойство `font-family` и указывается тот шрифт, который следует применять. Предположим, вы хотите определить для абзацев шрифт Arial. Для этого можно создать селектор тега для элемента `p` и воспользоваться свойством `font-family`:

```
p {  
    font-family: Arial;  
}
```

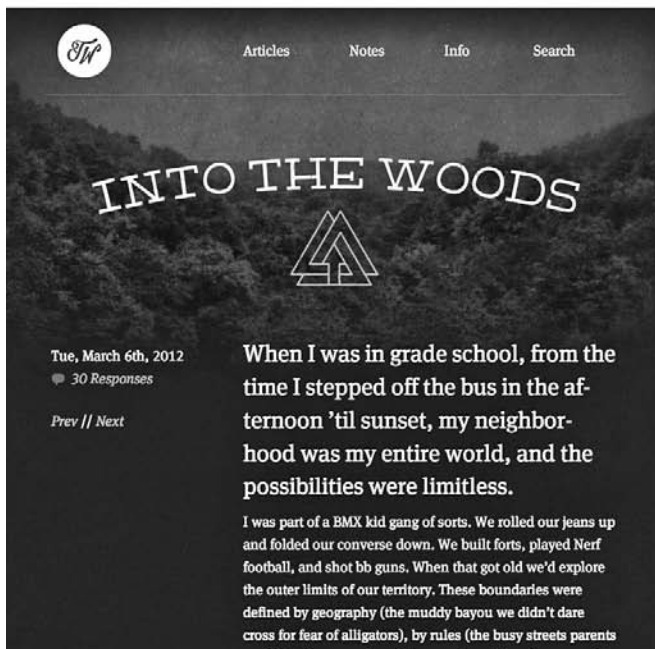


Рис. 6.1. Использование слишком большого количества шрифтов или типографических украшений зачастую приводит читателя в замешательство, затрудняя понимание сути информации, изложенной на веб-странице (вверху). Использование текста со шрифтом различных размеров, умелый подбор стиля и всего пара разных шрифтов облегчает просмотр страницы и делает ее приятной для чтения (внизу)

Изначально свойство `font-family` функционирует, только если у посетителей сайта имеется такой же шрифт, установленный на их компьютерах. Иначе говоря, применительно к показанному выше примеру, если у кого-то из посетителей вашего сайта не будет на компьютере установлен шрифт Arial, абзацы страницы будут отображены с использованием исходного шрифта браузера (обычно это какой-нибудь вариант Times New Roman). Поэтому веб-дизайнеры ограничивались небольшим набором шрифтов, предустановленным на большинстве компьютеров.

В последнее время браузеры стали поддерживать *веб-шрифты*, то есть такие шрифты, которые браузер загружает и применяет при просмотре вашего сайта. Веб-шрифты также используют свойство `font-family`, но требуют набора дополнительного CSS-кода, называемого правилом `@font-face` — оно инструктирует браузер загрузить указанный шрифт. Веб-шрифты открывают многие захватывающие возможности в области дизайна, позволяя выбирать из бурно растущего количества гарнитур.

Но вскоре мы увидим, что они также приносят свой набор проблем. Иначе говоря, как веб-дизайнер вы можете остановиться на выборе проверенных и реально существующих шрифтов, то есть выбрать шрифты из числа установленных на большинстве компьютеров, или же выбрать веб-шрифты, расширив свой дизайнерский кругозор (ценой дополнительной работы). Но вы не ограничены только одним из этих подходов. Многие дизайнеры используют их в сочетании, в одних случаях применяя стандартные шрифты (например, для абзацев основного текста страницы), а в других — веб-шрифты (например, для создания привлекательных заголовков).

Выбор подходящего шрифта

Если для указания шрифта используется свойство `font-family`, посетители могут не увидеть выбранный вами шрифт, так как у них он либо должен быть уже установлен на компьютере, либо, при указании веб-шрифтов, временно загружен в кэш браузера. Поскольку вы не можете знать, доступен ли предпочитаемый вами шрифт конкретному пользователю, сложилась практика указывать не только основной шрифт, но и пару резервных вариантов. Этот список называется *стеком шрифтов*. Если на компьютере посетителя сайта установлен (доступен) шрифт, выбранный первым, он и будет использоваться для форматирования текста. Но если первый шрифт не установлен, то браузер просматривает список, пока не обнаружит на компьютере указанный в стеке шрифт. Идея состоит в том, чтобы определить список похожих шрифтов, присутствующих в большинстве операционных систем. Например:

```
font-family: Arial, Helvetica, sans-serif;
```

В данном примере браузер сначала выяснит, установлен ли на компьютере шрифт Arial. Если да, то браузер использует этот шрифт. В противном случае он ищет шрифт Helvetica и, если и тот не установлен, применяется последний из списка — универсальный рубленый (также называется гротеском, шрифтом без засечек) шрифт семейства `sans-serif`. Если вы вносите в список универсальный тип шрифта (`sans-serif` или `serif`), то браузер выбирает установленный на компьютере шрифт из этого семейства. По крайней мере, таким образом вы можете определить его основной символ.

ПРИМЕЧАНИЕ

На практике для применения определенного стиля вы должны добавить селектор и фигурные скобки. Например:

```
p { font-family: Arial, Helvetica, sans-serif; }
```

Если в этой книге вы встретите примеры вида `font-family: Arial, Helvetica, sans-serif;`, помните, что это всего лишь отдельное свойство CSS-стиля для сокращения и удобства чтения.

Если имя шрифта состоит из нескольких слов, вам следует заключать его в кавычки:

```
font-family: "Times New Roman", Times, serif;
```

Далее представлены некоторые часто используемые сочетания обычно установленных шрифтов, сгруппированные по типу шрифта (каждый список заканчивается универсальным типом шрифта).

Антиквенные шрифты

Антиквенные шрифты (с засечками) идеальны для длинных фрагментов текста, поскольку распространено мнение, что засечки — маленькие росчерки на концах основных штрихов — хорошо для глаз связывают одну букву с другой, делая текст более читабельным. Примерами антиквенных шрифтов являются Times, Times New Roman, Georgia.

- "Times New Roman", Times, serif.
- Georgia, "Times New Roman", Times, serif.
- Baskerville, "Palatino Linotype", Times, serif.
- "Hoefler Text", Garamond, Times, serif.

Примеры этих шрифтов приведены на рис. 6.2.

Сглаживание экранных шрифтов в операционной системе OS X выполняется иначе, чем в Windows. Операционная система Windows использует технологию ClearType, которая позволяет улучшить отображение текста на экране. Вид экранного текста в Windows зависит от используемых настроек ClearType. Подробнее о технологии ClearType можно узнать по адресу tinyurl.com/pmzxbnm.

СОВЕТ

На сайте cssfontstack.com опубликован список шрифтов, по умолчанию установленных в операционных системах OS X и Windows, включая подробное процентное соотношение использования шрифтов для каждой из них. Например, шрифт Courier New установлен на 99,73 % компьютеров, работающих под управлением операционной системы Windows и 95,68 % компьютеров Mac.

Рубленные шрифты

Рубленные шрифты (без засечек) часто используются для заголовков благодаря простому и четкому внешнему виду. Примеры рубленных шрифтов — Arial, Helvetica и Verdana.

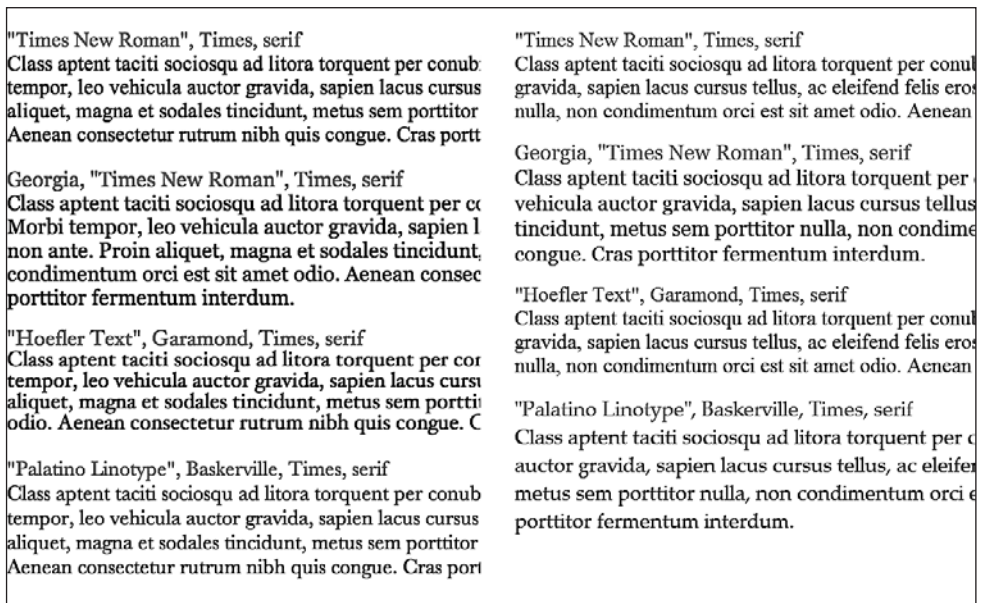


Рис. 6.2. Шрифты не всегда отображаются одинаково в операционных системах Windows (справа) и OS X (слева). У этих двух систем различные наборы предустановленных шрифтов

- Arial, Helvetica, sans-serif.
- Verdana, Arial, Helvetica, sans-serif.
- Geneva, Arial, Helvetica, sans-serif.
- Tahoma, "Lucida Grande", Arial, sans-serif.
- "Trebuchet MS", Arial, Helvetica, sans-serif.
- "Century Gothic", "Gill Sans", Arial, sans-serif.

Примеры данных шрифтов приведены на рис. 6.3.

Некоторые люди верят, что на веб-страницах нужно использовать лишь рубленые шрифты, потому что декоративные штрихи антиквенных шрифтов не очень хорошо отображаются на экранах с низким разрешением. Тем не менее современные мониторы с высоким разрешением, которые отображают большее количество пикселей на дюйм, избавлены от этой проблемы. В конце концов, ваш вкус — лучший ориентир. Выбирайте те шрифты, которые считаете нужными.

Моноширинные и декоративные шрифты

Моноширинный шрифт часто используется для отображения компьютерного кода (например, фрагментов кода повсюду в этой книге). Каждая буква в моноширинном шрифте имеет одинаковую ширину (как буквы в механических печатных машинках).

- "Courier New", Courier, monospace.
- "Lucida Console", Monaco, monospace.

- "Copperplate Light", "Copperplate Gothic Light", serif.
- "Marker Felt", "Comic Sans MS", fantasy.

С примерами этих списков шрифтов можете ознакомиться на рис. 6.4.

<p>Arial, Helvetica, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia nostra, lacus cursus tellus, ac eleifend felis eros non ante. Proin aliquet, i orci est sit amet odio. Aenean consectetur rutrum nibh quis congue.</p> <p>Verdana, Arial, Helvetica, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia auctor gravida, sapien lacus cursus tellus, ac eleifend felis sem porttitor nulla, non condimentum orci est sit amet oc fermentum interdum.</p> <p>Geneva, Arial, Helvetica, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia r gravida, sapien lacus cursus tellus, ac eleifend felis eros non porttitor nulla, non condimentum orci est sit amet odio. Aei fermentum interdum.</p> <p>Tahoma, "Lucida Grande", Arial, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia nostra, lacus cursus tellus, ac eleifend felis eros non ante. Proin aliquet, n orci est sit amet odio. Aenean consectetur rutrum nibh quis congue.</p> <p>"Trebuchet MS", Arial, Helvetica, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia nostra sapien lacus cursus tellus, ac eleifend felis eros non ante. Proin i condimentum orci est sit amet odio. Aenean consectetur rutrum</p> <p>"Century Gothic", "Gill Sans", Arial, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia r gravida, sapien lacus cursus tellus, ac eleifend felis eros nor nulla, non condimentum orci est sit amet odio. Aenean cor interdum.</p>	<p>Arial, Helvetica, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia tellus, ac eleifend felis eros non ante. Proin aliquet, magni consectetur rutrum nibh quis congue. Cras porttitor ferme</p> <p>Verdana, Arial, Helvetica, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia sapien lacus cursus tellus, ac eleifend felis eros non condimentum orci est sit amet odio. Aenean conse</p> <p>Geneva, Arial, Helvetica, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia tellus, ac eleifend felis eros non ante. Proin aliquet, magni consectetur rutrum nibh quis congue. Cras porttitor ferme</p> <p>Tahoma, "Lucida Grande", Arial, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia tellus, ac eleifend felis eros non ante. Proin aliquet, magni consectetur rutrum nibh quis congue. Cras porttitor ferme</p> <p>"Trebuchet MS", Arial, Helvetica, sans-serif Class aptent taciti sociosqu ad litora torquent per conubi tellus, ac eleifend felis eros non ante. Proin aliquet, magi Aenean consectetur rutrum nibh quis congue. Cras portt</p> <p>"Century Gothic", "Gill Sans", Arial, sans-serif Class aptent taciti sociosqu ad litora torquent per conubia tellus, ac eleifend felis eros non ante. Proin aliquet, magni consectetur rutrum nibh quis congue. Cras porttitor ferme</p>
---	---

Рис. 6.3. Текст, оформленный рублеными шрифтами в операционных системах Windows (справа) и OS X (слева)

<p>"Courier New", Courier, monospace Class aptent taciti sociosqu ad litora torquent per inceptos himenaeos. Morbi tempor, leo vehicula auctor cursus tellus, ac eleifend felis eros non ante. Proin tincidunt, metus sem porttitor nulla, non condimentum Aenean consectetur rutrum nibh quis congue. Cras por</p> <p>"Lucida Console", Monaco, monospace Class aptent taciti sociosqu ad litora torquent per inceptos himenaeos. Morbi tempor, leo vehicula auctor cursus tellus, ac eleifend felis eros non ante. Proin tincidunt, metus sem porttitor nulla, non condimentum Aenean consectetur rutrum nibh quis congue. Cras por</p> <p>"COPPERPLATE LIGHT", "COPPERPLATE GOTHIC LIGHT", SERIF CLASS APTE NT TACITI SOCIOSQU AD LITORA TORQUENT PER CON HIMENAEOS. MORBI TEMPOR, LEO VEHICULA AUCTOR GRAVIDA, S ELEIFEND FELIS EROS NON ANTE. PROIN ALIQUET, MAGNA ET SOL PORTTITOR NULLA, NON CONDIMENTUM ORCI EST SIT AMET ODIO. NIBH QUIS CONGUE. CRAS PORTTITOR FERMENTUM INTERDUM.</p> <p>"Marker Felt", "Comic Sans MS", fantasy Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos him gravida, sapien lacus cursus tellus, ac eleifend felis eros non ante. Proin aliquet, m</p>	<p>"Courier New", Courier, monospace Class aptent taciti sociosqu ad litora torquent Morbi tempor, leo vehicula auctor gravida, sapi non ante. Proin aliquet, magna et sodales tinci condimentum orci est sit amet odio. Aenean cons porttitor fermentum interdum.</p> <p>"Lucida Console", Monaco, monospace Class aptent taciti sociosqu ad litora torquent Morbi tempor, leo vehicula auctor gravida, sapi non ante. Proin aliquet, magna et sodales tinci condimentum orci est sit amet odio. Aenean cons porttitor fermentum interdum.</p> <p>"Copperplate Light", "Copperplate Gothic Light", serif Class aptent taciti sociosqu ad litora torquent per conubia n auctor gravida, sapien lacus cursus tellus, ac eleifend felis ero sem porttitor nulla, non condimentum orci est sit amet odio. fermentum interdum.</p> <p>"Marker Felt", "Comic Sans MS", fantasy Class aptent taciti sociosqu ad litora torquent per conubia n auctor gravida, sapien lacus cursus tellus, ac eleifend felis e metus sem porttitor nulla, non condimentum orci est sit ame</p>
---	---

Рис. 6.4. Пример использования моношириного шрифта в операционных системах Windows (справа) и OS X (слева). Courier New — самый распространенный моноширинный шрифт, но ограничиваться только им не обязательно. Шрифт Lucida Console очень популярен в Windows и OS X, а Monaco установлен на каждом компьютере Mac

Дополнительные шрифты

На самом деле существуют буквально тысячи шрифтов, и каждая операционная система поставляется со многими из тех, которые не перечислены здесь. Тем не

менее приведу несколько шрифтов, которые очень часто встречаются на персональных компьютерах и компьютерах OS X. Возможно, вы захотите выбрать какие-то из них:

- Arial Black;
- Arial Narrow;
- Impact.

Будьте осторожны с Arial Black и Impact: у них есть только одно начертание и они не поддерживают курсивный вариант. Соответственно, если вы используете эти шрифты, не забудьте присвоить свойствам `font-weight` и `font-style` значение `normal`. В противном случае, если шрифт будет полужирным или курсивным, браузер сделает все от него зависящее, чтобы текст выглядел ужасно.

Использование веб-шрифтов

Традиционный способ применения шрифтов в каскадных таблицах стилей прост и понятен: нужно указать желаемый шрифт, воспользовавшись свойством `font-family`. Но вы ограничены теми шрифтами, которые установлены на компьютерах посетителей вашего сайта. На этот случай, как уже ранее упоминалось, все основные браузеры поддерживают использование веб-шрифтов. При этом браузеры фактически загружают шрифт с веб-сервера и используют его для отображения текста на веб-странице.

Процедура использования веб-шрифтов довольно проста. Вам потребуется лишь:

- правило `@font-face`, отвечающее за сообщение браузеру как имени шрифта, так и пути, по которому его нужно загрузить. Вскоре вы узнаете, как это правило используется, а сейчас имейте в виду, что с его помощью браузеру сообщается о необходимости загрузить шрифт;
- свойство `font-family`, которое применяется с веб-шрифтами точно так же, как и со шрифтами, установленными на компьютере. Другими словами, если есть правило `@font-face`, инструктирующее браузер загрузить шрифт, вы можете назначить этот шрифт любому стилю CSS с помощью свойства `font-family`.

Теоретически веб-шрифты использовать не трудно. Но если вдаваться в подробности, то для их правильного применения нужно понимать суть некоторых специфических требований.

ПРИМЕЧАНИЕ

Довольно простой способ использования веб-шрифтов предлагается компанией Google. Вскоре мы рассмотрим, в чем именно он заключается.

Типы файлов шрифтов

Не верится, но браузер Internet Explorer поддерживал веб-шрифты, начиная с 5-й версии программы (выпущенной более 15 лет назад!). Но эта поддержка требовала использования весьма специфического и сложного способа создания типографиче-

ского форматирования. Иными словами, нельзя было взять обычный шрифт с жесткого диска компьютера, выложить его на веб-сервер и считать, что дело сделано. Сначала шрифт нужно было преобразовать в формат EOT (Embedded Open Type — встраиваемый формат Open Type). И такое положение вещей сохранялось вплоть до версии Internet Explorer 8.

Для веб-шрифтов используются и другие форматы шрифтов, причем разные браузеры поддерживают различные форматы. Чтобы указанными шрифтами могло любоваться как можно большее количество посетителей вашего сайта, нужно предоставлять эти шрифты в различных форматах (как это сделать, вы вскоре узнаете).

В следующем списке приведены различные типы шрифтов и перечислены браузеры, которые с ними работают.

- **EOT.** Шрифты Embedded Open Type поддерживаются только в браузере Internet Explorer. Чтобы преобразовать обычный шрифт в формат EOT, потребуется специальное программное обеспечение, но сделать это можно и на таких сайтах, как FontSquirrel.
- **True Type и Open Type.** Если заглянуть в папку Fonts вашего компьютера, то там наверняка найдутся файлы шрифтов с расширениями .ttf (True Type) или .otf (Open Type). Шрифты этих форматов чаще всего используются на компьютерах. Их можно задействовать для обработки текста и вывода его на экран, а также для веб-страниц. Ранее это были одни из самых используемых веб-шрифтов, и они все еще поддерживаются большинством браузеров. Однако им на смену пришел формат WOFF.
- **WOFF.** Формат Web Open Font был разработан специально для Всемирной паутины. WOFF-шрифты, по сути, являются сжатой версией шрифтов TrueType или Open Type. Это означает, что WOFF-шрифты обычно имеют меньший размер файла и загружаются быстрее других форматов. Формат WOFF имеет также широкую поддержку со стороны браузеров, включая Internet Explorer 9 и выше, Firefox, Chrome, Safari, Opera, а также браузеры в операционной системе iOS, Blackberry и Android версии 4.4 и выше.

ПРИМЕЧАНИЕ

WOFF2 — последняя версия формата WOFF, обеспечивающего на 30 % более эффективное сжатие, благодаря чему файлы шрифтов становятся меньшего размера и загружаются быстрее. Однако на момент написания этой книги данный формат не поддерживался браузерами Internet Explorer, Edge, Safari и Opera Mini (tinyurl.com/pucouja).

- **SVG.** Scalable Vector Graphic (формат масштабируемой векторной графики) сам по себе не является форматом шрифта. По сути, это способ создания *векторной графики* (то есть изображений, которые могут масштабироваться без потери качества). Поддержка SVG-шрифтов ограничена куда существеннее. SVG-формат поддерживается только браузером Safari. Другой проблемой, связанной с SVG, является то, что данный формат создает файлы размером в два раза больше, чем TrueType-файлы, и в три раза больше, чем WOFF-файлы. Единственное реальное преимущество SVG в том, что только этот формат шрифта поддерживают устаревшие версии операционной системы iOS с браузером Safari версии 4.1 или ниже, а также браузеры в операционной системе

Android 4.3 и ниже. Если вы не предполагаете просмотр ваших веб-страниц на этих устройствах, не используйте формат SVG.

Вам не нужно выбирать только один формат шрифтов, игнорируя все остальные, не поддерживающие его браузеры. Совсем скоро вы узнаете, что можно (и, как правило, нужно) указывать несколько форматов, позволяя браузеру выбирать только тот, который он поддерживает. Кроме того, вы можете загрузить шрифт, который уже был преобразован в эти четыре формата, или даже преобразовать обычный шрифт TrueType в несколько форматов.

ПРИМЕЧАНИЕ

Каждый файл шрифта содержит только одно начертание. Иначе говоря, если вам нужны и полужирный и курсивный начертания, следует загрузить отдельные файлы шрифта для каждого начертания. Некоторые шрифты, обычно декоративные, могут существовать только в одном начертании и больше подходят для заголовков или прочего текста, где не нужно курсивное или полужирное форматирование. Более полная информация рассмотрена далее в этой главе.

Правовые вопросы использования веб-шрифтов

Вторым препятствием для использования веб-шрифтов являются правовые вопросы. Эти шрифты с целью получения средств к существованию создаются и продаются как отдельными разработчиками, так и компаниями. После загрузки шрифта TrueType на ваш сервер для его использования посетителями при просмотре вашего сайта любой человек может скачать шрифт и пользоваться им на своем сайте или в установленных на его компьютере программах допечатной подготовки или текстовых редакторах. Большинству компаний, занимающихся созданием шрифтов, не нравится нелегальное использование плодов их труда, поэтому многие шрифты имеют лицензии, которые конкретно запрещают их использование во Всемирной сети.

Иными словами, даже если вы приобрели шрифт в корпорации Adobe, его нельзя использовать на вашем сайте. Многие компании, создающие шрифты, в настоящее время предлагают различные виды лицензий (по разным ценам), чтобы разрешить использование своих шрифтов во Всемирной паутине. Это касается даже шрифтов, поставляемых вместе с вашим компьютером. Вам разрешается использовать их с программами, установленными на компьютере, но может быть не разрешено помещать файлы тех же самых шрифтов на свой веб-сервер для использования их в качестве веб-шрифтов. Если вы не знаете, разрешено ли применение шрифта в Сети, лучше воздержаться от его использования и подобрать шрифт, который в ней может применяться легально.

ПРИМЕЧАНИЕ

Чтобы решить возникающие правовые вопросы, можно воспользоваться такими службами шрифтов, как Google Fonts или TypeKit, коммерческой службой веб-шрифтов корпорации Adobe (о которой будет рассказано в одной из следующих врезок).

Поиск веб-шрифтов

При поиске веб-шрифтов приходится сталкиваться с двумя проблемами: поиском таких шрифтов, которые легально можно использовать во Всемирной паутине, и по-

иском форматов шрифтов, которые понадобятся посетителям вашего сайта (EOT, WOFF, TrueType и SVG). Хотя некоторые компании стали предлагать лицензии, разрешающие использование приобретаемых шрифтов во Всемирной паутине, существует довольно широкий ассортимент бесплатных веб-шрифтов. Ниже представлены лишь некоторые из множества источников бесплатных веб-шрифтов.

- tinyurl.com/olkcst6. Представленный группой дизайнеров, этот сайт был одним из первых, предложивших бесплатное использование созданных вручную шрифтов во Всемирной паутине. Созданный ими шрифт League Gothic широко применяется на сайтах.
- exljbris.com. Предоставляет классические бесплатные шрифты: Museo, Museo Sans и Museo Slab.
- openfontlibrary.org. Доступно более 700 бесплатных шрифтов (на момент написания книги), и все они могут использоваться на ваших сайтах.
- fontquirrel.com. Весьма примечательный сайт в мире веб-шрифтов, предлагающий более тысячи шрифтов. В дополнение к шрифтам ресурс предлагает инструментарий для преобразования шрифта TrueType или Open Type в другие форматы, включая EOT, SVG и WOFF. Порядок использования этого инструментария будет рассмотрен в следующем разделе.
- google.com/webfonts. Компания Google предоставляет простой и бесплатный способ включения веб-шрифтов в ваши сайты. Порядок использования этой службы будет подробно рассмотрен чуть позже.

Преобразование форматов шрифтов

Большинство сайтов, предлагающих бесплатные шрифты, предоставляют шрифт в единственном формате, обычно TrueType (.ttf) или Open Type (.otf). Но TrueType и Open Type некоторыми браузерами не поддерживаются. Кроме того, формат WOFF поддерживается всеми современными браузерами и имеет преимущество, так как файлы в этом формате меньше по размеру, чем TrueType. В настоящий момент новый формат WOFF2 поддерживается лишь несколькими браузерами, но в скором времени ситуация должна измениться.

Существует несколько моментов, о которых надо помнить при использовании веб-шрифтов на вашем сайте. Во-первых, вы можете быть консервативными и использовать шрифты, которые поддерживаются старыми браузерами и мобильными устройствами. То есть добавить шрифт в формате EOT (для браузера Internet Explorer 8 и версии ниже), шрифты TrueType для устаревших браузеров, шрифты WOFF для современных браузеров и шрифты SVG для старых смартфонов и планшетов.

Кроме того, поскольку формат WOFF поддерживается всеми современными браузерами, вы можете использовать *только* его. В этом случае старые браузеры при просмотре страниц будут пропускать шрифт WOFF и использовать вместо него следующий в стеке шрифтов (см. предыдущий раздел).

И наконец, вы можете использовать только шрифты EOT и WOFF. Файл EOT предназначен для браузера Internet Explorer 8, который все еще применяется. Файл WOFF будут использовать остальные современные браузеры (включая браузер IE 9 и более поздние версии программы).

Для продолжения занятия вам необходимо создать файлы шрифтов. Для этих целей ресурс Font Squirrel предоставляет очень полезное средство, помогающее преобразовывать файлы шрифтов в нужные форматы. Инструмент Webfont Generator, доступный по адресу tinyurl.com/b56z7zq, является простым средством создания не только нужных шрифтов, но и пробных HTML-файла и таблицы стилей.

Чтобы воспользоваться им, выполните следующие действия.

1. Найдите шрифт TrueType (.ttf) или Open Type (.otf).

Воспользуйтесь одним из сайтов, перечисленных в предыдущем разделе, или найдите шрифт на своем компьютере. Убедитесь, что этот шрифт лицензирован для использования в качестве веб-шрифта. Если он не лицензирован или вы не уверены в его лицензировании, найдите другой шрифт.

2. Перейдите к инструменту Webfont Generator по адресу tinyurl.com/b56z7zq.

Это простая страница, на которой находится лишь несколько настроек (рис. 6.5).

3. Нажмите кнопку Upload fonts (Выгрузить шрифты) (см. рис. 6.5, 1).

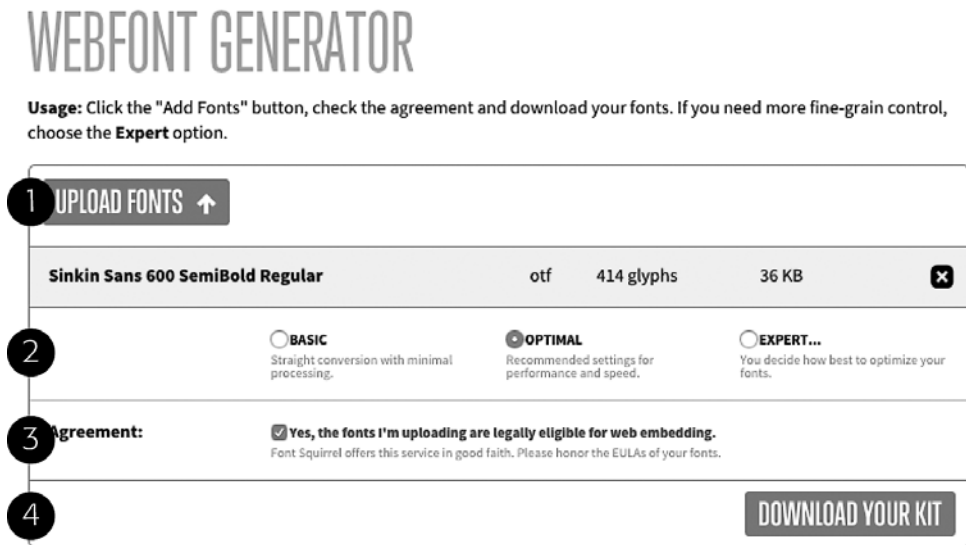


Рис. 6.5. Окно генератора шрифтов

Браузер откроет диалоговое окно **Выгрузка файла** (Select files).

4. Выберите один или несколько шрифтов на жестком диске компьютера и нажмите кнопку **Открыть** (Open).

Файл (файлы) будет загружен на сервер fontsqurrel.com.

5. Выберите вариант преобразования (см. рис. 6.5, 2).

- **Basic** (Простое) — будет выполнено только преобразование шрифта в форматы EOT, WOFF и SVG.

- **Optimal** (Оптимальное) — более удачный выбор, поскольку это не только приведет к преобразованию шрифта, но и будут выполнены другие действия по оптимизации форматов для повышения производительности и скорости загрузки.
- **Expert** (Экспертное) позволит провести тонкую настройку всех нюансов преобразования. Например, вы сможете настроить *набор символов*. Иными словами, из шрифта можно исключить определенные, неиспользуемые вами символы, например точку с запятой, восклицательный знак, букву Q или буквы с диакритическими знаками (такие как «ь», «й» или «х»). Кроме того, вы можете использовать экспертный режим, чтобы получить только определенные форматы шрифтов, такие как WOFF или WOFF2.

Пока вы не станете понимать смысл своих действий, придерживайтесь варианта настройки **Optimal** (Оптимальное). При выборе этого режима будут сгенерированы все необходимые вам форматы шрифтов и выполнены другие настройки, чтобы шрифты быстрее загружались и лучше выглядели на экране. А для полного контроля процесса выберите вариант **Expert** (Экспертное), чтобы отобразить перечень элементов управления, которыми можно воспользоваться для изменения настроек процесса создания шрифтов.

По всей вероятности, вам потребуется доступ ко всем символам, поэтому лучше остановить выбор на варианте **Optimal** (Оптимальное).

6. Установите флажок, обозначенный пунктом 3 на рис. 6.5.

Как уже упоминалось, шрифты — это интеллектуальная собственность и их простое размещение на веб-сервере недопустимо. Убедитесь в том, что ваши шрифты могут использоваться во Всемирной паутине, прежде чем устанавливать флажок.

7. Нажмите кнопку **Download Your Kit** (Загрузить свой комплект) (см. рис. 6.5, 4).

В зависимости от количества преобразуемых шрифтов и их сложности, процесс может занять некоторое время. Серверу fontquirrel.com нужно получить шрифты и преобразовать их в каждый выбранный формат. Как только все будет готово, вы сможете скачать архив, содержащий файлы различных форматов шрифтов, а также демонстрационные HTML- и CSS-файлы. Наиболее важными, конечно же, являются файлы шрифтов с расширениями `.eot`, `.ttf`, `.woff` и `.svg`, которые вы хотите использовать.

После получения шрифтов настало время научиться использовать их с правилом `@font-face`.

Использование правила `@font-face`

Сначала скопируйте шрифты в папку на компьютере, где хранятся файлы для вашего сайта. Многие веб-разработчики создают для этого специально выделенную папку в корневом каталоге сайта, которая носит имя `fonts`, `_fonts` или `webfonts`. В качестве альтернативы, если есть папка для CSS-файлов, вы можете поместить файлы шрифтов в эту папку. В принципе, неважно, куда вы поместите эти файлы на вашем сайте, но вышеописанное правило поможет навести в нем порядок.

Секретом использования веб-шрифтов является *правило* @font-face. Эта команда по своей сути присваивает шрифту имя и сообщает браузеру, где найти файл шрифта для его загрузки. Правило @font-face помещается в вашу таблицу стилей так же, как и обычный стиль. Предположим, вы используете шрифт League Gothic. У вас на сайте в папке fonts есть шрифт True Type с именем файла League_Gothic-webfont.ttf. Нужно инструктировать браузер загрузить этот шрифт, добавив правило @font-face в таблицу стилей:

```
@font-face {
  font-family: "League Gothic";
  src: url('fonts/League_Gothic-web font.woff');
}
```

Первое свойство, font-family, вы уже встречали, но здесь у него другое предназначение. При использовании внутри правила @font-face свойство font-family присваивает шрифту имя. Затем это имя задействуется, когда соответствующий шрифт нужно применить к стилю. Предположим, нужно использовать шрифт League Gothic для всех абзацев на странице. Для этого можно будет указать следующий стиль:

```
p {
  font-family: "League Gothic";
}
```

ПРИМЕЧАНИЕ

Для каждого шрифта, который нужно использовать, применяется отдельное правило @font-face. Если у вас задействованы три шрифта, понадобятся три правила @font-face. Лучше всего использовать их сгруппированными вместе и помещать в верхнюю часть таблицы стилей, чтобы браузер сразу же мог начать загрузку необходимых файлов шрифтов.

Второй атрибут, свойство src, сообщает браузеру, где следует искать файл шрифта на сервере. Путь к файлу шрифта помещается в кавычки внутрь атрибута url(). Путь к шрифту указывается точно так же, как и все другие пути, например к изображениям, ссылкам и JavaScript-сценариям. Предположим, у вас есть таблица стилей в папке с именем _styles и есть файл шрифта my_font.ttf в папке _fonts. Обе папки находятся в корневом каталоге вашего сайта. Следовательно, путь из таблицы стилей к файлу шрифта будет выглядеть так: ../_fonts/my_font.ttf. И тогда для этого шрифта нужно будет написать следующее правило @font-face:

```
@font-face {
  font-family: "Название шрифта";
  src: url('../_fonts/my_font.woff');
}
```

Возможно, вы заметили, что в приведенном выше примере фигурирует только один файл шрифта в формате WOFF. Это сделано для упрощения демонстрации общего принципа работы правила @font-face. Как уже говорилось, оно позволяет указать несколько файлов шрифтов различных форматов.

Если вы хотите обеспечить поддержку старых браузеров, смартфонов и планшетов, то синтаксис должен быть немного усложнен. Предположим, на сайте нужно использовать все разновидности форматов шрифта League Gothic. Тогда показанный выше код нужно переписать следующим образом:

```
@font-face {
  font-family: 'League Gothic';
  src: url('fonts/League_Gothic-webfont.eot');
  src: url('fonts/League_Gothic-webfont.eot?#iefix') format('embeddedopentype'),
       url('fonts/League_Gothic-webfont.woff2') format('woff2'),
       url('fonts/League_Gothic-webfont.woff') format('woff'),
       url('fonts/League_Gothic-webfont.ttf') format('truetype'),
       url('fonts/League_Gothic-webfont.svg') format('svg');
}
```

Код неоправданно усложнен из-за несовместимости браузера Internet Explorer. Давайте в нем разберемся.

- Строка 2 сохранила свой прежний вид. Свойство `font-family` предоставляет имя шрифта, это то же самое имя, которое вы будете использовать, применяя шрифт к своим стилям.
- Строка 3 предназначена для программы Internet Explorer 9, но только при ее работе в режиме совместимости — когда IE 9 работает как IE 8. Это странное свойство было добавлено в IE 9, чтобы сайты, адаптированные под недостатки IE 8 и более ранних версий, не теряли в дизайне и в IE 9. Пользователь должен будет преднамеренно переключиться в режим совместимости в IE 9, поэтому лучше, наверное, оставить эту настройку.
- Строка 4 начинается со второго свойства `src`, которое в соответствии с правилом `@font-face` может указывать на несколько шрифтов. Первым вновь указан шрифт `.eot`, но на этот раз к расширению файла добавлена строка `?#iefix`. Это сделано в попытке приспособиться к дополнительным недостаткам Internet Explorer, на этот раз дело касается версий 6–8 браузера. Если после расширения `.eot` не будет указано это значение, то шрифт может неправильно отобразиться в Internet Explorer 8 и более ранних версиях программы.

После URL-адреса можно также заметить новый фрагмент кода:

```
format('embedded-opentype')
```

Он указывает формат шрифта и добавляется после каждого URL-адреса различных форматов шрифтов.

- Строки 5–8 определяют дополнительные форматы шрифтов. Приведено всего одно свойство (`src`) на нескольких строках для упрощения чтения кода. Для каждого формата шрифта, указанного в свойстве `src`, добавляется URL-адрес, атрибут `format` и запятая (для всех, за исключением последнего шрифта):

```
url('fonts/League_Gothic-web font.woff') format('woff'),
```

ПРИМЕЧАНИЕ

В конце списка файлов свойства `src` добавляется точка с запятой, чтобы обозначить его конец (строка 7 в показанном выше примере). Об этой завершающей точке с запятой забывать нельзя, иначе правило `@font-face` работать не будет.

Даже если браузер поддерживает различные форматы шрифтов (например, программа Chrome может использовать шрифты WOFF2, WOFF, TrueType и SVG), он не станет загружать все файлы шрифтов. Вместо этого по мере считывания

списка форматов шрифтов браузер загружает только первый поддерживаемый шрифт. Иначе говоря, если браузеру Chrome попадется показанный выше код, он пропустит файл с расширением `.eot`, поскольку этот формат им не поддерживается, а загрузит файл с расширением `.woff`. Затем он полностью проигнорирует файлы форматов TrueType и SVG. Из этого следует, что порядок, в котором перечислены шрифты, играет важную роль. WOFF в большинстве случаев предпочтительнее, поскольку файлы этого формата меньше по размеру и загружаются быстрее. WOFF2 — это оптимизированная версия формата WOFF, которая обеспечивает использование файлов меньшего размера, но в настоящий момент поддерживается не всеми браузерами. Файлы формата SVG намного больше по размеру и поддерживаются только браузером Safari. Подводя итог вышесказанному, необходимо отметить следующее: для того чтобы браузеры сначала загружали файлы шрифтов меньших размеров, нужно убедиться, что шрифты перечислены в следующем порядке: `.eot`, `.woff2`, `.woff`, `.ttf` и `.svg`.

Если вы хотите обеспечить поддержку браузера Internet Explorer 8 и более современных браузеров, то можете использовать только форматы EOT, WOFF2 и WOFF. Фактически, если вас не интересует браузер IE 8 (см. врезку «Стоит ли волноваться насчет Internet Explorer 8?» в главе 1), вы можете использовать только шрифты формата WOFF. Следующие два листинга демонстрируют альтернативные способы применения правила `@font-face`:

```
@font-face {
  font-family: 'League Gothic';
  src: url('fonts/League_Gothic-webfont.eot?#iefix') format('embeddedopentype'),
       url('fonts/League_Gothic-webfont.woff2') format('woff2'),
       url('fonts/League_Gothic-webfont.woff') format('woff');
}
```

Если вас не интересует браузер Internet Explorer 8 и его более ранние версии, вы можете упростить событие следующим образом:

```
@font-face {
  font-family: 'League Gothic';
  src: url('fonts/League_Gothic-webfont.woff2') format('woff2'),
       url('fonts/League_Gothic-webfont.woff') format('woff');
}
```

В этой книге используется последний вариант, предназначенный только для современных браузеров. Но, если вы предполагаете, что посетители вашего сайта будут работать с Internet Explorer 8, используйте версию правила, включающую файлы `.eot`. Аналогичным образом, если вам необходима поддержка старых смартфонов и планшетов, перечисленных выше, вам лучше использовать более сложный синтаксис, показанный на с. 155.

ПРИМЕЧАНИЕ

Не стоит волноваться, что, если ваш сайт не содержит веб-шрифты, предназначенные для старых браузеров (таких форматов, как `.svg`, `.ttf`, или `.eot`), то посетители не смогут просмотреть его контент. Помните, что при определении шрифтов вы используете стек шрифтов, поэтому браузеры, не поддерживающие указанный в правиле шрифт, будут применять похожую гарнитуру. В большинстве случаев ваш сайт будет выглядеть одинаково, независимо от браузера.

Создание стилей с применением веб-шрифтов

Самым сложным в работе с веб-шрифтами является получение файлов шрифтов в нужном формате и настройка правила `@font-face`. После того как все это будет готово, веб-шрифты используются точно так же, как и рассмотренные ранее шрифты, предустановленные на компьютере (устройстве). Иначе говоря, при создании стиля применяется свойство `font-family` и указывается имя шрифта, которое добавлено в правиле `@font-face`. Например, в предыдущем коде в правиле `@font-face` шрифту было присвоено имя `League Gothic` (строка 2). Это имя и нужно использовать, применяя данный шрифт к стилю. Чтобы для всех заголовков `h1` устанавливался шрифт `League Gothic`, можно создать следующий стиль:

```
h1 {  
  font-family: 'League Gothic';  
  font-weight: normal;  
}
```

Обратите внимание на новое свойство `font-weight`. Обычно браузеры отображают содержимое элементов `h1` полужирным шрифтом. Большинство из них искусственно сделают шрифт полужирным, когда требуется такая версия шрифта. В результате получается некрасивое полужирное начертание. Присваивание свойству `font-weight` значения `normal` инструктирует браузер использовать шрифт `League Gothic` таким, какой он есть, и пресекает попытки сделать его полужирным. Более подробно работа с начертаниями применительно к веб-шрифтам будет рассмотрена в следующем разделе.

Рекомендуется также включить список резервных предустановленных шрифтов на тот случай, если браузер не сможет загрузить веб-шрифт. Здесь можно воспользоваться ранее рассмотренной технологией. Например:

```
h1 {  
  font-family: 'League Gothic', Arial, sans-serif;  
  font-weight: normal;  
}
```

СОВЕТ

На веб-странице можно также задействовать шрифты, содержащие различные символы и значки. Иначе говоря, вместо создания, к примеру, рисунка предупреждения и помещения его в текст абзаца можно воспользоваться правилом `@font-face` для загрузки шрифта, содержащего значок аналогичного текстового знака, и применить соответствующий символ. Тем не менее предварительно проверьте стандартные символы Unicode или задумайтесь об использовании SVG-графики для представления значка. Подробнее тема освещается на сайте tinyurl.com/oaclzaq.

Форматирование полужирным и курсивным начертаниями

Обычные шрифты, установленные на компьютере, включают варианты начертания и веса (насыщенности), поэтому при применении в HTML-коде элемента `strong` браузер использует полужирное начертание такого шрифта, а при использовании элемента `em` — курсивное. Если же указывать эти два элемента совместно,

вы увидите полужирную курсивную версию шрифта. Это совершенно другие шрифты, содержащиеся в других файлах шрифтов. При исходном способе применения шрифтов на веб-страницах (который был рассмотрен ранее) вам не нужно волноваться насчет этих других шрифтов, потому что браузер автоматически воспользуется правильной версией.

А вот в случае с веб-шрифтами для каждого варианта шрифта вам понадобится отдельный файл. Итак, для основного текста веб-страницы нужны как минимум обычная версия шрифта, полужирная, курсивная, а также комбинированная полужирная и курсивная версия. И об этом нужно помнить при подборе шрифта для своего сайта, поскольку у некоторых шрифтов доступна только версия с определенным начертанием и нет, к примеру, курсивной версии. Такой шрифт пригодится для заголовков, но применять его для форматирования длинных текстовых абзацев, где, скорее всего, будут встречаться слова в полужирном и курсивном начертании, нет смысла. Кроме того, для каждого варианта шрифта вы должны создать отдельное правило `@font-face`.

При работе с курсивными и полужирными начертаниями веб-шрифтов доступно два варианта действий. Один из них проще реализовать, но он не работает в программе Internet Explorer 8 или более ранней версии (или же IE 9 в режиме совместимости), другой требует больших трудозатрат, но работает и в старых версиях браузера IE.

Простой способ добавления полужирного и курсивного начертания

Самый простой способ добавления полужирного и курсивного начертаний ваших шрифтов заключается в добавлении в правило `@font-face` свойств `font-weight` и `font-style`. Обычно свойство `font-weight` требует от браузера отобразить шрифт в полужирном (`bold`), обычном (`normal`) варианте или в одном из нескольких других вариантов веса, а свойство `font-style` управляет отображением шрифта в курсивном (`italic`) или обычном (`normal`) начертании. Но при использовании в правиле `@font-face` свойство `font-style` требует от браузера применить шрифт, когда стиль запрашивает конкретный вариант шрифта.

Предположим, у вас есть шрифт PTSans. Вы начинаете с обычной — не жирной и не курсивной — версии шрифта. Различные начертания шрифта начинаются с PTSansRegular. В таблицу стилей нужно добавить следующее правило `@font-face`:

```
@font-face {
  font-family: 'PTSans';
  src: url('PTSansRegular.woff2') format('woff2'),
       url('PTSansRegular.woff') format('woff'),
  font-weight: normal;
  font-style: normal;
}
```

ПРИМЕЧАНИЕ

В приведенном выше примере не используется шрифт в формате EOT, необходимый браузеру Internet Explorer 8, поскольку этот способ в нем не работает. Код из примера также игнорирует старые версии браузеров для компьютеров и мобильных устройств. Однако он прекрасно функционирует в современных браузерах, например Internet Explorer 9.

Обратите внимание на следующее:

- для семейства шрифтов во второй строке показанного выше кода используется общее имя — `PTSans`, вместо имени, указанного для файла шрифта, — `PTSansRegular`;
- свойству `font-weight` присвоено значение `normal`, поскольку эта версия шрифта не является полужирной (строка 8);
- свойству `font-style` присвоено значение `normal`, поскольку это не курсивная версия шрифта (строка 9).

ПРИМЕЧАНИЕ

В приведенных здесь примерах кода предполагается, что файлы шрифтов `PTSansRegular.eot`, `PTSansBold.eot` и т. д. находятся в той же папке, что и таблица стилей. Если бы использовались разные папки, пришлось бы изменить URL-адрес для точного указания расположения файлов шрифтов относительно таблицы стилей.

Теперь предположим, что у вас имеется курсивная версия шрифта, у которой имя файла начинается с `PTSansItalic`. Его следует добавить в таблицу стилей:

```
@font-face {
  font-family: 'PTSans';
  src: url('PTSansItalic.woff2') format('woff2'),
       url('PTSansItalic.woff') format('woff');
  font-weight: normal;
  font-style: italic;
}
```

В строке 2 используется то же самое имя `font-family` — `PTSans`. Но значение свойства `font-style` изменилось на `italic` (строка 6). Тем самым браузеру сообщается, что указанный шрифт является курсивной версией шрифта `PTSans`. Нужно также добавить подобные правила `@font-face` для полужирной, а также полужирной/курсивной версии:

```
@font-face {
  font-family: 'PTSans';
  src: url('PTSansBold.woff2') format('woff2'),
       url('PTSansBold.woff') format('woff');
  font-weight: bold;
  font-style: normal;
}
```

```
@font-face {
  font-family: 'PTSans';
  src: url('PTSansBoldItalic.woff2') format('woff2'),
       url('PTSansBoldItalic.woff') format('woff');
  font-weight: bold;
  font-style: italic;
}
```

Иначе говоря, для охвата всех вариантов полужирного, курсивного и обычного начертаний шрифта вам нужны четыре правила `@font-face`: обратите внимание на то, что во всех случаях используется одно и то же имя шрифта в строке `font-family`,

а изменяются только значения свойств `src` (указывающие на разные файлы), `font-weight` и `font-style`.

Преимущество этого метода состоит в том, что к тексту можно применить обычный шрифт, добавить в HTML-код элементы `em` и `strong` и позволить браузеру взять на себя вопрос выбора файла шрифта для загрузки и использования. В данном примере, если нужно применить шрифт PTSans ко всем абзацам, достаточно добавить следующий стиль в таблицу стилей:

```
p {
  font-family: PTSans;
}
```

Затем нужно разметить абзацы HTML-элементами. Например, так:

```
<p>Когда я был моложе, я мог запомнить <em>все</em>, что было и чего <strong>не</strong> было. -- <strong><em>Марк Твен</em></strong></p>
```

Когда браузер прочитает таблицу стилей (с четырьмя правилами `@font-face` и стилем элемента `p`), он отформатирует большую часть абзаца шрифтом PTSansRegular. Для слова «все», заключенного в теги элемента `em`, будет использоваться шрифт PTSansItalic, а для слова «не» внутри тегов элемента `strong` — шрифт PTSansBold. Для слов «Марк Твен», окруженных тегами элементов `strong` и `em`, будет использоваться шрифт PTSansBoldItalic.

Эти правила работают и для заголовков. Если вы создаете стиль для форматирования всех элементов `h1` с использованием шрифта PTSans, то он может иметь такой вид:

```
h1 {
  font-family: PTSans;
}
```

При наличии этого стиля браузер будет фактически использовать полужирную версию шрифта PT-Sans, поскольку заголовки обычно отображаются полужирным шрифтом. (При использовании такой технологии с привлечением нескольких вариантов шрифтов уже не нужно, как раньше, добавлять фрагмент `font-weight: normal;`.)

Браузер Internet Explorer 8 и его более ранние версии не поддерживают этот метод и станут использовать для всего текста шрифт PTSansRegular. Для элементов `em` и `strong` Internet Explorer будет создавать искусственные (псевдо) курсивный и полужирный начертания шрифтов, то есть он начнет наклонять шрифт PTSansRegular на экране для получения курсива и делать шрифт PTSansRegular толще для получения полужирного варианта. Получающиеся в результате сгенерированные компьютером полужирный и курсивный варианты обычно не очень хорошо смотрятся.

Поддержка полужирного и курсивного начертаний в Internet Explorer 8

Если вы все еще учитываете поддержку Internet Explorer 8 (или более раннюю версию), то предыдущее решение по поводу полужирного и курсивного начертаний не будет работать так же хорошо. Поэтому для начала нужно, как и ранее, создать четыре пра-

вила `@font-face`, по одному для каждого варианта шрифта. Но вместо того, чтобы задавать для них одно и то же имя семейства в свойстве `font-family` (например, `PTSans`), каждому из них присваиваем уникальное имя (`PTSansRegular`, `PTSansItalic` и т. д.). Иными словами, четыре правила `@font-face` нужно переписать следующим образом:

```
@font-face { font-family: 'PTSansRegular';
  src: url('PTSansRegular.eot?#iefix') format('embedded-opentype'),
       url('PTSansRegular.woff2') format('woff2'),
       url('PTSansRegular.woff') format('woff');
}

@font-face { font-family: 'PTSansItalic';
  src: url('PTSansItalic.eot?#iefix') format('embedded-opentype'),
       url('PTSansItalic.woff2') format('woff2'),
       url('PTSansItalic.woff') format('woff');
}

@font-face { font-family: 'PTSansBold';
  src: url('PTSansBold.eot?#iefix') format('embedded-opentype'),
       url('PTSansBold.woff2') format('woff2'),
       url('PTSansBold.woff') format('woff');
}

@font-face { font-family: 'PTSansBoldItalic';
  src: url('PTSansBoldItalic.eot?#iefix') format('embedded-opentype'),
       url('PTSansBoldItalic.woff2') format('woff2'),
       url('PTSansBoldItalic.woff') format('woff');
}
```

Обратите внимание, что у каждого правила `@font-face` имеется собственное название семейства, которое соответствует варианту шрифта: `PTSansRegular`, `PTSansItalic`, `PTSansBold` и `PTSansBoldItalic`.

Кроме того, обратите внимание на то, что свойства `font-weight` и `font-style`, которые использовались ранее, здесь уже отсутствуют за ненадобностью.

Труднее дело будет обстоять с применением шрифта. В предыдущем примере свойство `font-family` просто применялось к стилю:

```
p {
  font-family: PTSans;
}
```

Теперь вам придется использовать различные имена шрифтов для разных элементов, для `p` — имя обычного шрифта, для `em` — курсивного, для `strong` — полужирного, а чтобы справиться со случаем применения полужирного/курсивного начертания, придется составить селектор потомков. Иначе говоря, чтобы могли работать различные варианты шрифта `PTSans`, нужно создать четыре стиля, содержащие довольно много строк кода:

```
p {
  font-family: PTSansRegular;
  font-size: 48px;
  font-style: normal;
```

```

    font-weight: normal;
}

p em {
    font-family: PTSansItalic;
    font-style: normal;
    font-weight: normal;
}

p strong {
    font-family: PTSansBold;
    font-style: normal;
    font-weight: normal;
}

p strong em, p em strong {
    font-family: PTSansBoldItalic;
    font-weight: normal;
    font-style: normal;
}

```

В первую очередь обратите внимание на четыре стиля:

- для селектора `p` используется шрифт `PTSansRegular`;
- `p em` представляет собой селектор потомков, используемый для элемента `em`, находящегося внутри абзаца `p`, — применяется шрифт `PTSansItalic`;
- `p strong` представляет собой еще один селектор потомков, который применяет шрифт `PTSansBold` к элементу `strong` внутри абзаца;
- и наконец, мы имеем дело с групповым селектором, составленным из двух селекторов потомков. Первый применяется к элементу `em`, находящемуся внутри `strong` в пределах абзаца `p`. Второй применяется к элементу `strong` внутри `em` в пределах абзаца `p`. Вам нужны оба этих селектора, потому что вы можете вложить элементы `em` внутрь `strong` и наоборот.

В результате может получиться такой HTML-код:

```

<p>
  <em><strong>Привет!</strong></em>
  Я говорю с
  <strong><em>тобой</em></strong>
</p>

```

Одиночный селектор `p strong em` неприменим к слову «Привет!», поскольку оно окружено тегами элемента `strong`, который находится в элементе `em`.

ПРИМЕЧАНИЕ

В языке HTML5 элементы `b` (для полужирного начертания) и `i` (для курсивного) восстановили свои позиции. Их можно использовать исключительно в оформительских целях, когда нужно, чтобы шрифт текста стал полужирным или курсивным, но без смысловой нагрузки данного форматирования. Например, курсивом часто пишут названия книг, поэтому в данном случае рекомендуется указывать элемент `i`:

```
<i>Большая книга CSS</i>
```

А при использовании элемента `em` текст будет акцентирован, что заставит программу экранного доступа прочитать его громче, не так, как обычный текст. В любом случае, если вы намерены использовать элементы `b` и `i`, не забудьте создать стили для курсивного и полужирного вариантов шрифта (волноваться об этом приходится только в том случае, если речь идет об Internet Explorer 8, для которого нужно обеспечить безопасный способ указания курсива, а первый, рассмотренный ранее, метод таких мер не предусматривает).

Еще нужно обратить внимание на необходимость присваивания во всех этих стилях значения `normal` свойствам `font-weight` и `font-style`. Если этого не сделать, многие браузеры (не только Internet Explorer) будут пытаться утолщать полужирный шрифт и наклонять курсивную версию шрифта.

Эта технология поддержки полужирного и курсивного начертаний, несомненно, требует большого объема работы, который существенно возрастает, если на одном сайте задействовано несколько шрифтов с полужирным и курсивным вариантами. Какой из технологий следует воспользоваться, зависит от того, насколько важна для вас поддержка Internet Explorer 8. На момент написания книги он все еще использовался на 2–11 % компьютеров (по сведениям tinyurl.com/ooevwyr и netmarketshare.com).

ПРИМЕЧАНИЕ

Для решения проблемы поддержки полужирных и курсивных начертаний шрифтов в программе Internet Explorer 8 можно применить и другой подход. Попробуйте воспользоваться первым методом и посмотрите, как выглядят страницы в браузере IE 8. Некоторые шрифты, обычно из числа рубленых, не всегда выглядят так уж плохо, когда IE применяет к ним методы придания псевдополужирного и псевдокурсивного начертания. Может оказаться, что разница не так уж и заметна, и можно воспользоваться первым методом, не испытывая при этом особых проблем. Следует также помнить, что количество устройств, на которых используется IE 8, постоянно снижается по мере приобретения пользователями новых компьютеров, перехода на другие браузеры или же обновления операционных систем.

Использование службы Google Fonts

Если инструкции по использованию веб-шрифтов, рассмотренные в предыдущем разделе, вас смущают, есть более простой способ, хотя и с меньшим выбором шрифтов, предоставленный компанией Google. В дополнение к таким службам, как поиск, карты, электронная почта и множество других, компания Google предоставляет простой в применении сервис веб-шрифтов. Вместо загрузки шрифтов, преобразования их в нужные форматы, а затем размещения на своем веб-сервере вы добавляете одну строку кода со ссылкой на внешнюю таблицу стилей Google, в которой показано, какой шрифт вы предпочитаете использовать. Сервер Google отправляет нужные шрифты в браузер посетителя без всякой путаницы и суеты с вашей стороны.

Вам остается только найти требуемые шрифты на сайте Google Web Fonts, скопировать необходимый код (предоставляемый Google) и добавить его в код своей веб-страницы, после чего создать CSS-стили, использующие указанные шрифты. Сначала нужно посетить сайт Google Fonts, который находится по адресу google.com/fonts (рис. 6.6).

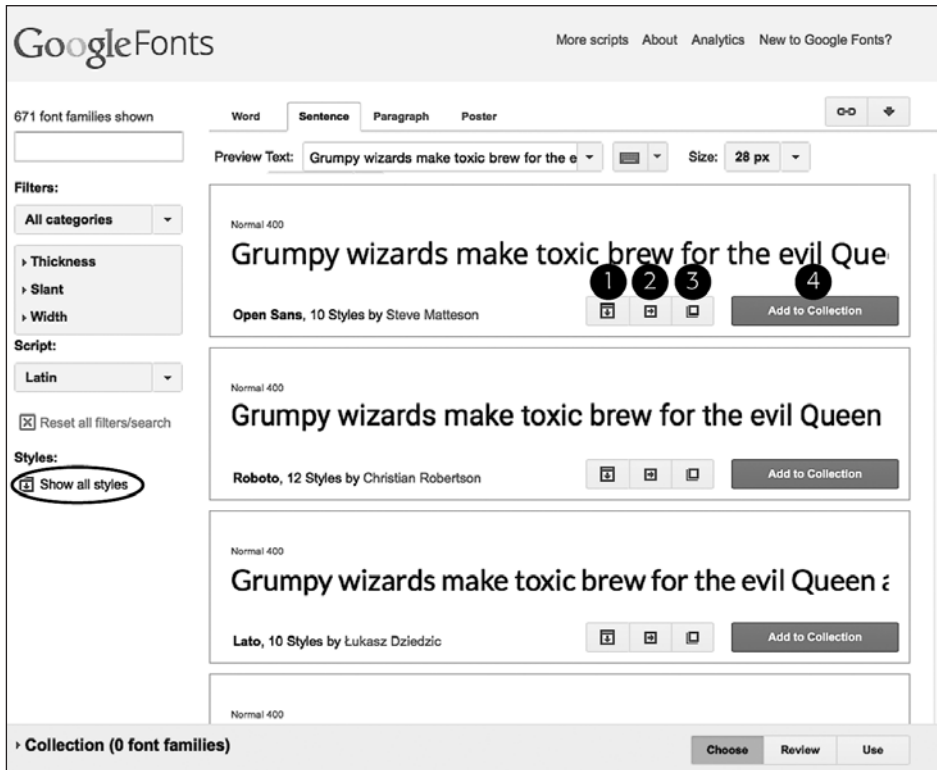


Рис. 6.6. На сайте Google Fonts перечислены шрифты, предлагаемые компанией Google

У некоторых шрифтов есть несколько стилей, например полужирный, курсив, тонкий, ультратонкий и т. д. Для просмотра всех вариантов конкретного шрифта нужно щелкнуть на ссылке **Show All Styles** (Показать все стили) на боковой панели слева (выделена на рис. 6.6). Кроме того, чтобы увидеть доступные стили конкретного шрифта, нажмите кнопку **See all styles** (Просмотреть все стили) (см. рис. 6.6, 1). Чтобы использовать шрифт, нажмите кнопку **Quick Use** (Быстрое использование) (см. рис. 6.6, 2), которая вскоре будет рассмотрена. Нажатие кнопки **Pop-out** (Показать в отдельном окне) (см. рис. 6.6, 3) приводит к открытию нового окна, в котором представляется подробная информация о шрифте, а также образец, показывающий каждую букву шрифта (хороший способ составить общее представление о внешнем виде алфавита и убедиться в наличии всех необходимых вам символов, например редко используемых или знаков пунктуации). Кнопка **Add to Collection** (Добавить к коллекции) (см. рис. 6.6, 4) позволяет добавить шрифт в коллекцию. (Если нужно использовать более одного шрифта, следует все шрифты добавить в коллекцию.)

Поиск и выбор шрифтов

Нужные шрифты выбираются путем создания *коллекции*. Вы находите понравившийся шрифт и нажимаете кнопку **Add to Collection** (Добавить к коллекции) (см. рис. 6.6).

Чтобы найти шрифт, можно прокрутить главную страницу сайта Google Fonts и просмотреть примеры доступных шрифтов, но при наличии более чем 500 шрифтов поиск нужного шрифта может занять довольно много времени. Если вы уже представляете себе, как должен выглядеть искомый шрифт, например, он должен быть полужирным рубленым шрифтом, предназначенным для заголовков, воспользуйтесь элементами управления в левой части страницы (рис. 6.7).

- **Поиск по имени.** Если известно название интересующего вас шрифта, то его нужно указать в поле поиска (см. рис. 6.7, 1). К списку шрифтов будет применен фильтр, чтобы показать те из них, которые соответствуют условию поиска.
- **Установка фильтра по категории.** В раскрывающемся списке категорий (см. рис. 6.7, 2) сбросом флажков можно отфильтровать шрифты, соответствующие одной или нескольким категориям: **Serif** (Антиквенные), **Sans Serif** (Рубленые), **Display** (Экранные), **Handwriting** (Рукописные) и **Monospace** (Моноширинные). Чтобы исключить ненужный тип шрифта, следует сбросить соответствующий флажок, а чтобы отобразить — установить. Экранные шрифты обычно бывают полужирными и декоративными, они не подходят для длинных текстовых отрывков, но могут пригодиться для коротких заголовков, акцентирующих внимание на странице. Рукописные шрифты создают иллюзию, что текст написан от руки. Они варьируются от каллиграфических до детских каракулей.
- **Выбор характеристик стиля.** Характеристики шрифтов можно определить с помощью трех ползунков (см. рис. 6.7, 3). Ползунок насыщенности (**Thickness**) позволяет подобрать шрифты, контуры которых варьируются от очень тонких (настолько тонких, что надписи трудно читаются, пока не будут отображены шрифтом большого размера) до очень толстых (полужирных и плотных). Ползунок наклона (**Slant**) фильтрует шрифты по степени наклона: обычно это означает использование курсивных версий шрифтов, но относится и к рукописным шрифтам, которые, как правило, имеют видимый наклон вправо. И наконец, использование ползункового регулятора ширины (**Width**) позволяет подобрать шрифты зауженного либо расширенного начертания. Используя более широкие шрифты, можно поместить на одну строку всего несколько символов, но зачастую это позволяет придать заголовку более внушительный вид.
- **Выбор набора символов.** И наконец, меню набора символов (**Script**) (см. рис. 6.7, 4) позволяет выбрать шрифты с поддержкой символов, отличных от латиницы. Английский язык и многие европейские языки используют набор латинских символов (**Latin**), но если нужен, например, шрифт для русского текста, можно выбрать набор кириллических символов (**Cyrillic**). Выберите тот набор символов, который соответствует вашему языку.

Если воспользоваться сразу всеми фильтрами, может не быть никаких результатов. В таком случае следует щелкнуть кнопкой мыши на ссылке **Reset all filters/search** (Сброс всех фильтров/поиска), выделенной на рис. 6.7, что позволит вернуться к полному списку веб-шрифтов Google Fonts.

ПРИМЕЧАНИЕ

Чтобы посмотреть демонстрацию некоторых лучших шрифтов Google Fonts, перейдите на сайт tinyurl.com/6lz7u7j.

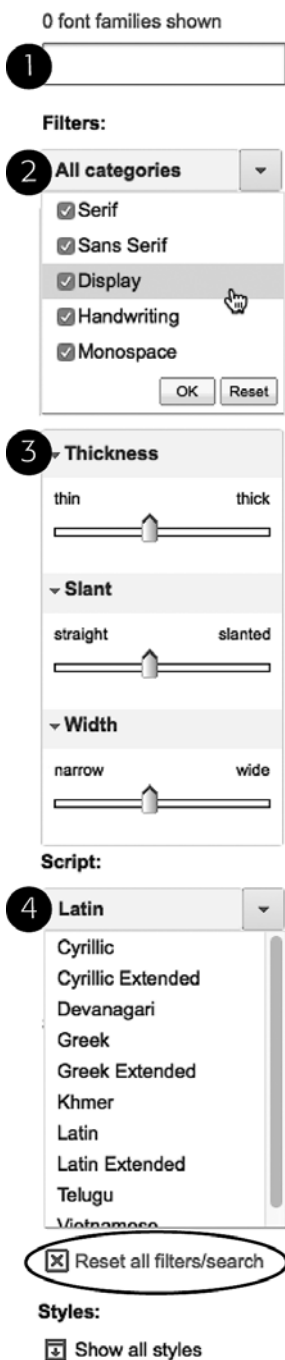


Рис. 6.7. Для содействия поиску шрифтов, соответствующих вашему дизайну, можно провести поиск по каталогу Google Fonts или отфильтровать список шрифтов по различным критериям

Как только нужные шрифты будут найдены, следует нажать кнопку **Add to Collection** (Добавить к коллекции) (рис. 6.8, 1). Коллекция представляет собой некое подобие корзины покупателя, в которой можно добавлять и удалять шрифты.

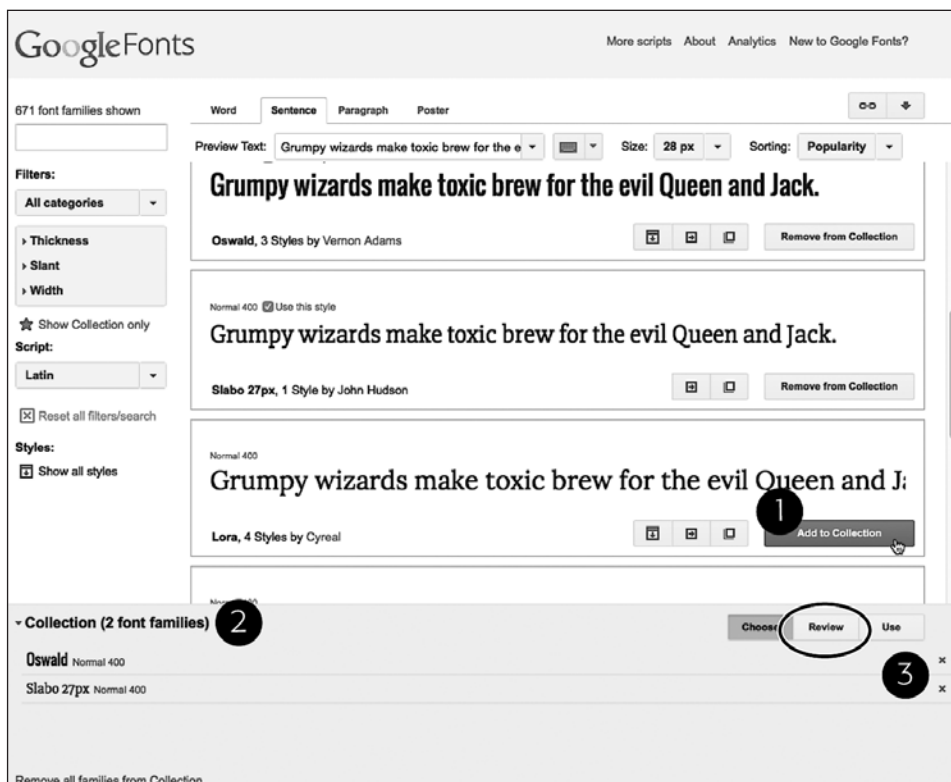


Рис. 6.8. Добавляем шрифты в коллекцию

Чтобы просмотреть шрифты, добавленные в коллекцию, щелкните на стрелке (см. рис. 6.8, 2). Шрифт можно удалить из коллекции, нажав кнопку с крестиком справа от названия шрифта (см. рис. 6.8, 3). Кнопка **Review** (Обзор), выделенная на рис. 6.8, позволяет сравнивать шрифты в коллекции и отображает подробную информацию. Например, можно увидеть полный набор символов для каждого шрифта (то есть каждую букву и символ), протестировать шрифт, печатая собственный текст и изменяя размер шрифта, и даже выполнить сравнение с наложением всех шрифтов в коллекции.

Использование шрифтов Google

После создания коллекции шрифтов настанет черед получения кода, необходимого для их использования.

1. В правом нижнем углу экрана Google Fonts (рис. 6.9, 1) нажмите кнопку **Use** (Использовать).

Откроется страница с настройками, а также с кодом, который нужно будет скопировать.

The screenshot shows the Google Fonts settings interface. At the top, it says "Verify your settings below and then copy the code for your website." Below this are four main sections:

- 1. Choose the styles you want:**
 - Under "Oswald", the "Normal 400" style is selected (marked with a '2').
 - Under "Slabo 27px", the "Normal 400" style is selected.
 - Preview text: "Grumpy wizards make toxic brew for the evil Queen and Jack." is shown in the selected styles.
 - A "Page Load" gauge shows a score of 26.
 - Tip: "Using many font styles can slow down your webpage, so only select the font styles that you actually need on your webpage."
- 2. Choose the character sets you want:**
 - "Latin (latin)" is selected.
 - Tip: "If you choose only the languages that you need, you'll help prevent slowness on your webpage."
 - Link: "Read more on how to use subsets"
- 3. Add this code to your website:**
 - Method "Standard" is selected (marked with a '3').
 - Code: `<link href='http://fonts.googleapis.com/css?family=Oswald|Slabo+27px' rel='stylesheet' t`
 - Instructions: "To embed your Collection into your web page, copy the code as the first element in the <head> of your HTML document."
 - Link: "See an example"
- 4. Integrate the fonts into your CSS:**
 - Text: "The Google Fonts API will generate the necessary browser-specific CSS to use the fonts. All you need to do is add the font name to your CSS styles. For example:"
 - Code: `font-family: 'Oswald', sans-serif;` and `font-family: 'Slabo 27px', serif;` (marked with a '4').
 - Instructions: "Add the font name to your CSS styles just as you'd do normally with any other font."
 - Example: `h1 { font-family: 'Metropolis', Arial, serif; font-weight: 400; }`

At the bottom, a summary bar shows "Collection (2 font families)" and buttons for "Choose", "Review", and "Use" (marked with a '1').

Рис. 6.9. Когда будете готовы к использованию шрифтов, добавленных в вашу коллекцию Google Fonts, нажмите кнопку Use (Использовать) (1), выберите нужный стиль (2) и метод, который хотите использовать для добавления этих шрифтов на страницу (3)

2. Выберите стиль, который нужно использовать (см. рис. 6.9, 2).

Некоторые шрифты включают курсивный, полужирный и другие начертания. Для основного текста страницы обычно нужны как минимум обычный, курсивный и полужирный варианты. Для заголовков можно обойтись и одним вариантом шрифта. Вы, наверное, заметили «обратный спидометр» в правой части страницы. По мере добавления новых стилей и шрифтов его указатель движется по часовой стрелке, показывая, что загрузка шрифтов потребует больше времени.

Это один из недостатков веб-шрифтов. Поскольку посетители вашего сайта вынуждены их загружать (наряду с веб-страницей, внешними таблицами стилей, графикой и другими элементами, составляющими страницу), нужно поступать разумно и не использовать слишком много шрифтов. В противном случае людям придется долго ждать, пока текст отобразится на экране. Цифры на спидометре показывают среднее количество миллисекунд, затрачиваемых на загрузку файлов шрифтов.

3. Дополнительно выберите тот набор символов, который вам нужен.

Это действие доступно не для всех шрифтов. Кроме того, при выборе набора символов, отличного от **Latin** (Латиница) (см. последний пункт предыдущего маркированного списка), можно увидеть и другие варианты, кроме **Latin** (Латиница) и **Latin Extended** (Латиница расширенная). Расширенный вариант пригодится в том случае, если ваш текст содержит слова на языке, использующем особые символы, например на турецком, валлийском или венгерском. Для большинства языков с символами латиницы, например французского и испанского, достаточно будет алфавита из набора **Latin** (Латиница). Если расширенный набор не нужен, то его лучше не использовать, чтобы не увеличивать размер файла и время загрузки шрифта. Для форматирования русскоязычного текста следует выбрать набор **Cyrillic** (Кириллица). Аналогично латинице набор **Cyrillic Extended** (Кириллица расширенная) содержит дополнительные редко используемые символы, например церковнославянские буквы.

ПРИМЕЧАНИЕ

Для просмотра перечня дополнительных символов, доступных в наборах **Latin Extended** (Латиница расширенная) и **Cyrillic Extended** (Кириллица расширенная), посетите сайт tinypurl.com/p577lm2 и tinypurl.com/ovvavqw соответственно.

4. Скопируйте код в поле **Add this code to your website** (Добавьте этот код на свой сайт) (см. рис. 6.9, 3).

Доступны три вкладки.

- **Standard** (Стандарт) — предоставляет элемент `link`, указывающий на внешнюю таблицу стилей (то же самое, что и создание ссылки на любую внешнюю таблицу стилей) на веб-сервере Google и предоставляющий информацию, необходимую Google для доставки нужных шрифтов. Например:

```
<link href='http://fonts.googleapis.com/css?family=Lato:300,400,300italic,400italic|Oswald:400,700' rel='stylesheet' type='text/css'>
```

Обратите внимание, что в конце атрибута `href` перечисляются шрифты и их стили. В данном примере используются шрифты **Lato** и **Oswald**. Сервер

Google загрузит несколько стилей: `300`, `400`, `300italic` и `700italic` для Lato. Эти числа являются способом обозначения веса (насыщенности) шрифта и рассматриваются далее в этом разделе. Кроме того, число и слово `italic` (например, `300italic`) означает шрифт указанной ширины с курсивным начертанием.

ПРИМЕЧАНИЕ

Если сайт Google Fonts предлагает использовать код `"type='text/css'"` как часть элемента `link`, его можно опустить. В версии HTML5 этот код не нужен.

- **@import** — использование правила `@import`. Для этого нужно щелкнуть на названной вкладке (см. рис. 6.9, 3). Преимущество такого подхода заключается в том, что правило `@import` можно добавить в начало другой таблицы стилей. Предположим, у вас есть одна таблица стилей, предназначенная для вашего сайта, на которую ссылаются все его страницы. Стандартный метод `link` требует добавления кода к каждой странице сайта. А используя правило `@import`, можно добавить код к единственной внешней таблице стилей.
- **Javascript** — подход, требующий использования кода на языке JavaScript. В этой книге данный метод не рассматривается, поскольку он требует большого объема кода, и если вы не сильны в языке JavaScript, то вполне можете допустить ошибку. Кроме того, этот вариант не дает весомых преимуществ по сравнению с двумя другими.

5. Добавьте код на веб-страницы своего сайта.

Если используется метод `link`, рассмотренный в предыдущем пункте, то код нужно добавить на каждую страницу, на которой вы собираетесь применять шрифты. Если вы только приступили к процессу создания своего сайта, это сделать нетрудно, но если ваш сайт состоит из большого количества страниц, то придется потрудиться. В таком случае присмотритесь к правилу `@import`: его можно поместить в верхней части внешней таблицы стилей вашего сайта, после чего все страницы, имеющие ссылку на эту таблицу стилей, также будут использовать нужные шрифты.

ПРИМЕЧАНИЕ

Метод `@import` может оказать небольшое воздействие на производительность ваших сайтов (то есть немного снизить скорость их работы).

6. Создайте стили, использующие шрифты.

После того как шрифты загружены, их можно применять практически так же, как и любой другой шрифт. Нужно лишь создать стиль, добавить свойство `font-family` и указать шрифт. Имя шрифта показано в нижней части страницы сайта Google Fonts (см. рис. 6.9, 4).

Если используется несколько начертаний шрифта, нужно также добавить к стилю свойства `font-weight` и `font-style`. В Google Fonts обычные ключевые слова `normal` или `bold` для обозначения насыщенности шрифтов не указываются. Вместо них задается числовая шкала в диапазоне значений от 100 до 900. Значение 700 соответствует варианту `bold`, 400 — `normal`, а остальные числа обозначают

другие варианты толщины. Предположим, вам нужно применить обычную курсивную версию шрифта Gentium Book Basic к элементу `em`. Для этого можно создать следующий стиль:

```
em {  
  font-family: "Gentium Book Basic", Palatino, serif;  
  font-weight: 400;  
  font-style: italic;  
}
```

Форматирование текста цветом

Черно-белые оттенки хорошо смотрятся в *классических* и документальных кинолентах, но когда мы имеем дело с текстом, шрифт небесно-синего цвета будет выглядеть намного более изящно и стильно по сравнению с черным. Окрашивание текста веб-страниц средствами каскадных таблиц стилей не представляет трудностей. Фактически мы уже использовали свойство `color` в предыдущих практиках. Существует несколько способов определения конкретного цвета, но все они базируются на одном принципе: нужно указать само свойство `color` и затем его значение:

```
color: #3E8988;
```

В этом примере значение цвета представлено шестнадцатеричным числом, определяющим слабый оттенок зеленовато-голубого цвета (о *шестнадцатеричных* значениях читайте далее).

В КУРС ДЕЛА!

Adobe TypeKit в качестве альтернативы Google Fonts

Из-за технических и правовых требований к использованию веб-шрифтов некоторые компании взяли на себя все связанные с этим трудности. И Google Fonts не единственный пример. Такие *службы* позволяют выбирать из больших коллекций *шрифтов*, находящихся на собственных веб-серверах компаний. Иными словами, вы не помещаете шрифты на свой сервер, а лишь ссылаетесь на серверы компаний, используя код на языке CSS или JavaScript. Эти службы берут заботу по отправке нужных шрифтов браузерам ваших посетителей на себя (например, EOT для Internet Explorer 8 и более ранних версий).

Коммерческая служба корпорации Adobe, которая называется TypeKit, также предоставляет широкий выбор шрифтов, но за плату. Эта служба является частью корпорации Adobe (которая вдобавок ко всему произ-

водимому программному обеспечению занимается также созданием шрифтов) и предоставляет доступ к широкому диапазону профессионально созданных шрифтов. С помощью TypeKit создаются отдельные *наборы* или коллекции шрифтов, которые назначаются сайту. Затем к каждой странице вашего сайта нужно добавить фрагмент JavaScript-кода. Этот код устанавливает подключение к серверам Adobe и доставляет запрошенные вами шрифты посетителям сайта. Adobe TypeKit, помимо платных подписок, предоставляет возможность и бесплатного использования шрифтов, но с ограничениями. В зависимости от количества шрифтов, к которым нужно получить доступ, и ежемесячного количества посетителей вашего сайта абонентская плата может составить от \$50 в год. Кроме того, получить доступ к службе можно, оформив подписку Creative Cloud по адресу tinyurl.com/nagp258.

Каждый графический редактор, например Fireworks, Photoshop или GIMP, позволяет указать цвет с помощью шестнадцатеричного или RGB-значения. Кроме того, встроенные в операционные системы Windows и OS X инструменты позволяют определить цвет на цветовом колесе или палитре, а затем преобразовать его в шестнадцатеричное или RGB-значение.

СОВЕТ

Если вам требуется помощь по цветовому оформлению, можете найти множество согласованных коллекций цветов, а также полезных связанных с ними ресурсов во Всемирной паутине по адресу colourlovers.com. По адресу paletton.com находится инструмент для создания веб-цвета и палитры.

Шестнадцатеричные значения цветов

Изначальной системой задания цвета, используемой веб-дизайнерами, является *шестнадцатеричная нотация*. Значение, например #6600FF, фактически содержит три шестнадцатеричных числа. В данном примере это 66, 00 и FF. Каждое число определяет уровень красного, зеленого и синего цветов соответственно (как и в цветовой модели RGB, описываемой далее). Окончательное значение цвета получается при смешивании этих трех составляющих.

СОВЕТ

Можете сокращать шестнадцатеричные числа до трех символов, если каждое из трех двухзначных чисел содержит по два одинаковых символа. Например, #6600FF можно привести к виду #60F, а #FFFFFF к #FFF.

RGB

Можно также пользоваться методом перечисления значений по цветовой модели RGB (red — «красный», green — «зеленый», blue — «синий»), с которой вы, возможно, знакомы из графических редакторов. Значение цвета кодируется тремя числами, представляющими процентные соотношения (0–100 %) или числа в диапазоне 0–255 для указания насыщенности каждого компонента (красный, зеленый, синий). Если вы хотите установить белый цвет текста (например, чтобы контрастно выделить его на темном фоне веб-страницы), можно использовать следующее свойство:

```
color: rgb(100%,100%,100%);
```

или

```
color: rgb(255,255,255);
```

СОВЕТ

Вы всегда можете вернуться к классическим именам цветов в языке HTML. Однако это будет минус в новизне и оригинальности вашего сайта. Существует 17 ключевых имен цветов: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, yellow. В CSS-коде их добавляют в стили следующим образом: color: fuchsia; и т. п. Кроме этого, большинство браузеров поддерживает 147 SVG-цветов (которые также называются цветовой схемой X11), поэтому если вам действительно хочется что-то представить в более выгодном цвете, начните использовать такие цвета, как льняной, шоколадный, хаки и белый дым. Перечень имен этих цветов можно найти по адресу tinyurl.com/kqdborf. На странице tinyurl.com/m4j9e5 имена цветов объединены по оттенкам.

RGBA

Чтобы придать странице выразительность, стоит рассмотреть более современные методы задания цвета. Аббревиатура RGBA означает Red — красный, Green — зеленый, Blue — синий, Alpha — *альфа-канал*, определяющий уровень смешивания цвета с фоном. Этот способ задания цвета работает так же, как и RGB, но с добавлением альфа-канала. То есть можно задать уровень прозрачности, превратив цвет из сплошного в полупрозрачный (рис. 6.10). К значениям цвета по модели RGB добавляется еще одно число с диапазоном от 0 до 1. Значение 0 обозначает полностью прозрачный цвет, а 1 — на 100 % непрозрачный (то есть сквозь него ничего невозможно увидеть):

```
color: rgba(255, 100, 50, .5);
```

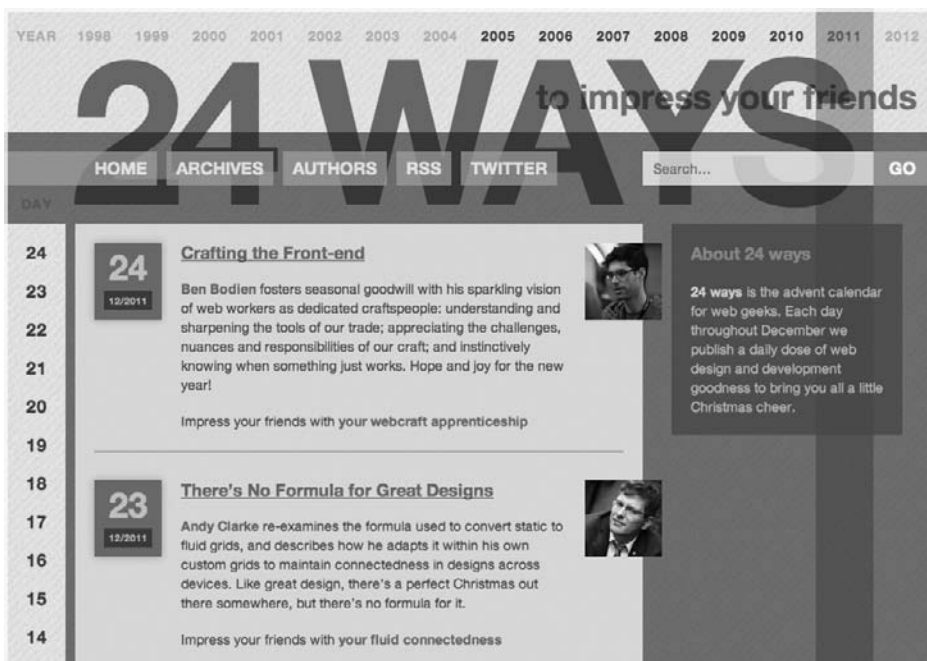


Рис. 6.10. Запись значений по модели RGBA применяется не только для текста. Ею можно пользоваться в сочетании с любым свойством каскадных таблиц стилей, использующим значение цвета, например с цветом фона поля поиска в правом верхнем углу этого изображения или с цветом кнопок навигации: Home, Archives, Authors и т. д.

Накладывая текст, цвет шрифта которого задан с помощью модели RGBA, на фоновые изображения, можно создать интересные визуальные эффекты. Например, можно создавать изображения, частично видимые сквозь символы текста (путем использования такого большого значения, как .9) или видимые более явно (значение .1).

СОВЕТ

Модель RGBA особенно хорошо поддерживается свойствами `text-shadow` и `box-shadow`, которые будут рассмотрены далее. Используя запись значений по модели RGBA, можно создавать более тонкие эффекты отбрасываемых теней, позволяя основной части фона просматриваться сквозь тень.

А в чем же недостатки такого метода? Internet Explorer 8 и его более ранние версии не поддерживают значения по модели RGBA. Один из выходов заключается в задании сначала сплошного цвета с использованием шестнадцатеричной нотации для старых браузеров, а затем второго свойства цвета с помощью модели RGBA:

```
color: rgb(255,100,50); /* для IE8 */  
color: rgba(255,100,50,.5); /* для современных браузеров */
```

Первую строку интерпретируют все браузеры, а вторая строка отменяет первую, но только для тех браузеров, которые поддерживают значения по модели RGBA. Иными словами, IE 8 использует первое объявление цвета и игнорирует второе, а IE 9 и другие браузеры используют модель RGBA. Вы просто не увидите в программе IE 8 эффекта прозрачности.

HSL и HSLA

Аббревиатура HSL расшифровывается как Hue — «тон», Saturation — «насыщенность» и Lightness — «светлота» (иногда также используется термин luminance — «яркость»). Это еще один способ задания цвета. Он не поддерживается Internet Explorer 8 и более ранними версиями, но работает во всех остальных браузерах. Если вы используете запись значений по модели RGB или шестнадцатеричные значения, то синтаксис HSL может показаться немного необычным. Вот так выглядит определение ярко-красного цвета:

```
color: hsl(0, 100%, 50%);
```

Внутри конструкции `hsl()` задаются три значения. Первое — значение градуса от 0 до 360 цветового круга. Если вы помните очередность следования цветов в радуге — красный, оранжевый, желтый, зеленый, голубой, синий и фиолетовый, то поймете основную идею значения, через которое выражен цвет. Красный — это 0 (это также и 360, поскольку это один полный оборот по кругу), желтый — это приблизительно 50, оранжевый — приблизительно 100, зеленый — 150 и т. д. Каждый цвет занимает примерно 51°.

Вторым значением указывается насыщенность, или то, насколько чистым является оттенок цвета. Насыщенность указывается в диапазоне значений от 0% до 100%. Значение 0% означает полное отсутствие насыщенности, или тусклый серый оттенок. Фактически неважно, какой тон задан, 0% всегда даст один и тот же серый тон. Значение 100% задает чистый цвет, яркий и живой. Третье значение определяет степень светлоты, указанную в процентах от 0% (полностью черный) до 100% (полностью белый). Если нужно получить чистый цвет, следует воспользоваться значением 50%.

Предполагалось, что метод HSL будет интуитивно понятнее, чем RGB или шестнадцатеричные значения, но если вы считаете, что его трудно освоить, можете им не пользоваться. Вместо этого обратитесь к Fireworks или Photoshop либо воспользуйтесь палитрой цветов во Всемирной паутине, что существенно упростит выбор значения по модели RGB или шестнадцатеричного значения.

ПРИМЕЧАНИЕ

Если вас заинтересовала модель HSL, то по адресу hslpicker.com можно найти простую в применении палитру цветов.

Точно так же, как у RGB есть сопутствующий формат RGBA с альфа-каналом, HSL поддерживает прозрачность с помощью формата HSLA. Он работает аналогично RGBA. Значение прозрачности указывается в диапазоне от 0 (невидимый) до 1 (полностью непрозрачный). Следующее свойство приведет к созданию яркочерного цвета с 50%-ной прозрачностью:

```
color: hsla(0, 100%, 50%, .5);
```

Примеры в книге больше ориентированы на применение моделей RGB и RGBA, но если HSL вам более понятен, используйте этот формат!

Изменение размера шрифта

Установка определенного размера шрифта текста веб-страницы — хороший способ визуально придать тексту значимость и привлечь внимание посетителей к наиболее важным фрагментам страницы. Заголовки, отображенные шрифтом большего размера, привлекают внимание. В то же время информация, отображенная маленьким, возможно, подстрочным шрифтом, позволяет этому блоку не выделяться на фоне общего текста веб-страницы, создавая ненавязчивый комментарий.

Свойство `font-size` устанавливает размер шрифта текста. За значением всегда должна следовать единица измерения величины. Например, так:

```
font-size: 1em;
```

Сочетание значения свойства и единицы измерения, которые вы указываете для установки размера шрифта (в данном примере `1em`), определяют размер шрифта текста. Каскадные таблицы стилей предлагают большой выбор единиц измерения: предопределенные ключевые слова, `em` (стандартная единица измерения в типографской системе: размер буквы М, напечатанной шрифтом Cicero), `ex` (то же самое, только берется размер буквы X), пиксели, проценты, пики, пункты, дюймы, сантиметры и миллиметры.

Единицы измерения, обычно используемые в печати (пики, пункты, дюймы и т. д.), не очень удобно применять к веб-страницам, так как невозможно предугадать их вид на разных мониторах. Однако пункты удобно использовать при создании таблиц стилей для веб-страниц, ориентированных для печати на принтере. Только небольшую часть всех существующих единиц измерения — пиксели, ключевые слова, `em`, проценты — можно применять для определения размеров шрифтов текста веб-страниц, отображаемых браузером на экране монитора. В следующей части книги рассказывается, как они работают.

Пиксели

Значения в пикселях наиболее просты для понимания, поскольку не зависят от настроек браузера. Когда вы определяете, например, размер шрифта равным 36 пикселей для заголовка `h1`, браузер отображает текст высотой 36 пикселей. Дизайнеры выбирают эти единицы измерения потому, что они обеспечивают

постоянные согласованные параметры размеров текста на различных типах компьютеров и браузеров.

Чтобы присвоить свойству `font-size` значение в пикселах, введите число с символами `px`:

```
font-size: 36px;
```

ПРИМЕЧАНИЕ

Не добавляйте пробел между значением свойства и единицей измерения. Например, правильно `36px`, но не `36 px`.

В КУРС ДЕЛА!

Пиксели и Retina-дисплеи

Когда компания Apple представила первые плееры iPod и смартфоны iPhone с Retina-дисплеем (начиная с модели iPhone 4 и iPod 4-го поколения), владельцы этих устройств радовались яркости и четкости изображений. Retina-дисплеи компании Apple предоставляют более четкое изображение за счет использования большего количества пикселей на квадратный дюйм. В то время как в обычных компьютерных дисплеях этот показатель варьируется в пределах 72–100 пикселей на дюйм, Retina-дисплеи могут похвастаться значением вплоть до 400 пикселей на дюйм.

С тех пор Apple использует Retina-дисплеи (за исключением плееров и смартфонов) в таких устройствах, как планшеты iPad и компьютеры iMac/MacBook. Другие изготовители планшетов и компьютеров пытаются не отставать, предлагая устройства с дисплеями с существенно большим количеством пикселей на дюйм,

чем ранее. Что все это означает для веб-дизайнеров? Существенное увеличение объема работы. Далее мы еще вернемся к этой теме. Из врезки «Когда пиксел уже не пиксел?» главы 15 вы узнаете, что Retina-дисплеи существенно влияют на изображения и вам придется потрудиться, чтобы изображения на таких экранах выглядели достойно.

Что же касается пикселей, упомянутых выше, то браузеры устройств, оснащенных Retina-дисплеями, фактически умножают значение в пикселах на 2. Иначе говоря, если указать для текста размер 16 пикселей, то браузеры, подстраиваемые под Retina-дисплей, фактически будут использовать на экране для прорисовки текста 32 пиксела. Это хорошая новость. Если бы браузер для отображения текста использовал только 16 из своих маленьких пикселей, то никто не смог бы прочитать текст на Retina-дисплее.

Ключевые слова, проценты и единица измерения `em`

Все перечисленные далее способы изменения размера шрифта средствами CSS с помощью ключевых слов, процентных значений и единиц `em` влияют на текст, отображаемый в окне браузера, в соответствии с некоторыми правилами. Другими словами, если вы не определите размер средствами каскадных таблиц стилей, то браузер применит к тексту свои предопределенные параметры. В большинстве случаев обычный текст, находящийся вне элементов заголовков, отобразится высотой 16 пикселей — это *базовый размер шрифта* текста.

Пользователи могут корректировать настройки браузеров, увеличивая или уменьшая базовый размер шрифта, но для этого нужно ковыряться в исходных настройках браузера, с чем большинство людей не захочет связываться.

ПРИМЕЧАНИЕ

У большинства браузеров имеется функция масштабирования, позволяющая увеличить/уменьшить весь контент на странице. Выбор масштаба на самом деле не изменяет базовый размер шрифта текста. Нажатие сочетания клавиш Ctrl++ (⌘++) на большинстве браузеров приводит к увеличению, а сочетания Ctrl+- (⌘+-) — к уменьшению масштаба. Удерживание нажатой клавиши Ctrl (^) и использование колесика мыши также позволяет увеличивать и уменьшать масштаб страницы.

Когда вы изменяете текст с помощью каскадных таблиц стилей, браузер берет за основу базовый размер шрифта текста (будь то первоначальный высотой 16 пикселей или другой) и корректирует его в большую или меньшую сторону в соответствии со значением ключевого слова, единицей измерения em или процентным отношением.

Ключевые слова

Каскадные таблицы стилей предлагают семь ключевых слов, которые позволяют назначить размер шрифта относительно базового: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` и `xx-large`. CSS-код будет выглядеть следующим образом:

```
font-size: large;
```

Среднее значение `medium` — базовый размер шрифта браузера. Остальные значения уменьшают или увеличивают размер шрифта с различными коэффициентами. Другими словами, несмотря на то, что, казалось бы, каждое изменение размера должно быть последовательным увеличением или уменьшением предыдущего значения, это не так. Обычно `xx-small` является эквивалентом 9 пикселей (принимая во внимание, что вы не изменяли базовый размер шрифта в своем браузере), `x-small` соответствует 10 пикселям, `small` — 13, `large` — 18, `x-large` — 24, а `xx-large` — 32 пикселям.

Как вы заметили, имеется ограниченный набор ключевых слов — всего семь вариантов. Если требуется большая свобода действий по масштабированию, надо обратиться к другим средствам и единицам измерения, описываемым далее.

Процентные значения

Подобно ключевым словам, процентные значения изменяют размер текста относительно базового размера шрифта, определенного браузером. Они обеспечивают более гибкое и точное управление, чем ключевые слова `large`, `x-large` и т. д. Любой браузер имеет предопределенный базовый размер шрифта. У большинства он составляет 16 пикселей. Можно изменить это значение в настройках браузера. Независимо от настройки, основной размер шрифта эквивалентен 100 %. Другими словами, по умолчанию он равнозначен установке шрифта высотой 16 пикселей.

Допустим, вы хотите отобразить определенный заголовок в два раза крупнее, чем средний шрифт веб-страницы. Для этого установите размер шрифта, равный 200 %:

```
font-size: 200%;
```

Или, если хотите, чтобы шрифт был немного меньше базового, укажите значение 90 %.

Приведенные примеры являются самыми простыми, но на практике встречаются более сложные ситуации. Например, относительный размер 100 % может изменяться, если наследуется значение элемента-предка.

В левом нижнем углу веб-страницы, показанной на рис. 6.11, есть текст в элементе `div`, шрифт которого установлен размером 200 %. Это в два раза больше базового размера и составляет 32 пиксела. Все вложенные элементы наследуют его и используют для вычисления собственных размеров шрифтов. Другими словами, для элементов, вложенных в `div`, размер 100 % равен 32 пикселям. Так, текст заголовка `h1`, находящегося внутри элемента `div`, составляет 100 % и отображается в два раза больше базового для этой веб-страницы, то есть высотой 32 пиксела.

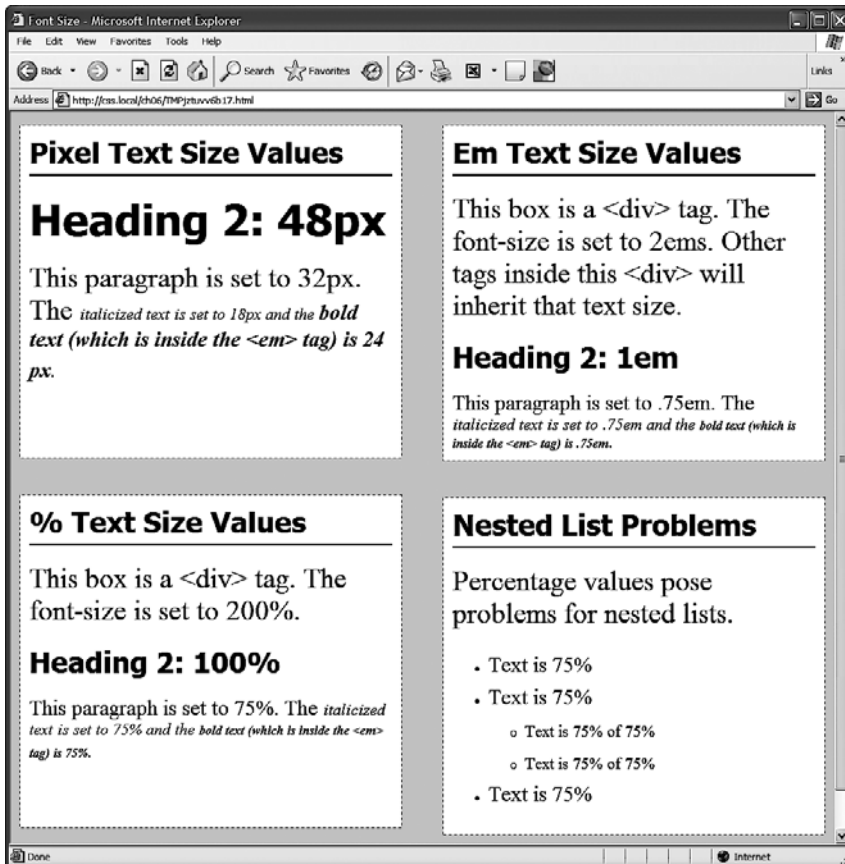


Рис. 6.11. Пиксели, em и проценты — три самые распространенные и наиболее применяемые единицы измерения для установки размеров шрифтов. Будьте внимательны при использовании единиц em и процентных значений, когда есть наследуемые свойства размеров шрифтов. Если вы заметили, что какой-либо текст выглядит не так, как предусмотрено, а необычно крупным или мелким, убедитесь в том, что он не наследует параметры. Или попробуйте использовать значение `rem`

Единица измерения em

Если вы поняли, как вычисляются процентные значения, то легко поймете и единицу `em`. Принцип их применения практически одинаков, но большинство веб-дизайнеров все же используют `em`, так как она применяется в книгопечатании.

Слово *em* заимствовано из типографского дела. Оно имеет отношение к размеру заглавной буквы «М» определенного шрифта. В мире веб-дизайна это слово приобрело собственное значение. В каскадных таблицах стилей понятие относится к базовому размеру шрифта текста. Иными словами, значение размера шрифта `1em` означает то же самое, что и `100 %`, как описано в предыдущем разделе. Иначе говоря, процентное значение эквивалентно *em*, умноженному на `100`: `.5em` — `50 %` и т. д.

Например, этот CSS-код задает то же форматирование, что и `font-size: 200%;`
`font-size: 2em;`

ПРИМЕЧАНИЕ

Как и при использовании пикселей, вы не должны указывать пробел между числом и единицей измерения *em*.

Принцип работы механизма наследования с *em* точно такой же, как и с процентами. Взгляните, например, на верхний правый угол рис. 6.11. Шрифт нижнего абзаца установлен равным `.75em`, а поскольку элемент *p* наследует размер шрифта `2em` (32 пиксела) от контейнера *div*, в результате получается размер шрифта: $0,75 \times 32 = 24$ пиксела. Внутри абзаца *p* есть два других элемента, свойству `font-size` которых присвоено значение `.75em`. Размер шрифта элемента наиболее глубокой вложенности *strong* установлен равным `.75em`, или, по сути, `75 %` от *унаследованного*. Довольно сложный расчет: `32` пиксела (размер, унаследованный от *div*) \times `75` (размер от *p*) \times `0,75` (размер *em*) \times `75` (собственный размер *strong*). В результате вычислений получается приблизительно 14 пикселей.

Унаследованные значения свойства `font-size` могут вызвать проблемы для вложенных (многоуровневых) списков (см. рис. 6.11). Например, у вас имеется стиль `ul { font-size: 75% }`, а затем вы создаете вложенный список, то есть внутри *ul* расположены другие элементы. Получается, что для вложенного *ul* должен быть установлен размер шрифта, равный `75 %` от внешнего *ul*. Следовательно, подпункты списков будут отображены меньшим шрифтом, чем пункты, и так далее применительно к подпунктам следующего подуровня.

Чтобы обойти эту проблему, создайте дополнительный стиль с селектором потомков (см. раздел «Форматирование вложенных элементов» главы 3): `ul ul {font-size: 100%}`. Этот стиль устанавливает размер шрифта любого элемента *ul*, вложенного в другой *ul*, равным `100 %`. Другими словами, `100 %` от размера шрифта родительского элемента *ul*, в который вложен другой элемент. В данном примере это обеспечивает сохранение размера шрифта вложенных подпунктов списков равным `75 %` от базового размера. Это еще один способ предотвратить эффект изменения шрифта вследствие наследования.

Единица измерения *rem*

В спецификацию CSS3 включена новая единица измерения под названием *rem*. Это название означает *Root EM*, то есть его значение основано на размере текста корневого (*root*) элемента. В большинстве случаев это относится к базовому размеру шрифта текста, поэтому значение `.75em`, показанное на рис. 6.11, можно заметить на такое:

```
font-size: .75rem;
```


Этот стиль задает размер шрифта равным 0,75 единиц от базового, а не текущего размера шрифта (как в случае применения единиц измерения `em`). Корневым элементом в языке HTML считается `html`, который можно найти в начале веб-страницы. При использовании значений `rem` можно установить базовый размер шрифта текста элемента `html`, а затем использовать единицу измерения `rem` для форматирования текста относительно этого базового размера. Например, можно установить базовый размер шрифта текста равным 20 пикселям:

```
html {  
  font-size: 20px;  
}
```

А затем воспользоваться единицей измерения `rem` для создания стиля относительно этого 20-пиксельного базового размера шрифта текста. Тогда, чтобы задать для всех абзацев размер шрифта текста, равный 15 пикселям, нужно добавить такой стиль:

```
p {  
  font-size: .75rem;  
}
```

Единица измерения `rem` позволяет избежать проблем наследования размера шрифта, с которыми сталкиваются дизайнеры при использовании процентных значений и единиц `em`. Процентные значения и единицы `em` основаны на размере шрифта их родительского элемента. Размер шрифта, заданный в процентном значении и примененный к нескольким вложенным элементам, приведет к усложнению, изменяя каждый последующий размер шрифта на определенный процент, по отношению к предыдущему. Тем не менее единицы `rem` всегда основаны на размере шрифта элемента `html`. Другими словами, они всегда сохраняют один и тот же размер шрифта, даже если вложены в элементы, которые наследуют различное форматирование.

Но нужно иметь в виду, что такую единицу, как `rem`, поддерживают на данный момент все основные браузеры, кроме Internet Explorer 8 и его более ранних версий.

СОВЕТ

Текст веб-страницы выделяют различными способами. Этого можно добиться, изменив размер шрифта определенного фрагмента текста, отдельных слов или с помощью контрастности. Окрашивание текста в темный, светлый или более яркий оттенок визуально выделяет его. Применение контрастности — одно из наиболее важных правил хорошего графического дизайна. Контрастность может помочь выделить важные сообщения, фрагменты текста, привлечь внимание. Краткий обзор особенностей управления контрастностью в типографике описан во Всемирной паутине по адресу tinyurl.com/ng39zb9.

Форматирование символов и слов

Вам потребуется немало времени для точной и тонкой настройки параметров текста: цвета, размеров шрифтов и т. д. Однако каскадные таблицы стилей предоставляют также множество других средств для форматирования текста. Из самых

распространенных свойств можно выделить, например, полужирное и курсивное начертания, а из менее широко используемых — создание заголовка, изменение межбуквенного интервала и т. д.

СОВЕТ

Средства языка CSS позволяют комбинировать множество свойств форматирования текста. Излишне «тяжеловесное» оформление усложняет страницу для чтения и понимания. Хуже всего, если из-за такого форматирования остается негативное впечатление от посещения сайта.

Курсивное и полужирное начертания

Браузеры отображают текст, заключенный внутри элементов `em` и `i`, *курсивом* (шрифтом с наклонным начертанием), а текст, находящийся в элементах `strong`, `b`, `th` (table header — «заголовок таблицы»), заголовки (`h1`, `h2` и т. д.) — **полужирным**. Вы можете управлять этим форматированием. Можно отменить полужирное начертание для заголовков или выделить курсивом обычный текст, используя свойства `font-style` и `font-weight`.

Чтобы выделить текст курсивом, добавьте к стилю следующий код:

```
font-style: italic;
```

Или, наоборот, можете установить для текста обычный, *не* курсивный шрифт:

```
font-style: normal;
```

ПРИМЕЧАНИЕ

Свойство `font-style` также содержит третью команду `oblique`, которая имеет тот же эффект, что `italic`.

Свойство `font-weight`, определяющее вес (насыщенность) шрифта, позволяет форматировать текст полужирным или обычным начертанием. Фактически, согласно правилам каскадных таблиц стилей, можно указать любое из девяти числовых значений (в диапазоне от 100 до 900) для определения точных, едва различимых градаций насыщенности (от «супернасыщенного» (900) до «крайне легкого, почти невидимого» веса 100) шрифта. Естественно, чтобы эти команды работали, сами используемые шрифты должны иметь девять различных значений насыщенности, обеспечивая тем самым визуальный эффект. Единственный способ использования числовых значений касается рассмотренных ранее веб-шрифтов. Фактически числовые значения применяются службой Google Fonts исключительно для задания насыщенности шрифта.

ПРИМЕЧАНИЕ

При использовании веб-шрифтов форматирование текста полужирным и курсивным начертанием требует выполнения ряда других, рассмотренных ранее в этой главе действий.

Чтобы сделать шрифт текста полужирным, используйте такой код:

```
font-weight: bold;
```

Обычный шрифт устанавливается следующим образом:

```
font-weight: normal;
```

СОВЕТ

Поскольку для заголовков уже предопределен стиль с полужирным начертанием, возможно, потребуется искать другой способ для выделения, которое вы хотите придать некоторым словам заголовка. Вот один из способов:

```
h1, strong {color: #3399FF; }
```

Этот стиль, с селектором потомков, меняет цвет всех элементов `strong` (обычно отображаемых полужирным шрифтом), заключенных внутри заголовка `h1`.

Прописные буквы

Набрать текст прописными буквами очень просто: нажмите клавишу `Caps Lock` и вводите текст. Что же делать, если нужно, чтобы прописными буквами были выделены все уже набранные заголовки веб-страницы или текст со строчными буквами, который скопировали и вставили из редактора Microsoft Word? Вместо того чтобы повторно набирать текст заголовков, обратитесь к свойству `text-transform`. С его помощью можно преобразовать любой текст в верхний или нижний регистры или даже превратить первые буквы каждого слова в прописные (для названий и заголовков). Ниже представлен пример преобразования в прописные буквы:

```
text-transform: uppercase;
```

Сделать буквы строчными можно с помощью значения `lowercase`, а превратить первые буквы в прописные — с помощью слова `capitalize`.

Поскольку это свойство наследуемо, любые элементы, вложенные в теги со свойством `text-transform`, приобретают то же форматирование: верхний регистр, нижний, прописные буквы. Чтобы *запретить* изменять регистр текста, используйте значение свойства `none`:

```
text-transform: none;
```

Капитель. Для придания тексту типографической изысканности можно использовать свойство `font-variant`, которое позволяет преобразовать текст таким образом, что все буквы станут капителью (малыми прописными). В этом стиле строчные буквы выглядят как немного уменьшенные прописные. Большие фрагменты такого текста становятся трудночитаемыми, но стиль придает веб-странице старосветскую, книжную многозначительность. Для создания стиля с малыми прописными буквами используйте следующий код:

```
font-variant: small-caps;
```

Декорирование текста

Каскадные таблицы стилей позволяют использовать также свойство `text-decoration` для улучшения внешнего вида текста путем добавления различных дополнительных элементов. С его помощью можно добавить линии подчеркивания, надчеркивания, зачеркнуть текст (рис. 6.12) или сделать его мигающим. Свойство `text-decoration` применяется в сочетании со следующими ключевыми словами: `underline`, `overline`, `line-through` или `blink`. Например, чтобы подчеркнуть фрагмент, наберите код:

```
text-decoration: underline;
```

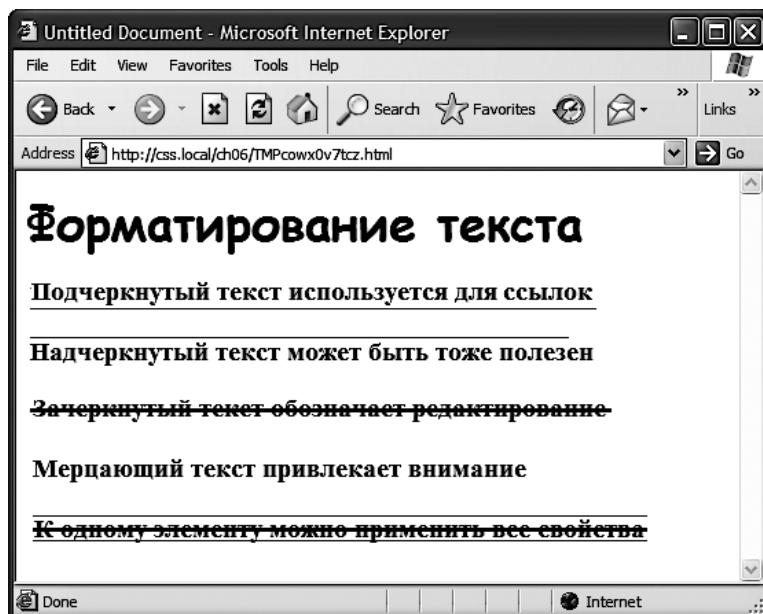


Рис. 6.12. Результат применения свойства text-decoration

Вы можете также объединять несколько ключевых слов вместе для комбинирования эффектов. Добавить к тексту линии подчеркивания и надчеркивания одновременно можно следующим способом:

```
text-decoration: underline overline;
```

Однако не стоит изощряться в таком «вычурном» форматировании только потому, что *есть такая возможность*. Во-первых, у всех, кто посещает Всемирную паутину на протяжении долгого времени, любой подчеркнутый текст инстинктивно ассоциируется с гиперссылкой, возникает желание непременно щелкнуть на ней. Таким образом, подчеркивать слова, не являющиеся частью гиперссылки, будет не очень хорошей идеей. Установив значение мерцания `blink`, вы на самом деле сделаете текст больше похожим на неоновую вывеску типа «Для любителей!» (к тому же большинство браузеров не сделают текст мерцающим, даже если вы укажете это свойство).

ПРИМЕЧАНИЕ

Вы можете применить эффект, подобный подчеркиванию и надчеркиванию, если добавите к элементу — в данном случае тексту — линию границы `border` сверху или снизу (см. раздел «Добавление границ» главы 7). Преимущество состоит в том, что с помощью свойства `border` вы можете управлять одновременно многими параметрами: размещением, размером, цветом линий-рамок и в конечном счете придать тексту более привлекательный вид, не делая его похожим на гиперссылки.

Значение `overline` свойства `text-decoration` помещает линию над текстом, а `line-through` зачеркивает текст. Некоторые веб-дизайнеры применяют последний эффект, чтобы показать читателю, что фрагмент был удален из первоначального варианта документа.

Отменить декорирование полностью можно с помощью ключевого слова `none`:

```
text-decoration: none;
```

Зачем вам может понадобиться отмена декорирования? Самый распространенный пример — удаление стандартной подчеркивающей линии у гиперссылок (см. раздел «Форматирование ссылок» главы 9).

Интервал между символами и словами

Другой способ выделения текстового фрагмента состоит в регулировании интервала, с которым отображаются отдельные буквы или слова (рис. 6.13). Уменьшение межсимвольного интервала (трекинга) с помощью свойства `letter-spacing` поможет сжать текст заголовков. Они будут казаться еще более плотными и визуально «тяжелыми», а заодно вмещать больше текста на единственной строке заголовка. И наоборот, увеличение интервала может придать заголовкам более спокойный, величественный вид. Чтобы это сделать, используйте отрицательные значения свойства:

```
letter-spacing: -1px;
```

Положительные значения свойства делают промежуток между буквами больше:

```
letter-spacing: .7em;
```

Аналогично можно изменить интервал между словами, используя свойство `word-spacing`. Оно увеличивает/уменьшает промежуток между словами, не затрагивая интервалы между буквами внутри слов:

```
word-spacing: 2px;
```

С этими свойствами можно использовать любые единицы измерений, применимые к тексту: пиксели, `em`, проценты (с положительными или отрицательными значениями).

Если вы хотите, чтобы текст был удобен для чтения, применяйте небольшие значения интервалов. В противном случае буквы и слова будут перекрываться, накладываться друг на друга. Чтобы обеспечить комфортное представление текстового контента сайта, аккуратно настраивайте интервалы между символами и словами.

Добавление тени

В спецификации CSS3 включено одно свойство, позволяющее добавлять к тексту тени для придания глубины и выразительности заголовкам, спискам и абзацам (рис. 6.14).

Свойство `text-shadow` требует задания трех параметров: горизонтального смещения (насколько левее или правее текста должна отображаться тень), вертикальное смещение (насколько выше или ниже текста должна появиться тень), степень размытости тени и цвет отбрасываемой тени. Например, эффект, показанный в верхней части рис. 6.14, задается следующим кодом:

```
text-shadow: -4px 4px 3px #999999;
```

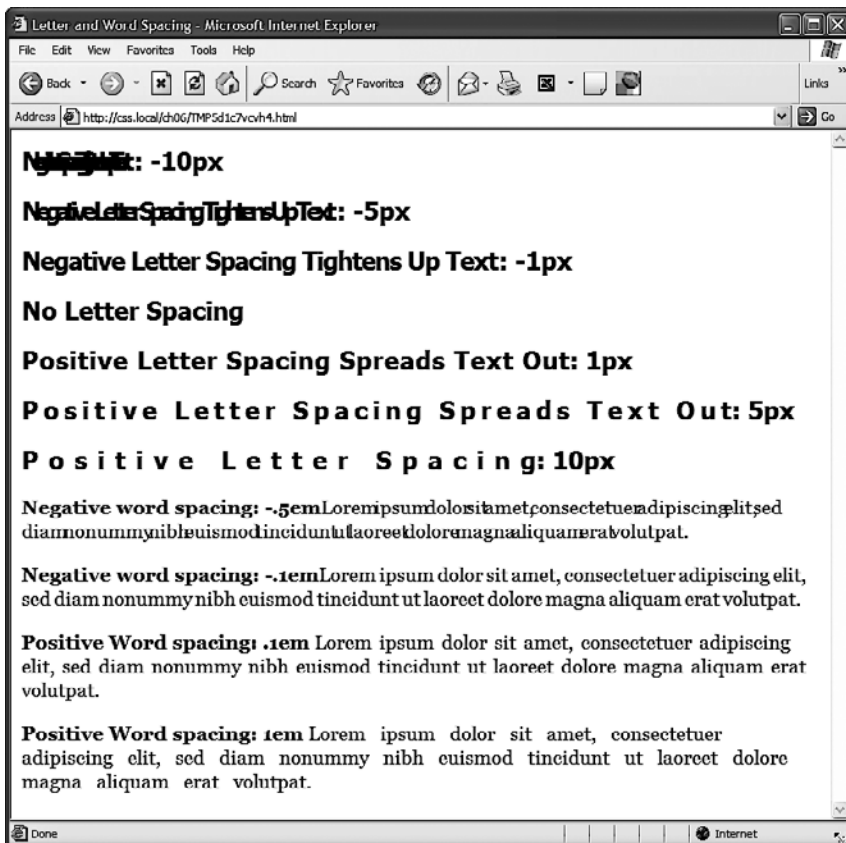


Рис. 6.13. Слишком большое или слишком маленькое значение интервала между символами либо словами может ухудшить читабельность текста

Первое значение, -4px , говорит «отобразить тень на 4 пиксела левее текста» (положительное значение приведет к отображению тени правее текста). Второе значение, 4px , задает отображение тени на 4 пиксела ниже текста (отрицательное значение приведет к отображению тени над текстом). Значение 3px определяет степень размытости тени (значение 0px (без размытости) приводит к отбрасыванию четкой тени, и чем больше будет значение, тем более размытой будет тень). И наконец, последнее значение определяет цвет отбрасываемой тени.

Для создания более сложных эффектов можно даже добавить несколько отбрасываемых теней (см. рис. 6.14, *внизу*): нужно добавить запятую, а после нее — дополнительные значения отбрасываемой тени:

```
text-shadow: -4px 4px 3px #666, 1px -1px 2px #000;
```

Количество добавляемых таким образом теней ничем не ограничивается (кроме вашего вкуса). Этот эффект не поддерживается в Internet Explorer 9 и более ранних версиях браузера. Но он работает во всех других современных браузерах (даже в более поздние версии Internet Explorer). Иными словами, чтобы текст лучше

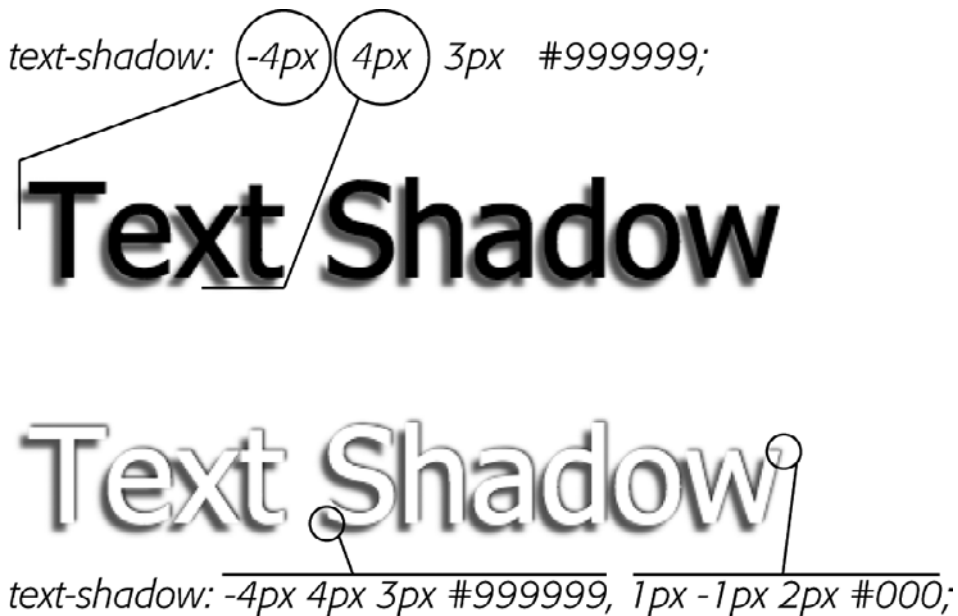


Рис. 6.14. Текст с тенью — отличный способ придания едва заметной (но если вы на этом не настаиваете, то и весьма заметной) глубины заголовкам и прочему контенту. Но свойство `text-shadow` не поддерживается в Internet Explorer 9 и более ранних версиях браузера

читался, полагаться на этот эффект *не* стоит. Изображение в нижней части рис. 6.14 показывает, что не нужно делать: использовать белый цвет текста, обеспечивая его читаемость исключительно за счет отбрасываемой тени, определяющей очертания текста. В Internet Explorer 9 и более ранних версиях браузера белый текст на белом фоне виден не будет.

ПРИМЕЧАНИЕ

Чтобы увидеть красивые приемы использования текстовых теней, посетите сайт по адресу tinyurl.com/kh5dj2s. Прекрасный пример создания трехмерного текста с помощью нескольких текстовых теней находится по адресу tinyurl.com/kh5dj2s.

Форматирование абзацев

В CSS есть свойства, которые используются для форматирования не отдельно взятых слов, а фрагментов, блоков текста. Иначе говоря, их можно применять к целым абзацам, заголовкам и т. д.

Установка межстрочного интервала

В дополнение к настройке интервала между словами и символами каскадные таблицы стилей позволяют устанавливать межстрочный интервал (*интерлиньяж*) — промежуток между базовыми линиями двух соседних строк текста, используя свойство

line-height. Чем больше межстрочный интервал, тем больше промежуток между отдельными строками (рис. 6.15).

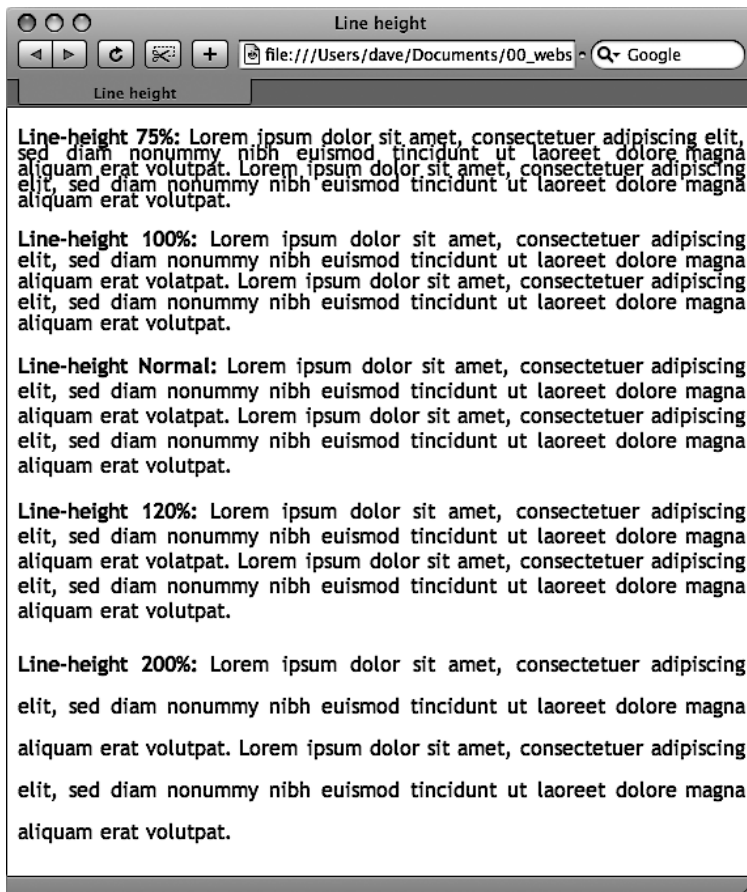


Рис. 6.15. Свойство установки межстрочного интервала, line-height, позволяет растянуть строки абзаца или расположить их ближе друг к другу, с меньшим промежутком. Стандартный межстрочный интервал эквивалентен 120 %. Таким образом, *вверху* мы видим, что меньшее процентное значение сжимает строки, а большее значение — раздвигает (*внизу*)

Интерлиньяж в пикселах, единицах em и процентах

Как и в свойстве размера шрифта, font-size, вы можете использовать пиксели, единицы em или проценты для установки размера межстрочного интервала с помощью свойства line-height:

```
line-height: 150%;
```

Вообще, процентные значения или единицы em лучше, нежели пиксели: размер, установленный в этих единицах измерения, напрямую зависит от параметров шрифта и автоматически корректируется пропорционально изменению значения свойства font-size. Так, если вы установите межстрочный интервал равным 10 пикселей

и затем измените на гораздо больший (например, 36 пикселей), то высота останется равной 10, а строки будут накладываться друг на друга. Однако при использовании значения размера в процентах (скажем, 150 %) межстрочный интервал корректируется пропорционально всякий раз, когда вы изменяете размер шрифта в соответствующем свойстве `font-size`.

Стандартный размер межстрочного интервала браузера составляет 120 %. Таким образом, когда вы хотите уменьшить высоту строк, промежутков между ними, используйте меньшее значение. Соответственно, чтобы увеличить межстрочный интервал, то есть распределить строки дальше друг от друга, используйте значение больше 120 %.

ПРИМЕЧАНИЕ

Чтобы определить размер межстрочного интервала, браузер вычитает высоту шрифта из высоты строки. В результате получается значение интерлиньяжа между двумя соседними строками текста абзаца. Допустим, размер шрифта составляет 12 пикселей. Межстрочный интервал, установленный в размере 150 %, в итоге равняется 18 пикселям. Таким образом, браузер добавляет пустой промежуток размером 6 (18 — 12) пикселей между двумя строками текста.

Интерлиньяж в виде числового значения

Каскадные таблицы стилей предлагают еще одну единицу измерения для установки размера межстрочного интервала — обычное числовое значение. CSS-код выглядит следующим образом:

```
line-height: 1.5;
```

После этого значения не нужно указывать единицу измерения. Чтобы определить межстрочный интервал или высоту строки, браузер попросту умножает это число на размер шрифта. Так, если размер шрифта текста составляет 1 em, а высота строки установлена равной 1,5, то расчетное значение межстрочного интервала равно 1,5 em. В большинстве случаев эффект при указании значения 1,5 em или 150 % идентичен.

Однако, поскольку вложенные элементы наследуют значение свойства `line-height`, часто при использовании процентных значений и единиц `em` можно столкнуться с проблемами.

Например, вы присваиваете свойству `line-height` элемента `body` значение 150%. Все элементы веб-страницы унаследуют его. Однако наследуется не процентное, а *рассчитанное* значение межстрочного интервала. Допустим, основной размер шрифта установлен равным 10 пикселям; 150 % от 10 пикселей составляет 15. Все элементы унаследуют межстрочный интервал размером 15 пикселей. Так, если на веб-странице есть абзац текста со шрифтом высотой 36 пикселей, его межстрочный интервал размером 15 пикселей будет намного меньше самого текста, а строки сольются вместе.

В этом примере вместо высоты строки 150 % для элемента `body` лучше установить общий для всех элементов пропорциональный базовый межстрочный интервал в размере 1,5. Любой элемент, вместо того чтобы наследовать точное абсолютное значение высоты строки в пикселях от стиля `body`, умножает размер своего шрифта на этот коэффициент. Так, в вышеупомянутом примере, где абзац текста отображен размером шрифта 36 пикселей, межстрочный интервал составит: $1,5 \times 36 = 54$ пикселя.

Другими словами, для установки межстрочного интервала вместо процентных значений и единиц `em` лучше использовать обычные числовые значения.

Выравнивание текста

Одним из самых быстрых способов изменить внешний вид веб-страницы является *выравнивание (выключка)* текста. Используя свойство `text-align`, вы можете расположить абзац в центре веб-страницы, вдоль левого или правого края или выровнять по ширине (формату) (подобно тексту этой книги). Чтобы устанавливать выравнивание для текста, пользуйтесь одним из следующих ключевых слов: `left`, `right`, `justify`, `center`.

```
text-align: center;
```

Выровненный текст замечательно выглядит на печатной странице главным образом потому, что при большой разрешающей способности печати учитываются даже мельчайшие настройки интервалов, а также потому, что большинство программ предпечатной подготовки могут переносить слова (тем самым пытаясь равномерно распределить символы по строкам). Это предотвращает большие промежутки между абзацами. Веб-страницы в форматировании ограничены возможностями намного более грубой регулировки интервалов из-за низкой разрешающей способности мониторов компьютеров и из-за того, что браузеры не поддерживают перенос слов. Когда вы пользуетесь ключевым словом `justify` для выключки по формату, получаются совершенно разные промежутки между словами, текст становится трудночитаемым. Поэтому, применяя такое выравнивание на веб-страницах, убедитесь в том, что получившийся текст читабелен.

Отступ первой строки и изменение интервала между абзацами

В большинстве печатных изданий первая строка каждого абзаца имеет так называемый абзацный отступ. Он выделяет начало каждого абзаца текста, если нет дополнительных промежутков или увеличенных межстрочных интервалов, визуально разделяющих абзацы. В веб-страницах во Всемирной паутине нет отступов, но применяется интервал между абзацами, как в книге.

Если у вас есть желание придать веб-страницам индивидуальность, сделать их непохожими на другие, то воспользуйтесь преимуществами таких свойств CSS, как `text-indent` и `margin`. С их помощью вы можете создать отступ первой строки абзацев и удалить (или увеличить) интервал между ними.

Отступ первой строки

Для установки отступа первой строки абзаца можно использовать такие единицы измерения, как пиксели и `em`:

```
text-indent: 25px;
```

или

```
text-indent: 5em;
```

Значения в пикселах — абсолютные значения, точное число, в то время как `em` определяет размер отступа в количестве символов (базируется на текущем размере шрифта).

СОВЕТ

В свойстве абзацного отступа `text-indent` вы можете использовать отрицательные значения для создания выступа, то есть абзаца с висячей строкой (выступающей влево по отношению к абзацу). Обычно отрицательное значение абзацного отступа используется вместе с указанием значения поля, чтобы отрицательный абзацный отступ не выходил за левую сторону страницы, колонки или блока разметки.

Вы можете также использовать процентные значения, но со свойством `text-indent` эти единицы измерения приобретают другое значение. В данном случае размер выступа, установленный в процентах, связан не со шрифтом текста, а с шириной элемента, в который заключен абзац. Например, если текстовый отступ установлен равным 50 % и абзац охватывает всю ширину окна браузера, то первая строка будет начинаться посередине экрана. Если вы меняете размеры окна, то изменяется ширина абзаца и, соответственно, отступ (про значения свойств, устанавливаемые в процентах, и о том, как они взаимодействуют с шириной элементов веб-страницы, читайте далее в этой главе).

ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Сокращенный метод форматирования текста

Многочисленный повторный набор свойств стилей — достаточно утомительное занятие, особенно если нужно использовать несколько различных текстовых свойств сразу. На этот случай в языке CSS есть свойство `font`, облегчающее написание стилей. Оно позволяет объединять несколько свойств в одну строку: `font-style`, `font-variant`, `font-weight`, `font-size`, `line-height` и `font-family`. Рассмотрим, к примеру, следующее объявление:

```
font: italic bold small-caps 18px/1.5 Arial, Helvetica, sans-serif;
```

Оно приводит к созданию полужирного курсивного текста, набранного капителью, с размером шрифта 18 пикселей, семейства Arial (или Helvetica, или другого рубленого шрифта), с межстрочным интервалом 150 %. Запомните следующие правила.

- Не обязательно применять все эти свойства, но *нужно* включить размер шрифта и семейство шрифтов:

```
font: 1.5em Georgia, Times, serif;
```

- Используйте по одному пробелу после каждого значения свойства, а запятую только для того, чтобы разделить шрифты в списке:

```
Arial, Helvetica, sans-serif
```

- Определяя межстрочный интервал, добавляйте слеш после размера шрифта, за которым должно следовать значение межстрочного интервала:

```
1.5em/1.5
```

- Последние два свойства должны быть следующими: `font-size` (или `font-size/line-height`), а затем `font-family`; все остальные могут быть перечислены в любом порядке. Например, оба объявления равнозначны и к ним применен одинаковый эффект:

```
font: italic bold small-caps 1.5em Arial;
font: bold small-caps italic 1.5em Arial;
```

- Наконец, исключение значения из списка означает то же, что его установка по умолчанию. Допустим, вы создали стиль `p`, который форматирует все абзацы полужирной курсивной капителью и межстрочным интервалом 2000 % (совсем *не обя-*

зательно это повторять). Затем создали класс с именем, скажем, `.special-Paragraph` с таким объявлением стиля шрифта:

```
font: 1.5em Arial;
```

- После этого применили его к какому-либо абзацу имеющегося текста. В результате теперь наш аб-

зац *не* унаследует полужирную курсивную капитель и межстрочный интервал. Исключение этих четырех значений из стиля `.specialParagraph` можно приравнять к написанию следующего кода:

```
font: normal normal normal 1.5em/normal Arial;
```

Настройка полей между абзацами

Многие веб-дизайнеры не любят дополнительные поля (так называемый воздух), которые любой браузер добавляет между абзацами. До появления каскадных таблиц стилей с этим приходилось мириться. Теперь можно воспользоваться свойствами `margin-top` и `margin-bottom` для удаления (или увеличения) этих промежутков. Чтобы полностью избавиться от верхнего и нижнего полей, используйте следующий код:

```
margin-top: 0;
margin-bottom: 0;
```

Чтобы удалить поля между *всеми* абзацами веб-страницы, создайте такой стиль:

```
p {
  margin-top: 0;
  margin-bottom: 0;
}
```

Для установки значений абзацных полей, как и для отступов, вы можете применять такие единицы измерения, как пиксели или `em`. Можно также использовать проценты, но, как и в случае с абзацными отступами, процентные значения относятся к *ширине* элемента, в который заключен абзац. Во избежание путаницы, связанной с вычислением верхнего и нижнего полей, расчет которых базируется на ширине абзацев, проще применять значения в `em` или пикселах.

ПРИМЕЧАНИЕ

Поскольку не все браузеры обрабатывают верхнее и нижнее поля заголовков и абзацев согласованно, рекомендуется обнулить (то есть удалить) все поля в начале таблицы стилей. Посмотреть на то, как это работает, можно в подразделе «С чистого листа» раздела «Управление каскадностью» главы 5.

Для создания специальных эффектов можно назначить *отрицательное* значение верхнему или нижнему полю между абзацами. Например, верхняя граница, установленная равной 10 пикселям, приподнимает абзац выше на 10 пикселей, возможно даже визуально накладывая его на вышестоящий элемент веб-страницы.

Буквица и форматирование первой строки абзаца

Каскадные таблицы стилей позволяют форматировать абзацы с использованием псевдоэлементов `::first-line` и `::first-letter` (рис. 6.16). С технической точки зрения, это не свойства, а селекторы, определяющие фрагмент, к которому применены

CSS-свойства. С помощью псевдоэлемента `::first-letter` можно отформатировать начальную прописную букву или буквицу, имитируя рукописный стиль текста. Например, чтобы сделать первый символ каждого абзаца полужирным и выделить его красным цветом, необходим следующий код:

```
p::

```

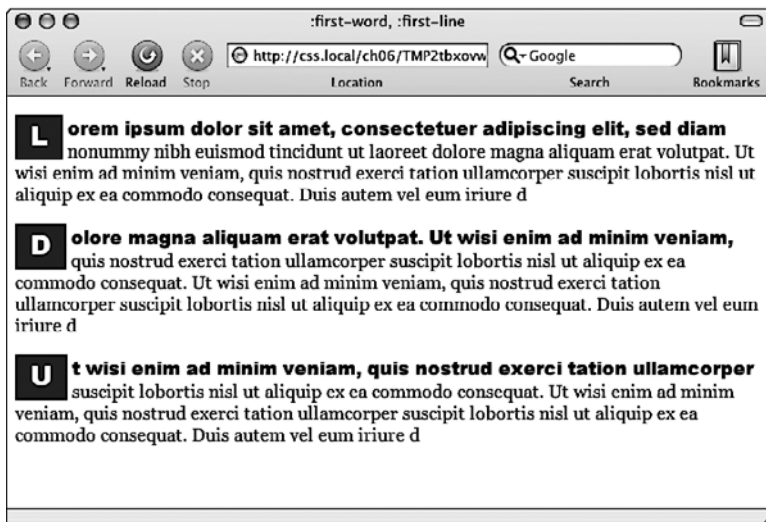
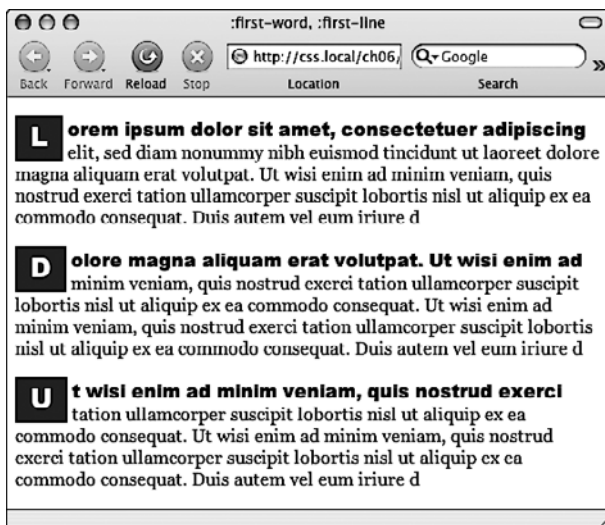


Рис. 6.16. Псевдоэлемент `::first-letter` изменяет первый символ формируемого элемента, например заглавные буквы абзацев. С другой стороны, селектор `::first-line` определяет стиль первой строки абзаца. Даже если посетители вашего сайта изменят размер окна (*внизу*), браузер будет форматировать каждое слово в первой строке абзаца

Для избирательного форматирования, скажем, только первого символа определенного абзаца можно применить класс, например `.intro`:

```
<p class="intro">Текст вводного абзаца начинается здесь...</p>
```

Затем вы можете создать стиль `.intro::first-letter`. Псевдоэлемент `::first-line` форматирует первую строку абзаца. Вы можете применить его к любому фрагменту текста, будь то заголовок (`h2::first-line`) или абзац (`p::first-line`). Как и в случае с `::first-letter`, можно применить класс к единственному абзацу и отформатировать только его первую строку. Допустим, вы хотите отобразить прописными буквами первую строку первого абзаца на странице. Примените класс к HTML-коду первого абзаца: `<p class="intro">` — и напишите следующий код:

```
.intro::first-line { text-transform: uppercase; }
```

ПРИМЕЧАНИЕ

По какой-то странной причине браузеры Chrome и Safari не распознают свойство `text-transform`, когда оно используется с псевдоэлементом `::first-line`. Другими словами, вы не можете применять в Chrome и Safari каскадные таблицы стилей для преобразования букв первой строки абзаца в прописные.

Форматирование списков

Элементы `ul` и `ol` создают маркированные и нумерованные списки: взаимосвязанных элементов, пунктов или пронумерованных шагов, последовательности действий. Однако, как вы, наверное, заметили, не всегда подходит predetermined браузером способ форматирования. Возможно, вы захотите заменить в маркированных списках стандартный маркер собственным, более красивым, использовать в нумерованных списках буквы вместо чисел и т. д.

Типы списков

Большинство браузеров отображают маркированные списки (элементы `ul`), используя маркеры в виде окружности, а нумерованные списки (`ol`) — предваряя пункты числами. С помощью каскадных таблиц стилей вы можете выбрать маркеры трех типов: `disc` (сплошной кружок), `circle` (полый кружок), `square` (сплошной квадрат). Для нумерованных списков предусмотрено шесть вариантов-схем нумерации: `decimal`, `decimal-leading-zero`, `upper-alpha`, `lower-alpha`, `upper-roman`, `lower-roman` (рис. 6.17). Все эти варианты можно выбрать, используя свойство `list-style-type` каскадных таблиц стилей:

```
list-style-type: square;
```

или

```
list-style-type: upper-alpha;
```

Числа можно заменить буквами греческого алфавита — α , β , γ , воспользовавшись значением `lower-greek`. Существует множество других схем нумерации, включая армянский, грузинский, катакана и другие региональные варианты. Информацию о них можно найти по адресу tinyurl.com/pmkpsj9.

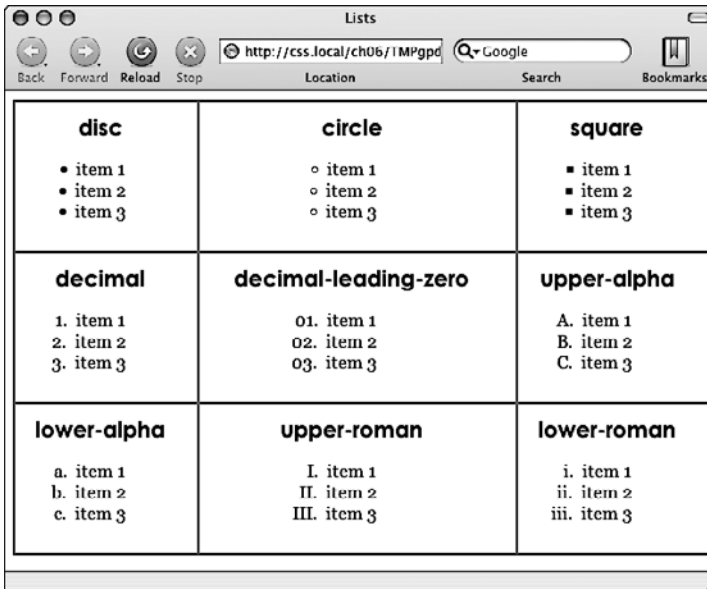


Рис. 6.17. Каскадные таблицы стилей предоставляют множество различных способов маркировки нумерованных и нумерованных списков, начиная с набора геометрических фигур и заканчивая различными системами счисления

Чаще всего это свойство используют при определении стилей, форматирующих элементы `ol` или `ul`. Типичные примеры — включение свойства в стили тега `ol` или `ul`: `ul { list-style-type: square; }` либо в класс, применяемый к одному из этих элементов. Тем не менее вы можете применить это свойство и к отдельно взятому элементу списка (`li`). Вы даже можете применить несколько стилей с различными маркерами к отдельным пунктам-элементам одного и того же списка. Например, можете создать стиль тега `ul`, устанавливающий маркеры в виде квадратов, а затем класс `.circle`, который изменяет тип маркера на полые кружки:

```
li {list-style-type: square; }
.circle { list-style-type: circle; }
```

Теперь примените класс к элементам списка через один для чередования квадратных и круглых маркеров:

```
<ul>
<li>Элемент 1</li>
<li class="circle">Элемент 2</li>
<li>Элемент 3</li>
<li class="circle">Элемент 4</li>
</ul>
```

Или, используя рассмотренный в главе 3 селектор `nth-of-type`, можно вообще избавиться от имени класса:

```
li {list-style-type: square; }
li:nth-of-type(odd) { list-style-type: circle; }
```

Иногда может понадобиться скрыть маркеры, например, если вы захотите использовать собственные графические значки (см. практикум этой главы). Кроме того, когда панель навигации сайта представляет собой список ссылок, вы также можете использовать список `ul`, скрыв его маркеры (см. подраздел «Использование маркированных списков» раздела «Создание панелей навигации» главы 9). Чтобы отключить отображение маркеров, используйте ключевое слово `none`:

```
list-style-type: none;
```

Позиционирование маркеров и нумерации списков

Как правило, браузеры отображают маркеры или числа слева от текста элементов списка (рис. 6.18, *слева*). Благодаря каскадным таблицам стилей вы можете управлять размещением маркеров, используя свойство `list-style-position`. Установить местоположение можно вне (стандартный способ отображения браузерами, см. рис. 6.18, *слева*) или внутри текстовых блоков элементов списка (см. рис. 6.18, *справа*):

```
list-style-position: outside;
```

ИЛИ

```
list-style-position: inside;
```

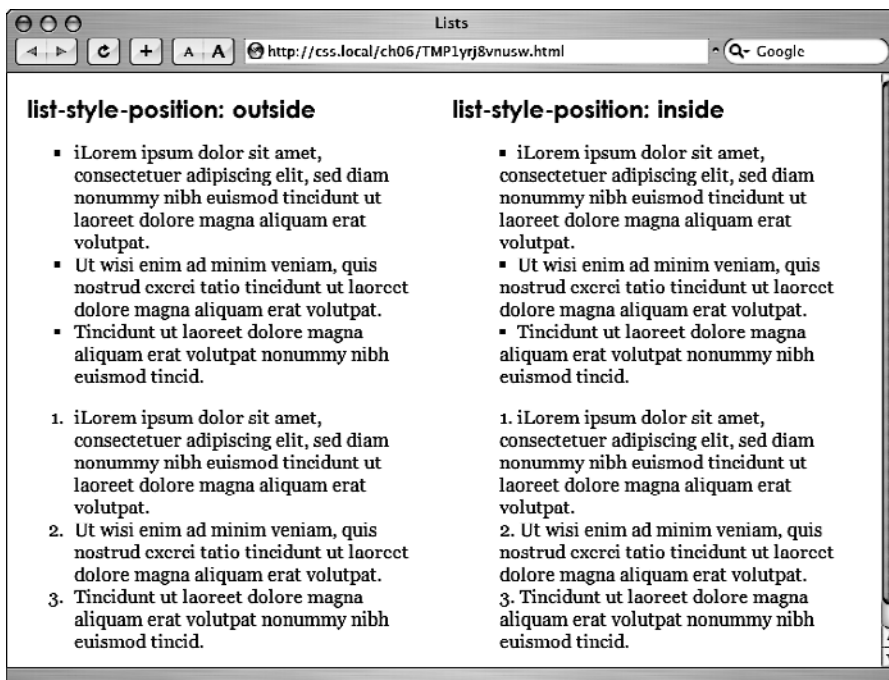


Рис. 6.18. Используя свойство `list-style-position`, вы можете управлять позиционированием маркеров и чисел в списках. Значение `outside` визуально выделяет список и каждый его элемент из всего текста. Значение `inside` обеспечит списку максимальную ширину на странице

СОВЕТ

Вы можете изменить промежуток между маркером и текстом, увеличив или уменьшив значение свойства `padding-left` (см. раздел «Управление размерами полей и отступов» главы 7). Чтобы использовать его, вам нужно создать стиль для элементов `li`. Этот способ работает только в том случае, если свойство `list-style-position` определено со значением `outside` (или вообще отсутствует).

Кроме того, если вам не нравится, что браузеры создают для списков отступ, сдвигая все содержимое вправо, можете переопределить стиль, присвоив свойствам `margin-left` и `padding-left` значение 0. Чтобы удалить отступ, можно создать следующий групповой селектор:

```
ul, ol {
  padding-left: 0;
  margin-left: 0;
}
```

Вы можете создать класс с такими свойствами и применить его к конкретным элементам `ul` или `ol`. Рекомендуется указать значения обоих свойств, `padding` и `margin`, по той причине, что одни браузеры для управления отступом используют свойство `padding` (Firefox, Mozilla, Safari), а другие — `margin` (Internet Explorer). Подробно про свойства `padding` и `margin` вы можете прочесть в следующей главе.

В обычном порядке браузеры отображают пункты маркированных списков друг за другом, без дополнительного промежутка. Добавить интервал между ними можно, применяя свойства `margin-top` и `margin-bottom` к конкретным элементам списка. Они работают с интервалом между пунктами списков точно так же, как с абзацами. Единственное, вы должны удостовериться в том, что стиль применяется к элементу `li`: создайте класс и примените его индивидуально к каждому элементу. Стиль *не* должен применяться к элементам `ul` или `ol`. Добавление верхнего или нижнего полей к этим элементам увеличивает промежуток между всем списком и абзацами выше или ниже его. На интервал между элементами они не оказывают никакого влияния.

Графические маркеры

Если вам недостаточно стандартных маркеров квадратной и круглой формы, вы можете создать свои собственные. Используя графический редактор, например Photoshop или Fireworks, можно быстро создать красочные и интересные маркеры. В качестве источников также можно рассмотреть встроенные в программы клипарты и символные шрифты (такие как Webdings).

Свойство `list-style-image` позволяет определить путь к графическому символу на сервере таким же образом, как вы указываете местонахождение файла с изображением, используя атрибут `src` HTML-элемента `img`. Синтаксис команды следующий:

```
list-style-image: url(images/bullet.gif);
```

Значение `url` и круглые скобки обязательны. Часть, заключенная в круглые скобки, — в данном примере `images/bullet.gif` — это и есть путь к графическому символу. Обратите также внимание на то, что путь в кавычки заключать не нужно.

ПРИМЕЧАНИЕ

Указывая путь или адрес к графическим файлам (изображениям, графическим символам) во внешней таблице стилей, имейте в виду, что путь должен указываться относительно таблицы стилей, а не веб-страницы. Более подробно об этом вы прочтете в разделе «Добавление фоновых изображений» главы 8, когда мы начнем использовать графику.

Свойство `list-style-image` позволяет использовать графические символы в качестве маркеров. Однако оно не позволяет управлять их положением. Маркер может оказаться слишком высоко или низко расположенным относительно пункта списка. Придется редактировать сам графический символ маркера, пока он не будет сочетаться со списком. О более грамотном подходе вы узнаете в главе 8. Он основан на использовании свойства `background-image`. Это свойство позволяет точно позиционировать графические элементы, в том числе маркеры в списках.

ПРИМЕЧАНИЕ

Как и в случае со свойством `font` (см. врезку «Для опытных пользователей» ранее в этой главе), может применяться сокращенный метод указания атрибутов списков. В свойстве `list-style` могут быть перечислены все настройки форматирования списка, в том числе `list-style-image`, `list-style-position` и `list-style-type`. Например, стиль `ul { list-style: circle inside; }` отобразит списки с маркерами в виде полых кружков, расположив их внутри списка. Когда вы описываете стиль списка с одновременным указанием типа маркера и графического символа — `list-style: circle url(images/bullet.gif) inside;` — и графический символ не может быть обнаружен по заданному пути, браузер будет использовать предопределенный маркер, в данном случае полый кружок.

ЧАВО

Настройка маркеров и чисел в списках

Я хочу отформатировать числа нумерованных списков так, чтобы они отображались полужирным шрифтом красного цвета вместо надоевшего черного. Как можно настроить внешний вид маркеров и чисел?

Каскадные таблицы стилей предлагают несколько способов настройки параметров маркеров, предваряющих пункты — элементы списка. Вы можете использовать собственные графические символы, как описано выше. Более совершенный способ, экономящий объем CSS-кода, состоит в том, чтобы использовать так называемый *генерированный контент*. Этим термином обозначаются, по сути, элементы, отсутствующие в исходном коде и автоматически добавляемые браузером при отображении страницы. Хороший пример — сами маркеры. Вы не указываете значки маркеров при создании списка — они добавляются на веб-страницу автоматически. С помощью каскадных таблиц стилей можно сообщить браузеру, чтобы он генерировал, добавлял такое содержимое и даже форматировал должным образом все, что находится перед текстом пунктов списка — `li`. Вы узнали о генерируемом контенте из

главы 3 и теперь можете сделать обычные маркеры списка красными, добавив в свою таблицу стилей следующий CSS-код:

```
ul li {
  list-style-type: none;
}
ul li:before {
  content: counter(item, disc) " ";
  color: red;
}
```

А если нужно красным цветом отформатировать числа нумерованного списка, можно добавить такой CSS-код:

```
ol li {
  list-style-type: none;
  counter-increment: item;
}
ol li:before {
  content: counter(item) ". ";
  color: red;
}
```

Как придать стиль нумерованному списку, подробно написано по адресу tinyurl.com/qcevfy7.

Практикум: форматирование текста

В этом разделе мы попрактикуемся в форматировании таких элементов веб-страниц, как заголовки, списки, абзацы текста, используя мощные средства языка CSS.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке **06**.

Настройка параметров страницы

Начнем с таблицы стилей и добавим правило `@font-face`, чтобы загрузить некоторые веб-шрифты для форматирования основного текста страницы.

1. Запустите браузер и откройте файл `text.html` (рис. 6.19).

Здесь пока особо не на что смотреть — есть только несколько заголовков, абзацев и один маркированный список. Но скоро вы заметно преобразите эту страницу.

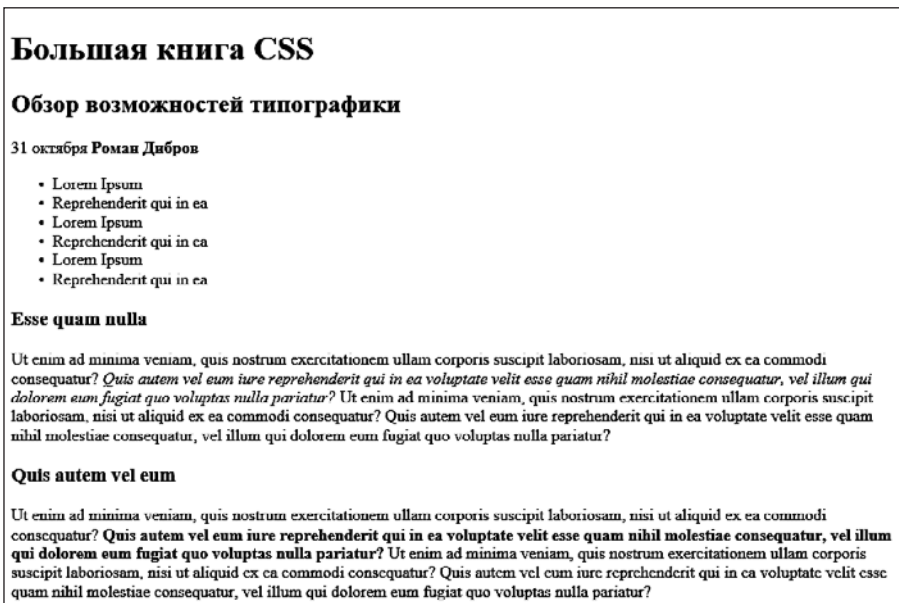


Рис. 6.19. Работа над любой веб-страницей начинается с обычного HTML-контента

2. Откройте файл `text.html` в редакторе HTML-кода.

В этом проекте вы будете использовать как собственные файлы шрифтов, так и службу Google Fonts. Начнем с добавления ссылки на шрифт Google.

3. В разделе заголовка веб-страницы щелкните кнопкой мыши сразу после закрывающего тега `</title>`. Нажмите клавишу **Enter** и наберите код:

```
<link href="http://fonts.googleapis.com/css?family=Slabo+27px" rel="stylesheet">
```

Этот код говорит серверу Google отправить вам антиквенный (с засечками) шрифт Slabo размером 27px. Далее вы добавите ссылку на внешнюю таблицу стилей, которая будет основой для стилей данного урока.

4. После только что добавленного элемента `link` введите следующий код:

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

Вы только что связали HTML-файл с внешней таблицей стилей, расположенной в папке `css`. В этой папке находятся внешняя таблица стилей и веб-шрифты.

5. Откройте файл `styles.css`, расположенный в папке `css`. Эта таблица стилей содержит базовый набор сброшенных стилей (см. раздел «Управление каскадно-стью» главы 5).

Если вы просмотрите файл `text.html` в браузере, то увидите, что текст на странице (заголовки, абзацы и т. д.) выглядит примерно одинаково, то есть все стандартное HTML-форматирование браузера было устранено и теперь можно начинать с чистого листа.

Затем нужно добавить необходимые правила `@font-face` для загрузки четырех веб-шрифтов. Все они описывают один и тот же шрифт `PTSans`, но включают полужирное, курсивное и полужирно-курсивное начертание.

6. В верхней части файла `styles.css` добавьте следующий код (перед кодом сброса стилей) (если не хотите вводить весь этот код, откройте файл `at-font-face.css`, расположенный в папке с учебными файлами, скопируйте и вставьте его в файл `styles.css`):

```
@font-face {
  font-family: 'PTSans';
  src: url('fonts/PTSansRegular.woff2') format('woff2'),
  url('fonts/PTSansRegular.woff') format('woff');
  font-weight: normal;
  font-style: normal;
}
@font-face {
  font-family: 'PTSans';
  src: url('fonts/PTSansItalic.woff2') format('woff2'),
  url('fonts/PTSansItalic.woff') format('woff');
  font-weight: normal;
  font-style: italic;
}
@font-face {
  font-family: 'PTSans';
  src: url('fonts/PTSansBold.woff2') format('woff2'),
  url('fonts/PTSansBold.woff') format('woff');
  font-weight: bold;
  font-style: normal;
}
@font-face {
  font-family: 'PTSans';
  src: url('fonts/PTSansBoldItalic.woff2') format('woff2'),
  url('fonts/PTSansBoldItalic.woff') format('woff');
```

```
font-weight: bold;
font-style: italic;
}
```

Если вкратце, то вы создали новое семейство шрифтов PTSans, которое сможете использовать в любых создаваемых вами новых стилях. Для добавления полужирно-курсивных вариантов используется метод, рассмотренный ранее. Суть его в том, что в области применения обычного элемента полужирного начертания (например, в заголовках или при использовании элемента `strong`) браузер будет автоматически переключаться на полужирную версию шрифта.

Вы добавили правило `@font-face` в начале внешней таблицы стилей. Не обязательно, но рекомендуется поместить все шрифты в верхней части, а стили — в нижней.

Затем вы создадите стиль, определяющий ряд общих свойств для всего текста на странице.

7. В нижней части таблицы стилей, после стилей сброса, перейдите на новую строку и введите `html {`.

Это основной селектор тега, применяемый к элементу `html`. В главе 4 мы выяснили, что прочие элементы наследуют свойства этого элемента. Можно настроить ряд основных характеристик текста, таких как шрифт, цвет и размер для их использования последующими элементами в качестве их исходных настроек.

8. Снова нажмите клавишу **Enter** и добавьте следующие два свойства:

```
font-family: PTSans, Arial, sans-serif;
font-size: 62.5%;
```

Эти две инструкции устанавливают шрифт PTSans (или Arial, если браузер не может загрузить PTSans) и значение для размера шрифта 62.5%.

ПРИМЕЧАНИЕ

Зачем нужно устанавливать базовый размер шрифта веб-страницы равным 62,5 %? Оказывается, если умножить 62,5 % на 16 пикселей (стандартный размер шрифта текста большинства браузеров), получится 10 пикселей. При таком начальном размере шрифта очень просто вычислить размеры всех последующих и представить, как они будут выглядеть на экране монитора. Например, 1,5em будет равняться 1,5 × 10, или 15 пикселей, 2em — 20 пикселей и т. д. Считать очень просто, умножая размер в em на 10. Вы также можете использовать единицы rem (описанные в разделе «Форматирование символов и слов» данной главы), которые похожи на em, но позволяют избежать проблемы, связанной с наследованием вложенных единиц em.

9. Завершите определение текущего стиля, нажав клавишу **Enter** и набрав закрывающую фигурную скобку для указания окончания стиля.

На данном этапе стиль должен выглядеть следующим образом:

```
html {
font-family: PTSans, Arial, sans-serif;
font-size: 62.5%;
}
```

Ваша таблица стилей закончена.

10. Сохраните файл `styles.css` и откройте файл `text.html` для просмотра в браузере, чтобы увидеть результат работы.

Текст на странице изменил свой цвет и шрифт. Он стал действительно маленьким. Не волнуйтесь, это из-за размера 62,5 %, который вы установили на шаге 8. Это начальный параметр для всего текста, и вы сможете спокойно увеличить текст, определяя размеры в единицах `em` для других элементов.

Форматирование заголовков и абзацев

Теперь, когда основное форматирование текста выполнено, пришло время улучшить вид заголовков и абзацев.

1. Вернитесь к файлу `styles.css` в редакторе HTML-кода. Установите курсор после закрывающей скобки селектора тега `html`, добавьте новую строку и наберите код `.main h1 {`.

Это *селектор потомков*, более специфичный по сравнению с базовым селектором тега `html`. В данном случае селектор указывает браузеру «применить соответствующее форматирование» к любому элементу `h1`, находящемуся внутри другого элемента с классом `main`. Если вы просмотрите HTML-код веб-страницы, то увидите, что там присутствует элемент `div` с классом `main` (`<div class="main">`). Как вы узнаете позже, при разметке дизайнов, основанной на каскадных таблицах стилей, достаточно распространено группирование HTML-элементов внутри контейнеров `div`. Вы сможете размещать отдельные элементы `div` для создания колонок и других блоков разметки страниц. Распространено также использование селекторов потомков наподобие этого для точного определения свойств форматирования с воздействием лишь на элементы в определенных областях страницы.

2. Нажмите клавишу `Enter` и наберите три свойства:

```
color: rgb(249,212,120);  
font-family: "Arial Black", Arial, Helvetica, sans-serif;  
font-size: 4em;
```

Вы только что изменили цвет заголовков `h1`, а также их шрифт. На этот раз вместо веб-шрифта указан шрифт, который может быть установлен на компьютере посетителя. Шрифт `Arial Black` установлен на многих компьютерах, но если у какого-нибудь посетителя он отсутствует, браузер воспользуется шрифтом `Arial` или `Helvetica` либо другим рубленным шрифтом. Кроме того, вы установили размер шрифта равным `4em`, что для большинства браузеров составляет 40 пикселей (если, конечно, посетитель не изменял параметры шрифтов в настройках своего браузера). Это все благодаря значению 62,5%, установленному ранее на шаге 7. Таким образом, базовый размер шрифта стал составлять 10 пикселей в высоту, а вычисление 4×10 задает размер 40 пикселей. Затем к заголовку будет добавлена тень.

3. Завершите текущий стиль, нажав клавишу `Enter` и добавив код, выделенный ниже полужирным шрифтом (не забудьте указать закрывающую скобку):

```
.main h1 {  
  color: rgb(249,212,120);  
  font-family: "Arial Black", Arial, Helvetica, sans-serif;
```

```
font-size: 4em;
text-shadow: 4px 4px 6px rgba(0,0,0,.75);
}
```

Выделенной строкой кода добавлена тень, смещенная на 4 пиксела вправо, на 3 пиксела ниже и с размытостью 6 пикселов. Кроме этого, используется RGBA-цвет, устанавливающей для тени черный цвет и 75%-ный уровень прозрачности.

4. Сохраните файл и просмотрите HTML-страницу в браузере.

Теперь изменим внешний вид остальных заголовков и абзацев.

ПРИМЕЧАНИЕ

При открытии файла браузер сохраняет его в хранилище на жестком диске компьютера посетителя, которое называется кэшем. Если браузеру вновь потребуется тот же файл, то вместо повторной загрузки с сервера он обратится к нему в кэше. Однако кэшированный файл может быть неактуален. Браузер может загрузить старую кэшированную версию вашей внешней таблицы стилей вместо обновленной. Если после обновления внешней таблицы стилей страница выглядит по-прежнему, в браузере нажмите клавишу F5 или сочетание клавиш Ctrl+Shift+R (⌘+Shift+R), чтобы загрузить обновленную версию файла.

5. Вернитесь к файлу `styles.css` в редакторе HTML-кода. Установите курсор после закрывающей скобки стиля `.main h1`, нажмите клавишу `Enter` и добавьте следующие стили:

```
.main h2 {
  font: normal 3.5em "Slabo 27px", Garamond, Times, serif;
  color: rgb(37,76,143);
  border-bottom: 1px solid rgb(200,200,200);
  margin-top: 25px;
}
```

Здесь используется еще один селектор потомков, который относится только к заголовкам `h2`, находящимся внутри другого элемента с классом `main`. Свойство `font` сокращает количество кода, объединяя в себе правила `font-weight`, `font-size` и `font-family`. Другими словами, одна строка кода форматирует заголовок обычным начертанием, устанавливает для него высоту 3.5em и определяет шрифт.

Кроме того, этот стиль изменяет цвет заголовков, добавляет декоративную границу под заголовком и немного пространства над заголовком.

Пришло время взяться за другие заголовки.

6. Добавьте новый стиль под тем, который вы создали на предыдущем шаге:

```
.main h3 {
  color: rgb(241,47,6);
  font-size: 1.9em;
  font-weight: bold;
  text-transform: uppercase;
  margin-top: 25px;
  margin-bottom: 10px;
}
```

Он настраивает некоторые свойства обычного форматирования: цвет, размер шрифта, насыщенность, а также использует свойство `text-transform`, чтобы текст заголовка `h3` был оформлен прописными буквами. Наконец, он добавляет немного пространства сверху и снизу заголовка с помощью свойства `margin`.

Далее вы улучшите внешний вид абзацев.

7. Добавьте новый стиль на страницу:

```
.main p {
  font-size: 1.8em;
  line-height: 1.5;
  margin-left: 150px;
  margin-right: 50px;
  margin-bottom: 10px;
}
```

Этот стиль содержит свойство `line-height`, которое устанавливает расстояние между строками. Значение `1.5` является коэффициентом: для вычисления высоты строки размер шрифта (`1.75em`) умножается на величину `1.5`. Обычно коэффициент увеличивает высоту строки в `1,5` раза, или на `150 %` относительно размера шрифта. Стиль увеличивает интервал между строками абзаца. Благодаря этому дополнительному пространству текст располагается свободнее, а предложения станowiąтся легче читать (но только если вы используете латиницу).

Этот стиль также увеличивает размер шрифта до `1,8 em` (18 пикселей для большинства браузеров) и смещает абзац от левого и правого краев страницы. Вы можете заметить, что для свойства `margin` приходится набирать слишком много кода. На этот случай, как вы узнаете в следующей главе, есть сокращенная запись свойства.

Теперь попробуем более совершенный тип селектора.

8. Добавьте следующий стиль в таблицу стилей:

```
.main p::first-line {
  font-weight: bold;
  color: rgb(153,153,153);
}
```

Псевдоэлемент `::first-line` воздействует только на первую строку абзаца. В этом случае именно первая строка текста в каждом абзаце внутри контейнера `div` с классом `main` будет окрашена в серый цвет и выделена полужирным.

9. Сохраните файл `styles.css` и откройте файл `text.html` для просмотра результата в браузере.

На текущий момент ваша веб-страница должна быть похожа на показанную на рис. 6.20.

Форматирование списков

На этой странице есть один маркированный список. Вы переместите его вверх к правому краю страницы и сделаете так, чтобы текст, расположенный после

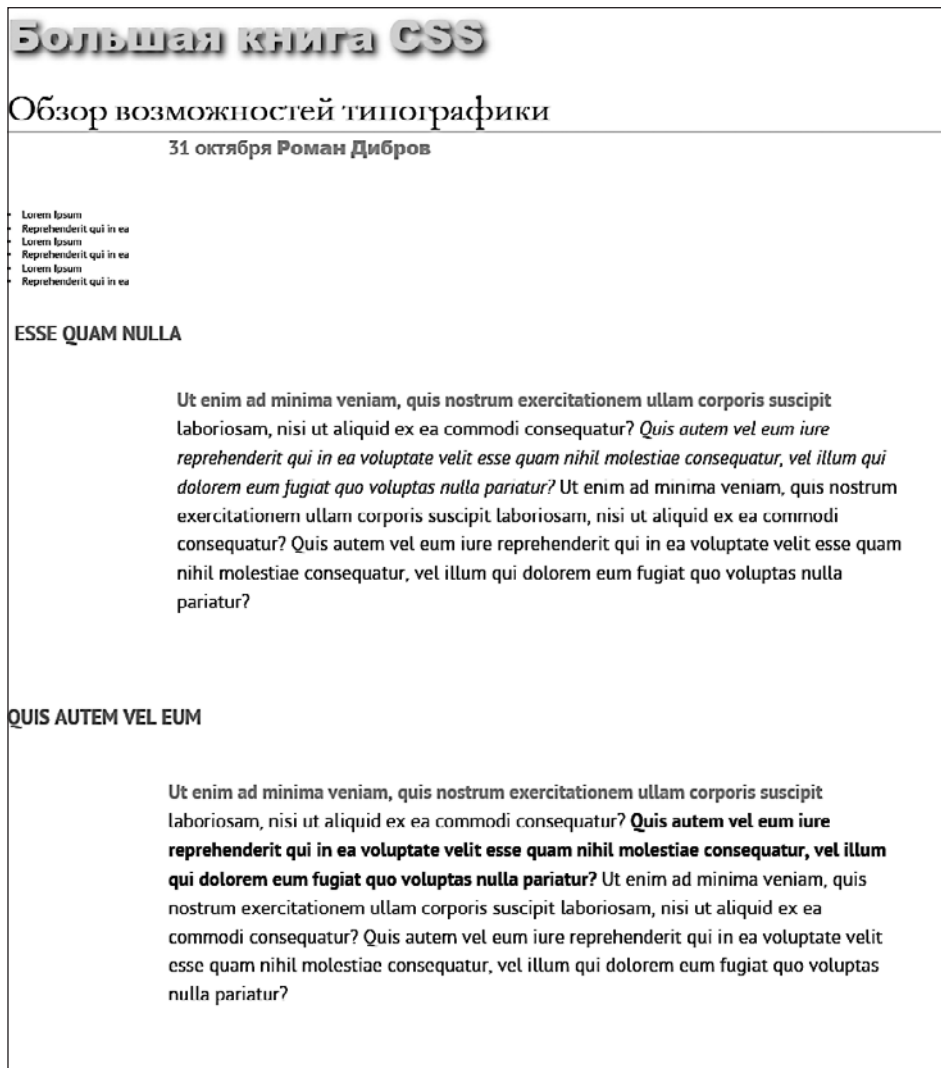


Рис. 6.20. Внешний вид веб-страницы преобразуется: заголовки, абзацы и основной текст приведены в порядок. В зависимости от шрифтов, установленных на вашем компьютере, результат, получившийся у вас, может несколько отличаться

этого списка, обтекал его. С помощью каскадных таблиц стилей сделать это достаточно легко.

1. Вернитесь к файлу `styles.css` в редакторе HTML-кода. Добавьте следующий стиль в конце таблицы стилей:

```
.main ul {
  margin: 50px 0 25px 50px;
  width: 25%;
```



```
float: right;
}
```

При форматировании списков обычно создаются стили для двух разных элементов: непосредственно для самого списка (элемент `ul` для маркированных либо `ol` для нумерованных списков) и отдельных элементов списка (элемент `li`). Этот стиль управляет всем списком.

ПРИМЕЧАНИЕ

В Firefox страница может отображаться в искаженном виде. В этом случае попробуйте использовать другой браузер, например актуальную версию Internet Explorer.

В стиле выполняется несколько действий. Во-первых, свойство `margin` используется в сокращенной записи. Одна строка кода устанавливает все четыре поля вокруг списка, заменяя четыре отдельных свойства полей (`margin-top`, `margin-right` и т. д.). Четыре значения представлены в следующем порядке: верхняя сторона, правая, нижняя и левая. Таким образом, этот стиль устанавливает поле шириной 50 пикселей сверху списка, 0 — справа, 25 — снизу и 50 пикселей — слева.

Свойство `width` задает для всего списка ширину, равную 25 % от ширины окна браузера. Если какой-нибудь пункт списка содержит больше текста, чем может поместиться в пределах этого пространства, он переходит к другой строке.

Свойство `float` по-настоящему волшебное: в данном случае `float: right` означает перемещение списка к правому краю страницы. Это свойство также приводит к тому, что следующий за списком текст обтекает список слева. Это замечательный метод, более подробно о плавающих элементах вы узнаете в главе 7.

Далее вы определите внешний вид отдельных пунктов списка.

2. Добавьте еще один стиль в таблицу:

```
.main li {
  color: rgb(32,126,191);
  font-size: 1.5em;
  margin-bottom: 7px;
}
```

Здесь нет ничего нового: обычное изменение цвета и размера, а также добавление пространства под каждым пунктом списка. Пришло время взглянуть на результат.

ПРИМЕЧАНИЕ

Если вы хотите добавить пространство между пунктами списка, необходимо добавить верхние или нижние поля для элемента `li`. Добавление полей к элементам `ul` или `ol` увеличит пространство вокруг всего списка.

3. Сохраните страницу и воспользуйтесь предварительным просмотром в браузере.

Теперь страница должна быть похожа на ту, что представлена на рис. 6.21.

Большая книга CSS

Обзор возможностей типографики

31 октября Роман Дибров

ESSE QUAM NULLA

Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? *Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?* Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? **Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?**

- Lorem Ipsum
- Reprehenderit qui in ea
- Lorem Ipsum
- Reprehenderit qui in ea
- Lorem Ipsum
- Reprehenderit qui in ea

QUIS AUTEM VEL EUM

Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? **Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?** Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

Рис. 6.21. Свойство float дает интересные возможности при проектировании дизайна.

В данном случае маркированный список перемещается к правому краю страницы. Фактически свойство float настолько полезно, что является основным при компоновке макетов на базе CSS, пример чего мы рассмотрим в главе 12

Точная настройка с классами

Иногда появляется желание иметь еще больше контроля над тем, как применяется стиль. Например, вы хотите видеть большинство абзацев в каком-либо разделе страницы одинаковыми, но, вероятно, пожелаете определить уникальный вид для одного или двух из них. В этом примере абзац рядом с верхней частью страницы — November 30 Rod Dibble — содержит особую информацию (о дате публикации и об авторе). Сделаем так, чтобы этот абзац выделялся среди других, добавив класс в HTML-код и создав для него стиль.

1. Откройте файл `text.html` в редакторе HTML-кода. Найдите вышеупомянутый абзац в HTML-коде (`<p>31 октября Роман Дибров</p>`)

и добавьте `class="byline"` к открывающему тегу `<p>`. HTML-код должен выглядеть так:

```
<p class="byline">31 октября <strong>Роман Дибров</strong></p>
```

Теперь осталось создать класс, замещающий общие свойства форматирования абзацев на этой странице.

2. Откройте файл `styles.css` в редакторе HTML-кода. В нижней части таблицы стилей добавьте следующий код:

```
.main .byline {  
  font-size: 1.6em;  
  margin: 5px 0 25px 50px;  
}
```

Стиль настраивает размер и расположение только одного абзаца.

Обратите внимание, что, если бы вы назвали этот стиль `.byline` (простой селектор класса), он бы не работал. Благодаря правилам каскадности, описанным в предыдущей главе, `.byline` менее специфичен, чем стиль `.main p`, созданный в шаге 7 ранее, поэтому он будет неспособен заместить размер и поля, указанные в стиле `.main p`. А стиль `.main .byline`, в свою очередь, более специфичен.

Форматирование этого абзаца по-прежнему требует доработки. Было бы замечательно, если бы имя выделялось еще больше. В этом случае нам поможет HTML-разметка.

3. Добавьте следующий код в таблицу стилей:

```
.main .byline strong {  
  color: rgb(32,126,191);  
  text-transform: uppercase;  
  margin-left: 11px;  
}
```

Если вы просмотрите HTML-код в шаге 1, то увидите, что имя окружено тегami элемента `strong`. Он используется для того, чтобы выделить текст и пометить его как важный. Но это не значит, что вам нужно позволять ему быть полужирным, как большинство браузеров отображают этот элемент по умолчанию. Напротив, селектор потомков относится к элементу `strong`, но только в тех случаях, когда он появляется внутри другого элемента с классом `.byline` и лишь если они все находятся внутри еще одного элемента с классом `main`, — вот так, очень специфично.

Этот стиль форматирует шрифт синим цветом, превращает буквы в прописные и добавляет немного пространства с левой стороны (немного отодвигая имя от текста `November 30`).

Последние штрихи

Заключительная шлифовка дизайна веб-страницы будет заключаться в добавлении нескольких стилей, форматирующих страницу и контейнер `div` с классом `main`, чтобы они выглядели лучше.

Кроме того, здесь вы реализуете несколько интересных типографических приемов.

1. Вернитесь к файлу `styles.css` в редакторе HTML-кода.

Сначала зададим фоновый цвет и изображение для страницы.

2. Найдите стиль `html` в верхней части таблицы стилей и добавьте новое свойство (изменения выделены полужирным шрифтом):

```
html {  
  font-family: PTSans, Arial, sans-serif;  
  font-size: 62.5%;  
  background: rgb(225,238,253) url(../images/bg_body.png) repeat-x;  
}
```

Свойство `background` представляет собой мощный инструмент для любого веб-дизайнера. Вы уже использовали его пару раз в предыдущих практикумах; оно позволяет добавлять цвет, а также изображения и управлять их расположением в области какого-либо HTML-элемента.

Вы узнаете все тонкости этого свойства в следующей главе, а сейчас учтите, что добавленная строка кода изменяет цвет фона страницы на светло-голубой и добавляет темно-синюю полосу в верхнюю часть страницы.

Далее мы преобразим элемент `div` с классом `main`.

3. Добавьте еще один стиль между `html` и `.main h1`:

```
.main {  
  max-width: 740px;  
  margin: 0 auto;  
  padding: 0 10px;  
  border: 4px solid white;  
  background: transparent url(../images/bg_banner.jpg) no-repeat;  
}
```

Щелкните кнопкой мыши после закрывающей скобки `}` стиля `html`, нажмите клавишу `Enter` и введите код, представленный выше. Вам не обязательно создавать стиль именно в этом месте, чтобы он исправно работал. Однако кажется логичным размещение стиля, управляющего элементом `div`, перед другими стилями, которые форматируют элементы внутри контейнера `div`.

Свойство `max-width` устанавливает максимальную ширину данного контейнера `div`. Это значит, что если у посетителя страницы ширина экрана меньше 740 пикселей, то элемент `div` сожмется. Однако его ширина никогда не превысит 740 пикселей. Значения свойства `margin` здесь — `0 auto` — добавляют 0 пикселей пространства над и под контейнером `div` и устанавливают для правого и левого полей параметр `auto`, располагающий этот элемент в центре окна браузера.

Свойство `padding` добавляет пространство внутри контейнера, смещая контент внутри элемента `div` от его границ. Наконец, мы также поместили изображение в качестве фона контейнера `div`.

Два последних стиля не имеют ничего общего с форматированием текста, но, если вы просмотрите страницу, то увидите, что благодаря им он выглядит намного лучше, за исключением двух заголовков. Первый из них недостаточно жирный, а второй должен появляться под недавно добавленным рисунком.

4. Добавьте следующий код после стиля `.main h1`:

```
.main h1 strong {
  font-size: 150px;
  color: white;
  line-height: 1;
  margin-right: -.5em;
}
```

HTML-код заголовка выглядит следующим образом:

```
<h1><strong>CSS</strong> The Missing Manual</h1>
```

Аббревиатура CSS заключена в теги элемента `strong`, поэтому данный селектор потомков форматирует только этот текст (здесь он подобен стилю, который вы добавили в шаге 3 ранее). Размер шрифта был увеличен, его цвет изменился, а высота строки задана так, что текст теперь вписывается в верхнюю часть страницы.

Вы заметите, что высота строки равна 1, а, как вы читали в начале этой книги, единица `em` основывается на текущем размере шрифта элемента, поэтому в данном случае высота строки будет составлять 150 пикселей — таков размер шрифта данного стиля.

Еще один интересный прием позволяет осуществить свойство `margin-right`, которому присвоено отрицательное значение `-.5em`. Поскольку положительные значения размеров полей отодвигают элементы друг от друга, отрицательные, в свою очередь, сближают. В данном случае остальной текст заголовка (`The Missing Manual`) располагается рядом с аббревиатурой `CSS`.

ПРИМЕЧАНИЕ

Использование отрицательных значений размеров полей разрешено в CSS (хотя это и неочевидный прием).

5. Сохраните таблицу `styles.css` и просмотрите файл `text.html` в браузере.

Веб-страница должна иметь вид, показанный на рис. 6.22. Теперь вы можете сравнить получившийся файл с законченным вариантом `text.html`, который находится в папке `06_finished` учебного материала.

Вы изучили основные способы форматирования текста, предлагаемые каскадными таблицами стилей, и превратили малопривлекательный текст в страницу с отличным дизайном.

В следующей главе мы рассмотрим размещение на веб-страницах графики, границ, полей и применение прочих мощных команд форматирования, предлагаемых каскадными таблицами стилей.

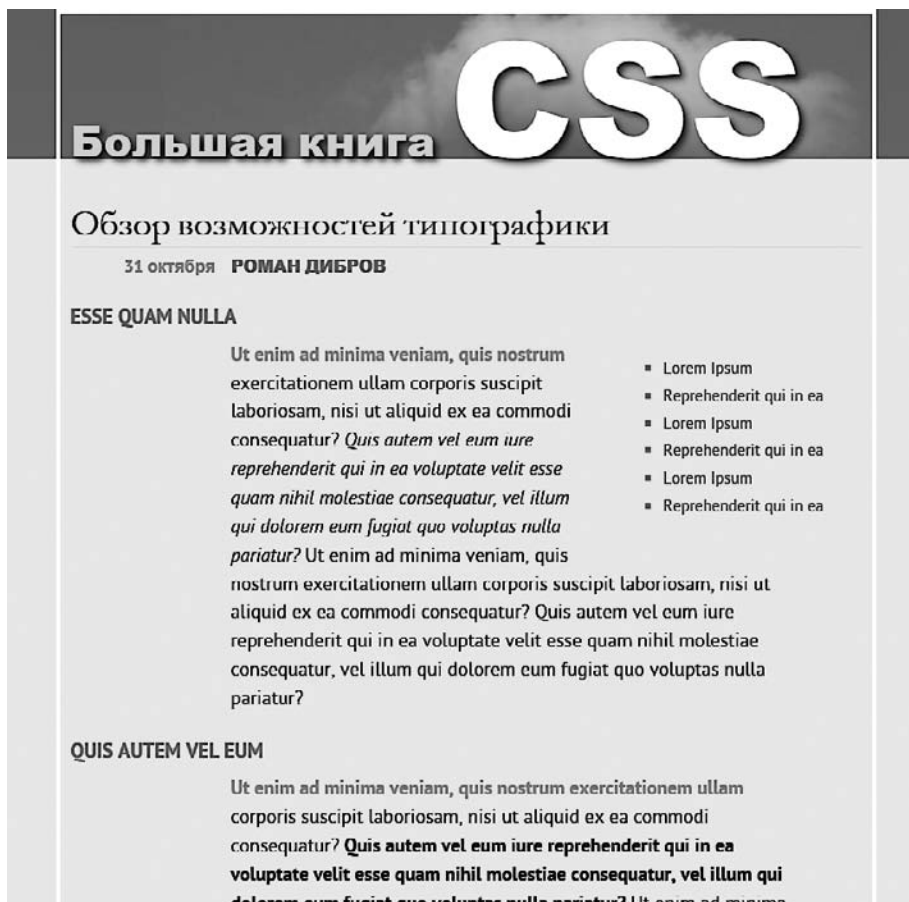


Рис. 6.22. С небольшой помощью языка CSS можно превратить простой текст в документ с профессиональным дизайном

7

Поля, отступы, границы

На любой HTML-элемент воздействует множество свойств каскадных таблиц стилей, определяющих, каким образом он будет отображен браузером. Некоторые из них, например границы и фоновый цвет, наглядны. Другие нельзя определить явно (например, отступы и поля), но они также обеспечивают форматирование. Понимая, как эти атрибуты работают, вы можете создать привлекательные колонки, меню, элементы навигации, а также управлять пространством вокруг них (веб-дизайнеры называют его *воздухом*). Это делается для того, чтобы ваши веб-страницы не казались беспорядочными и нечитаемыми и вообще выглядели профессионально.

Все свойства, описываемые в текущей главе, — основа *блочной модели* CSS, которая представляет одну из важнейших составляющих этого языка.

Понятие блочной модели

Когда вам говорят об абзаце текста или заголовке, вы представляете буквы, слова, предложения. Фотографии, логотипы и другие изображения должны ассоциироваться с элементом `img`. Браузер обрабатывает все элементы как небольшие *блоки*. Для него любой элемент — контейнер с содержимым: текстом, изображением или другими элементами (рис. 7.1). Область внутри границ, которая включает контент и отступы, может также иметь цвет фона. Он находится под границей, поэтому, когда вы используете пунктирную или точечную границу, цвет отображается в промежутках между отрезками или точками.

Блок элемента окружают следующие свойства.

- `padding` — *отступ*, пространство между контентом и границей.
- `border` — *граница*, линия вдоль каждого края блока. Граница может отображаться как для всех сторон сразу, так и для любой из них или комбинации сторон.
- `background-color` — *цвет фона*, заполняет пространство внутри границы, включая область отступа.
- `margin` — *поле*, отделяет один элемент от другого. К примеру, пространство, обычно отображаемое сверху и снизу абзацев текста, — это поля.

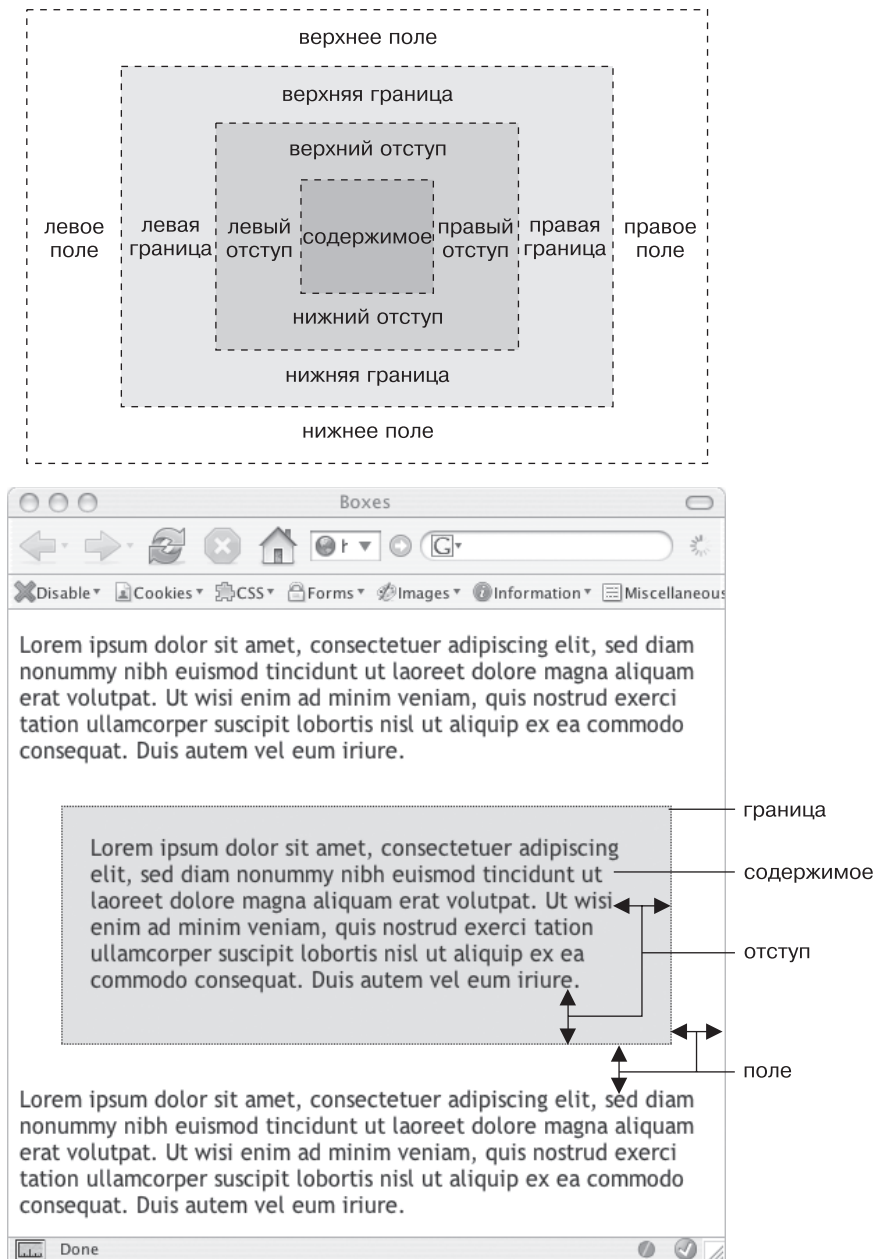


Рис. 7.1. Блочную структуру элемента образует его содержимое (например, несколько предложений текста), а также отступы, границы и поля

Для форматирования элемента можно использовать любые из этих свойств в любом сочетании или все сразу. Вы можете снабдить элемент исключительно полями или же добавить границы, поля и отступы. Или элемент может иметь границы и отступ, но

без полей и т. д. Если вы не настроите какое-либо из этих свойств, то браузер применит собственные настройки, которые вам могут как понравиться, так и нет. К примеру, хотя браузеры обычно не применяют отступы или границы к элементам на странице, некоторые элементы (например, заголовки и абзацы) имеют верхнее и нижнее поле.

ПРИМЕЧАНИЕ

Поскольку разные браузеры применяют отступы и поля разного размера, лучше всего сбрасывать значения этих свойств для всех элементов. Другими словами, используйте набор простых стилей, отвечающих за сброс CSS, для удаления отступов и полей из HTML-элементов. Потом, когда вы будете создавать дополнительные стили, добавляющие поля и отступы, вы сможете быть уверены в том, что страницы будут выглядеть одинаково в разных браузерах.

Управление размерами полей и отступов

Как поля, так и отступы добавляют промежутки вокруг содержимого элементов. Свойства `margin` и `padding` используются для отделения одного элемента веб-страницы от другого. Можно применять их, например, чтобы добавить пустое пространство между панелью навигации слева и основным контентом веб-страницы справа. Возможно, вы захотите отодвинуть границу от края фотографии (рис. 7.2).

На рис. 7.2 отступ разделяет два изображения друг от друга с помощью серого фона. Вы можете устанавливать границы, поля для каждой стороны изображения независимо друг от друга. Обратите внимание, что для нижних краев (оснований) фотографий установлены бóльшие по размеру отступы, чем для остальных.

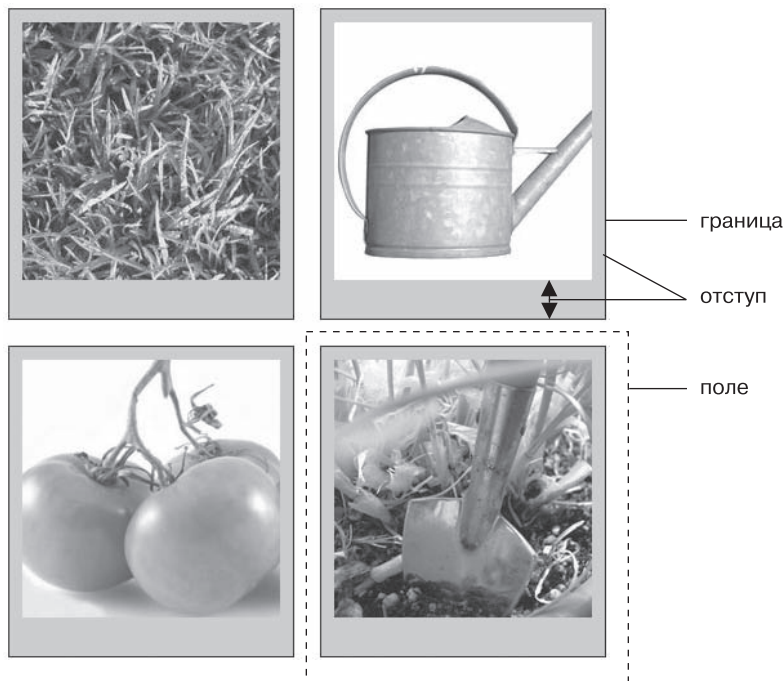


Рис. 7.2. Каждая фотография на этой веб-странице имеет поле размером 10 пикселей, то есть промежуток, отделяющий две соседние фотографии, составляет 20 пикселей

Свойства `padding` и `margin` производят одинаковый визуальный эффект, и, пока вы не добавите границу или цвет фона, вы не сможете сказать наверняка, каким свойством определен этот промежуток. Но если элемент имеет границу по периметру или цветной фон (подложку), вы заметите существенное различие этих свойств. Отступ добавляет промежуток между контентом и границей элемента и предотвращает появление эффекта заключения содержимого элемента в рамку. Он также включает область фона, поэтому пространство, занимаемое отступом, может быть свободно от содержимого (текста или фотографии), но заполнено фоновым цветом или изображением. А поля добавляют так называемые *средники* — промежутки между колонками, которые придают веб-странице более «воздушный» внешний вид.

Вы можете управлять полями или отступами каждого отдельного элемента независимо. Четыре свойства управляют соответствующими полями с каждой стороны элемента: `margin-top`, `margin-right`, `margin-bottom` и `margin-left`. Аналогично с отступами: `padding-top`, `padding-right`, `padding-bottom` и `padding-left`. Вы можете использовать любые единицы измерения, принятые в языке CSS, для определения размеров полей и отступов, например:

```
margin-right: 20px;
padding-top: 3em;
margin-left: 10%;
```

Пиксели и единицы `em` применяются и работают точно так же, как при форматировании текста (см. раздел «Изменение размера шрифта» главы 6). Поле размером 20 пикселей добавляет соответствующий пустой промежуток, отступ 3 `em` — промежуток, в три раза больший, чем размер шрифта форматируемого элемента.

Обычно используются значения в процентах. С их помощью можно гибко задавать значения границ и отступов, которые будут зависеть от ширины окна браузера, что идеально подходит для адаптивного дизайна (см. главу 15).

СОВЕТ

Чтобы удалить все пространство полей и отступов, используйте свойства со значением 0 (например, `margin-top: 0` или `padding-bottom: 0`). Чтобы убрать все дополнительное пустое пространство с четырех сторон окна браузера, нужно присвоить свойствам `margin` и `padding` нулевые значения: `margin: 0; padding 0;`. Это позволит поместить баннер, заголовок, логотип или какой-то другой элемент веб-страницы вплотную у самого края окна браузера, без промежутков.

ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Значения полей и отступов в процентах

При использовании процентов браузеры вычисляют размер полей и отступов на основе *ширины самого элемента-контейнера*, в который заключены форматируемые элементы. Рассмотрим самый простой случай, когда таким элементом-контейнером является `body`, который имеет ширину всего окна браузера. В данном случае значение в процентах в каждый конкретный момент времени вычисля-

ется на основании текущей ширины окна. Допустим, оно составляет 760 пикселей. Тогда левое поле, равное 10 %, добавит промежуток 76 пикселей с левого края форматируемого элемента. Но если вы измените размеры окна браузера, то размер промежутка левого поля тоже изменится. Уменьшение до 600 пикселей изменит размер на 60 пикселей (10 % от 600 пикселей).

Однако элемент-контейнер не всегда равен ширине окна браузера. В последующих главах книги, когда мы будем создавать более сложный дизайн веб-страниц, вы увидите, что для разработки комплексного дизайна придется добавлять дополнительные элементы.

Возможно, вы захотите добавить в веб-страницу элемент `div` для группировки содержимого навигационной панели. Допустим, она имеет ширину 300 пикселей. `div` будет контейнером для всех остальных вложенных

в него элементов. Таким образом, размер правого поля любого элемента, вложенного в `div` навигационной панели и установленного в размере 10 %, будет равен 30 пикселям.

При установке процентных значений верхнего и нижнего полей элементов ситуация еще более запутанная: эти значения вычисляются на основании ширины элемента-контейнера, а не его высоты. Таким образом, 20%-ное верхнее поле составит 20 % от ширины форматированного элемента-контейнера.

Сокращенная запись свойств `margin` и `padding`

Нередко требуется одновременно установить одинаковые размеры полей или отступов для всех четырех сторон форматированного элемента. Но последовательно набирать четыре различных свойства стиля (`margin-right`, `margin-left` и т. д.) утомительно и отнимает лишнее время. Здесь вы также можете использовать сокращенные варианты свойств `margin` и `padding` для быстрой установки всех четырех параметров одновременно:

```
margin: 0 10px 10px 20px;  
padding: 10px 5px 5px 10px;
```

ПРИМЕЧАНИЕ

Если свойству присваивается значение 0, то совсем не нужно указывать единицу измерения. Например, наберите всего лишь `margin: 0;` вместо `margin: 0px;`

Порядок определения четырех значений свойств `margin` и `padding` важен. Они должны указываться в следующей последовательности: сверху, справа, снизу и слева. Без учета этого у вас могут возникнуть проблемы с форматированием. Самый легкий способ запомнить очередность — сверху вниз по часовой стрелке.

Если вы хотите применить одинаковое значение свойства для всех четырех сторон, нет ничего проще — используйте единственное значение. Чтобы удалить все поля из заголовка `h1`, добавьте такой стиль:

```
h1 {  
  margin: 0;  
}
```

Кроме того, пользуйтесь сокращенной записью для добавления промежутков между содержимым и границами элемента:

```
padding: 10px;
```

ПРИМЕЧАНИЕ

Если нужно применить одинаковое значение свойства поля или отступа сверху и снизу элемента и одно и то же значение для левого и правого края, можно указать два значения. Так, объявление

margin: 0 2em; удаляет верхнее и нижнее поля, а левое и правое поля устанавливает равными 2 em. Точно так же, если верхние и нижние поля (или отступы) изменяются, а правые и левые остаются прежними, можно воспользоваться тремя значениями. Например, объявление margin: 0 2em 1em; установит верхнее поле равным 0, левое и правое — равными 2 em, а нижнее поле — 1 em.

Конфликты полей

В каскадных таблицах стилей не всегда справедливы математические расчеты. Вы сами можете в этом убедиться, когда нижнее поле одного элемента веб-страницы касается верхнего поля другого элемента. Вместо того чтобы объединить эти поля вместе, браузер использует большее из них (рис. 7.3, *вверху*). Предположим, значение нижнего поля маркированного списка установлено равным 30 пикселям, а значение верхнего поля следующего за ним абзаца составляет 20 пикселей. Вместо того чтобы сложить два значения, получив общий промежуток в размере 50 пикселей между списком и абзацем, браузер применяет *наибольшее* из двух значений — в данном случае 30 пикселей. Если вас такое поведение не устраивает, используйте вместо полей верхний или нижний отступ (см. рис. 7.3, *внизу*).

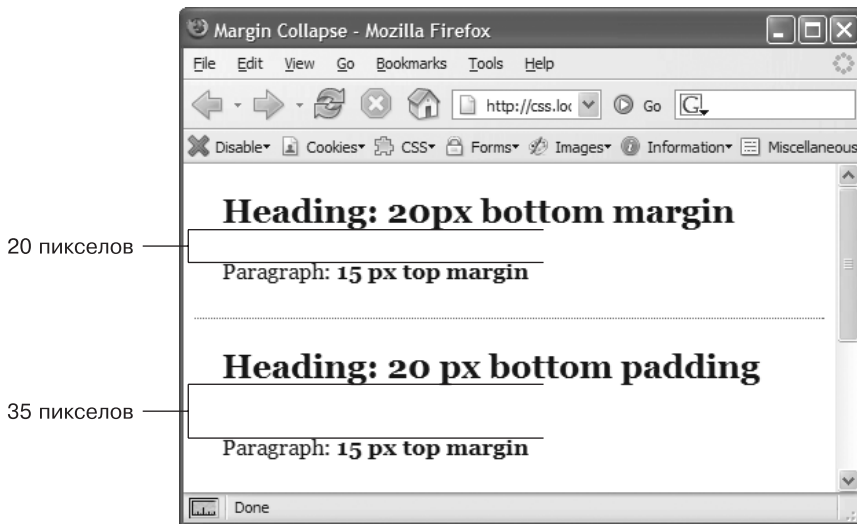


Рис. 7.3. При соприкосновении двух вертикальных полей меньшее из них игнорируется

На рис. 7.3, несмотря на то что и верхний заголовок имеет нижнее поле в размере 20 пикселей, а у расположенного ниже абзаца текста верхнее поле составляет 15 пикселей, браузер добавляет промежуток между ними, равный всего 20 пикселям. Чтобы получить тот промежуток, который вы хотели (35 пикселей), используйте вместо полей отступы, как показано в нижней части рисунка. Здесь для заголовка установлен нижний отступ 20 пикселей. Он складывается с верхним полем абзаца, равным 15 пикселям, и получается общий промежуток в размере 35 пикселей.

Ситуация еще более усугубляется, когда один элемент веб-страницы *вложен* в другой. Это может привести к затиранию отдельных частей. Допустим, вы добав-

ляете на веб-страницу «предупреждение» (заключенное в элемент `div`). Верхнее и нижнее поля устанавливаются равными 20 пикселям, чтобы отделить сообщение от заголовка сверху и от абзаца текста снизу. Пока все выглядит неплохо.

Но, предположим, вы вставляете заголовок в сообщение с предупреждением и, чтобы добавить небольшой промежуток между сообщением и верхним и нижним краем блока `div`, определяете для заголовка поле размером 10 пикселей. Вы, наверное, думаете, что добавили 10-пиксельный промежуток, но вы не правы (рис. 7.4, *вверху*). Вместо этого поле появляется *над* блоком `div`. В данном случае не имеет значения, какого размера поле применяется к заголовку, — поле все равно окажется *над* `div`.

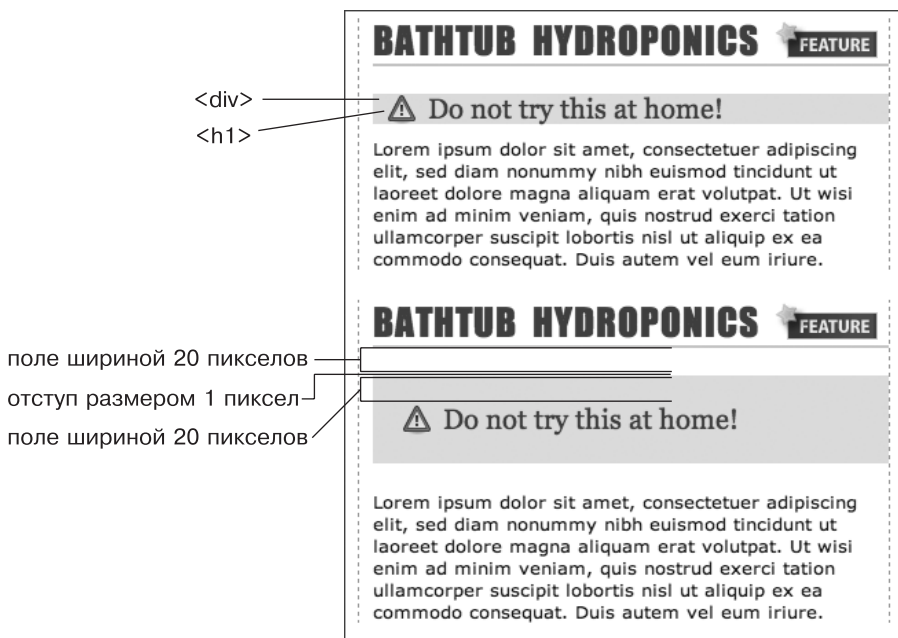


Рис. 7.4. Как бы ни соприкасались вертикальные поля, в любом случае произойдет конфликт

ПРИМЕЧАНИЕ

На профессиональном жаргоне CSS это явление называется схлопыванием полей. Оно означает, что два поля фактически превращаются в одно.

Есть два пути решения этой проблемы: добавить либо небольшой отступ, либо границу вокруг элемента `div`. Поскольку *между* этими двумя полями располагаются граница и отступ, поля больше не схлопываются и заголовок имеет небольшой отделяющий промежуток (см. рис. 7.4, *внизу*).

ПРИМЕЧАНИЕ

Горизонтальные (слева и справа) поля и поля между плавающими элементами не схлопываются. Между абсолютно и относительно позиционируемыми элементами, о которых вы узнаете в главе 13, также нет конфликта.

Удаление воздуха с помощью отрицательных значений

Большинство значений свойств в CSS должны быть положительными. Что же произойдет, если указать для размера шрифта текста отрицательное значение, например *минус 20 пикселей*? Вообще, отступы должны иметь положительные значения. Однако каскадные таблицы стилей допускают использование отрицательных значений для создания определенных визуальных эффектов.

Отрицательные значения полей вместо добавления пустого пространства между тем или иным элементом и соседними, наоборот, вызывают *удаление* этих промежутков. В результате может получиться абзац, накладывающийся на заголовок, выступающий из своего элемента-контейнера (боковой панели или другого элемента `div`) или даже совсем исчезающий за пределами окна браузера. Таким образом, можно с уверенностью сказать, что применение отрицательных значений полей дает немалую пользу.

Даже когда вы устанавливаете размеры полей, равные 0, между двумя заголовками, все равно остается небольшой промежуток (благодаря межстрочному интервалу, как описано в подразделе «Установка межстрочного интервала» раздела «Форматирование абзацев» главы 6). На самом деле это не так уж плохо, поскольку трудно читать предложения без дополнительных интервалов, сливающиеся друг с другом. Тем не менее разумное, умеренное использование небольших промежутков между заголовками поможет создать интересные визуальные эффекты. Второй заголовок на рис. 7.5 (который начинается со слов *Raise Tuna*) имеет верхнее поле размером 10 пикселей. Оно поднимает заголовок вверх, что обеспечивает небольшое наложение текста на пространство вышестоящего заголовка. Кроме того, левые и правые границы заголовка, начинающегося со слов *Extra! Extra!*, теперь соприкасаются с буквами большего заголовка, создавая эффект единой надписи.

Можно также использовать отрицательные значения полей для того, чтобы имитировать отрицательный отступ. В третьем заголовке на рис. 7.5, который начинается со слов *The Extraordinary Technique*, линия подчеркивания отображена прямо под текстом. Она на самом деле представляет собой *верхнюю* границу следующего абзаца (о том, как к тексту добавить границы, вы узнаете далее). Но, поскольку здесь определена верхняя граница с отрицательным значением, она располагается чуть выше текста абзаца и фактически находится под верхним заголовком. Обратите внимание на то, что хвост буквы *Q* заголовка буквально свисает под линией-границей. Поскольку отступ между содержимым (то есть *Q*) и границей не может быть отрицательным, вам не удастся поднять нижнюю границу так, чтобы она находилась ближе к тексту или любому другому содержанию, не говоря уже о наложении. И все же есть возможность добиться этого эффекта, применив границу с отрицательным значением к последующему, нижестоящему элементу веб-страницы.

СОВЕТ

Можно использовать либо верхнее поле абзаца с отрицательным значением, либо отрицательное нижнее поле заголовка. Оба варианта приведут к одному и тому же визуальному эффекту поднятия абзаца выше, ближе к тексту заголовка.



Рис. 7.5. Чтобы превратить верхнюю границу последнего абзаца в нижнюю границу вышестоящего заголовка, вернее, имитировать такой эффект, добавим небольшой отступ. Верхний отступ шириной 5 пикселей сместит абзац вниз относительно границы, в то время как левый отступ шириной 4 ем сместит текст абзаца, оставив верхнюю границу у левого края

Отображение строчных и блочных элементов

Хотя браузеры и обрабатывают любой тег веб-страницы подобно блочному элементу, на самом деле они не все одинаковы. В CSS существует два различных типа элементов: *блочные* и *строчные*.

В *блочных* элементах создается разрыв строки перед элементом и после него. Например, абзац `p` создает блок, отделенный от элементов, расположенных выше и ниже его. Другими примерами являются заголовки, контейнеры `div`, таблицы, списки и элементы списков.

Строчные элементы не создают разрывов строк ни до, ни после себя. Они отображаются на одной строке с содержимым рядом стоящих элементов. Элемент `strong` — строчный. Слово, отформатированное с его помощью, будет расположено на одной строке с текстом, заключенным в другие строчные элементы, например `em`. Было бы очень странно, если бы отдельное слово в середине абзаца, выделенное `strong`, вдруг появилось на отдельной строке. Другими примерами строчных элементов являются `img` — для добавления изображений, `a` — для создания ссылок, различные элементы для разметки форм и т. д.

В большинстве случаев каскадные таблицы стилей одинаково работают со строчными и блочными элементами. Можно применять шрифты, цвет, фон, границы к обоим типам элементов. Однако поля и отступы строчных элементов браузеры обрабатывают по-другому. Если добавить поля или отступы слева или справа строчного элемента, то посредством установки верхнего или нижнего отступа или поля увеличить высоту строчного элемента не удастся. В верхнем абзаце на рис. 7.6 строчный элемент отформатирован с применением границ, фонового цвета и полей размером 20 пикселей со всех четырех сторон. Но браузер добавляет пустые промежутки только с левой и правой стороны элемента.

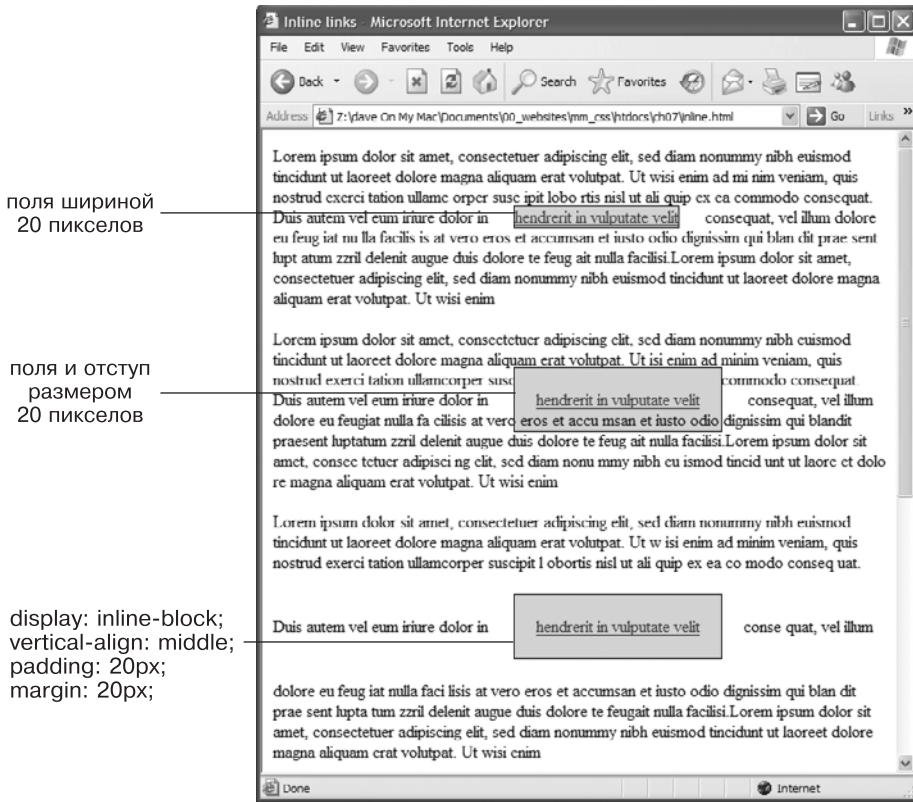


Рис. 7.6. Добавление к строчному элементу верхнего, нижнего поля или отступа не изменит высоту элемента: форматирование будет не таким, как вы ожидаете.

На рис. 7.6 в среднем абзаце фон и границы ссылки накладываются на текст, находящийся выше и ниже формируемого. Цвет фона строчного элемента отображается поверх вышестоящей строки текста, но под следующей, так как браузер обрабатывает каждую строку так, будто она расположена в стеке, наверху по отношению к предыдущей. Как правило, это не представляет проблемы, поскольку строки текста обычно не накладываются. Если вы хотите, чтобы верхние и нижние поля были применимы к строчному элементу, используйте инструкцию `display: inline-block` (внизу). Она оставит элемент строчным, но он будет восприниматься как блочный, поэтому отступы, поля, границы, ширина и высота будут к нему применимы.

ПРИМЕЧАНИЕ

Есть одно исключение из этого правила: если поля или отступы применяются к строчным элементам `img`, элементы не изменяют своей высоты. Браузеры корректно меняют высоту контейнера элемента-изображения, чтобы подогнать указанные отступы и поля.

Иногда требуется, чтобы строчные элементы вели себя так же, как блочные, или наоборот. Маркированные списки представляют элементы в виде отдельных блоков, то есть каждый элемент списка располагается в стеке поверх следующего. Но что

делать, если вы хотите изменить поведение пунктов списка таким образом, чтобы все они располагались рядом друг с другом, на одной строке (как панель навигации, показанная на рис. 9.4)? Или, возможно, вы захотите, чтобы строчный элемент обрабатывался как блочный, например, изображение, встроенное в абзац текста, было расположено на отдельной строке, с верхним и нижним промежутками-интервалами.

В языке CSS есть команда, которая позволяет вам это сделать, — это свойство `display`. С его помощью можно заставить блочный элемент функционировать как строчный:

```
display: inline;
```

Или, наоборот, вы можете сделать так, чтобы строчные элементы, например изображение или ссылка, вели себя как блочные:

```
display: block;
```

Наконец, вы можете заставить элемент действовать и как блочный, и как строчный. Значение `inline-block` не создает разрывов ни до, ни после элемента и одновременно заставляет элемент подчиняться верхним и нижним полям и отступам, а также настройкам высоты:

```
display: inline-block;
```

ПРИМЕЧАНИЕ

У свойства `display` есть большое количество значений, многие из которых поддерживаются не всеми браузерами. Значение `inline-block` поддерживается во всех современных браузерах (см. рис. 7.6). Другое значение — `none` — обрабатывается в большинстве браузеров и имеет множество вариантов использования. Это значение выполняет одну простую функцию — полностью скрывает форматированный элемент, чтобы он не отображался в окне браузера. Используя JavaScript-сценарии, вы можете скрыть элементы, чтобы они стали видимыми после присваивания свойству `display` значения `inline` или `block`. Сделать их видимыми можно и средствами каскадных таблиц стилей.

Добавление границ

Граница представляет собой обычную линию, которая очерчивает форматированный элемент. Как показано на рис. 7.1, она располагается между отступом и полем элемента. С помощью границ, добавленных со всех сторон, можно заключить изображение в рамку или выделить баннер и т. д. Но совсем не обязательно применять границы, создающие очертания всего содержимого элемента. Точно так же, как вы добавляете границы с четырех сторон элемента, можно добавить требуемую границу с любой из сторон по отдельности. Эта гибкость обеспечивает добавление произвольных элементов дизайна. Например, добавьте границу слева от элемента, придайте ей толщину около 1 em, и она станет похожа на маркер в виде квадрата. Единственная граница с нижнего края абзаца производит тот же визуальный эффект, что и элемент `hr` (горизонтальная линия), выступая разделителем контента веб-страницы.

Вы можете управлять тремя различными свойствами любой из границ: `color` (цвет), `width` (ширина) и `style` (стиль). Значение `color` может быть представлено шестнадцатеричным числом, ключевым словом или значением в системе RGB (или RGBA). Ширина границы `width` — толщина линии, используемой для очерчивания.

Вы можете указывать любые единицы измерения каскадных таблиц стилей (кроме процентов) или ключевые слова `thin` (тонкая линия), `medium` (средняя) и `thick` (толстая). Самые распространенные и понятные единицы измерения для данного свойства — пиксели.

И наконец, свойство `style` управляет типом линии границы. Существует множество различных стилей. Примеры приведены на рис. 7.7. Вы можете также определить стиль с помощью ключевых слов. Например, значение `solid` рисует сплошную линию, а `dashed` — штриховую (пунктирную). В каскадных таблицах стилей для границ имеются следующие стили: `solid`, `dotted`, `dashed`, `double`, `groove`, `ridge`, `inset`, `outset`, `none` и `hidden`. Ключевые слова `none` и `hidden` работают одинаково: они полностью удаляют границы. Но значение `none` удобно использовать для удаления границы с одной стороны элемента.

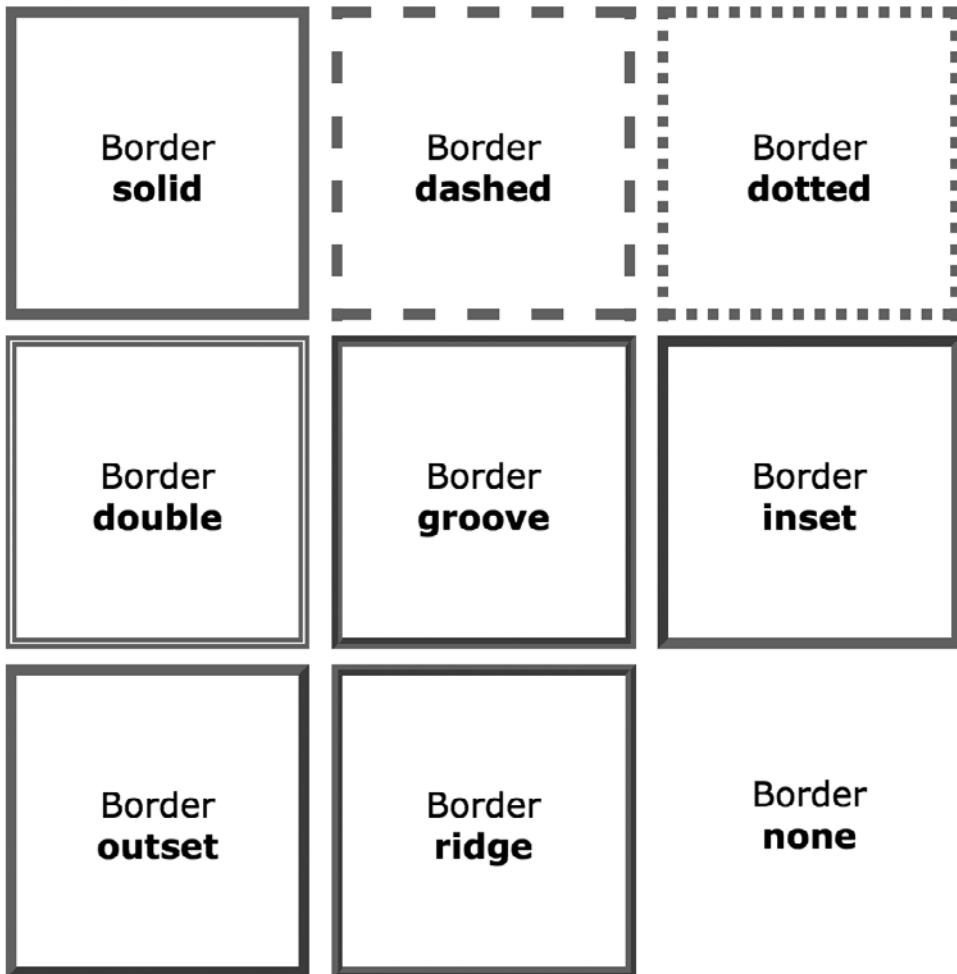


Рис. 7.7. Каскадные таблицы стилей предлагают девять различных стилей границ

Самая простая граница — сплошная (*solid*). Но если вы хотите добиться особого визуального эффекта, то можете сделать границу пунктирной или точечной, имитировать трехмерную рамку с помощью значений *groove*, *ridge*, *inset* и *outset*. Вы даже можете применить различные стили границ к разным сторонам элемента, например сделав верхнюю границу сплошной, а остальные три — пунктирными.

Сокращенная запись свойства `border`

Если вы взглянете на полный список свойств, доступных в языке CSS, то подумаете, что границы действительно сложны. Вообще, существует 20 разновидностей свойств границ, с которыми вы столкнетесь в следующих разделах книги, а также несколько относящихся к таблицам. Но все это — лишь варианты, обеспечивающие различные способы управления одними и теми же тремя свойствами: цвета, ширины (толщины) и стиля для каждой из четырех границ. Наиболее простое и понятное свойство — `border`, которое добавляет границы с заданными параметрами:

```
border: 4px solid rgb(255,0,0);
```

Стиль создает сплошную красную границу с толщиной линии 4 пиксела. Вы можете использовать это свойство для создания простейшей рамки, окаймляющей изображение, панель навигации или любой другой элемент, которые надо выделить в отдельный блок.

ПРИМЕЧАНИЕ

Последовательность указания свойств не имеет значения: `border: 4px solid rgb(255,0,0);` работает так же, как `border: rgb(255,0,0) solid 4px;`.

Форматирование границ по отдельности

Вы можете управлять границей с каждой стороны элемента отдельно, используя соответствующее свойство: `border-top`, `border-bottom`, `border-left` или `border-right`. Они работают точно так же, как стандартное свойство `border`, с тем исключением, что управляют границей только с одной стороны формируемого элемента. Добавить красную пунктирную линию снизу можно, используя следующее объявление свойства:

```
border-bottom: 2px dashed red;
```

Вы можете объединять общее свойство `border` со свойствами отдельных границ. Например, использовать свойство `border-left`, чтобы определить основной, общий стиль, а затем выборочно настроить одну или несколько границ. Допустим, вы хотите, чтобы верхняя, левая и правая стороны абзаца имели одинаковый тип границы, а нижняя выглядела по-другому. Можно написать такие четыре строки кода:

```
border-top: 2px solid black;  
border-left: 2px solid black;  
border-right: 2px solid black;  
border-bottom: 4px dashed #333;
```

Такого же эффекта можно достичь всего двумя строками кода:

```
border: 2px solid black;  
border-bottom: 4px dashed #333;
```

Первая строка определяет общий вид всех четырех границ, а вторая переопределяет вид нижней границы. Преимущество не только в том, что легче написать две строки кода вместо четырех, но и в том, что изменить стиль будет проще. Если вы захотите сделать цвет верхней, левой и правой границ красным, то необходимо отредактировать единственную строку кода вместо трех:

```
border: 2px solid red;  
border-bottom: 4px dashed #333;
```

При использовании этого сокращенного способа установки границ определяется общий вид всех четырех границ. Затем вид одной из границ замещается с помощью свойства конкретной границы, например `border-left`. Очень важно, чтобы свойства были написаны в определенной последовательности. В общем случае глобальные установки границ должны быть на первом месте, а установки отдельной границы — на втором:

```
border: 2px solid black;  
border-bottom: 4px dashed #333;
```

Поскольку свойство нижней границы `border-bottom` указано вторым, оно частично замещает общие установки свойства `border`. Если бы свойство `border-bottom` было расположено первым, то `border` было бы отменено и все четыре границы стали бы одинаковыми. Последнее явное свойство может заместить любые аналогичные свойства, определенные в коде выше. Это пример работы механизма каскадности CSS, который мы рассматривали в главе 5.

Вы также можете использовать этот сокращенный способ определения границ, чтобы скрыть их с помощью ключевого слова `none`. Предположим, вы хотите установить границы только с трех сторон элемента (сверху, слева, снизу). Следующие две строки кода обеспечат такое форматирование:

```
border: 2px inset #FFCC33;  
border-right: none;
```

Возможность тонкой настройки границ каждой из сторон формируемого элемента является причиной большого количества разновидностей свойств границ. Остальные 15 свойств позволяют определять индивидуальные цвета, стили, толщину линий границ для каждой стороны. Например, можно переписать определение `border: 2px double #FFCC33` следующим образом:

```
border-width: 2px;  
border-style: double;  
border-color: #FFCC33;
```

В этом варианте используются три строки кода вместо одной, поэтому вы, наверное, будете избегать такого способа. Однако каждая сторона имеет свой набор из трех свойств, которые удобно использовать для отмены одного. Правая граница: `border-right-width`, `border-right-style` и `border-right-color`. Левая, верхняя и нижняя границы имеют похожие свойства: `border-left-width`, `border-left-style` и т. д.

Вы можете изменить ширину единственной стороны границы так: `border-right-width: 4px;`. При таком подходе хорошо то, что, когда вы позже решите изменить границу на сплошную, нужно будет отредактировать только групповое свойство границы, изменив значение `dashed` на `solid`.

Кроме того, вы можете задать собственные значения для каждой стороны границы, используя свойства `border-width`, `border-style` и `border-color`. Например, правило `border-width: 10px 5px 15px 13px;` применит четыре различных значения ширины для каждой из сторон (верхней, правой, нижней и левой).

Допустим, вы хотите установить все четыре границы элемента в виде пунктирной линии толщиной 2 пиксела, но при этом нужно, чтобы каждая граница имела свой цвет (например, вы создаете сайт для детей). Ниже представлен способ быстро сделать это:

```
border: 2px dashed;  
border-color: green yellow red blue;
```

Этот набор правил создает границы в виде двухпиксельной пунктирной линии со всех четырех сторон элемента, при этом верхняя граница будет иметь зеленый цвет, правая — желтый, нижняя — красный, а левая — синий.

ПРИМЕЧАНИЕ

Как правило, при использовании границ требуется добавлять отступы. Они обеспечивают промежутки между границами и содержимым элементов: текстом, изображениями, прочими элементами. Обычно границы отображаются слишком близко к содержимому элементов, только если вы не захотите разместить их вокруг изображения.

Установка цвета фона

В языке CSS имеются средства для добавления фона как для всей веб-страницы, так и для отдельного заголовка или любого другого элемента страницы. Используйте свойство `background-color` в сочетании с любым из действительных определений цветов, которые описаны в разделе «Форматирование текста цветом» главы 6. При желании вы можете окрасить фон веб-страницы в яркий зеленый цвет, указав следующий код:

```
body { background-color: rgb(109,218,63); }
```

Или можете создать класс, например, `.review` со свойством `background-color` и значением желаемого цвета фона, а затем применить его к элементу `body` в HTML-коде таким образом: `<body class="review">`.

ПРИМЕЧАНИЕ

Вы можете также поместить изображение на заднем плане в качестве фона веб-страницы и управлять его положением различными способами. Мы рассмотрим это в следующей главе. Кроме того, к фону любого элемента можно добавить цветовой градиент, о котором вы узнаете в следующей главе.

Цвет фона удобно применять для создания множества различных визуальных эффектов. Вы можете придать заголовку контрастность, рельефность, установив темный цвет для фона и светлый — для текста. Цвет фона также является отличным

средством для выделения таких обособленных частей веб-страницы, как панель навигации, баннер или боковая панель.

И не забывайте о ранее рассмотренном методе задания цвета по модели RGBA. С его помощью можно сделать фон полупрозрачным, позволяя просматриваться находящимся позади него цвету, текстурам или изображениям. Например, можно сделать желто-коричневый фон страницы. Затем, предположим, возникло желание придать находящемуся внутри элементу `div` более светлый оттенок желто-коричневого цвета. Вместо задания для фона элемента `div` сплошного цвета можно добавить белый цвет, а затем управлять прозрачностью этого цвета, чтобы через него просвечивались различные оттенки желто-коричневого цвета:

```
body {  
  background-color: rgb(247,226,155);  
}  
.special-div {  
  background-color: rgba(255,255,255,.75);  
}
```

ПРИМЕЧАНИЕ

Когда вы пользуетесь одновременно фоновым цветом и границами, помните: если стиль границы — точечная или пунктирная линия (см. рис. 7.7), то фоновый цвет отображается в промежутках между точками или штрихами линий границ. Другими словами, браузеры размещают линию границ поверх цвета фона. Тем не менее вы можете обойти это поведение, присвоив свойству `background-clip` значение `padding-box`:

```
background-clip: padding-box;
```

Скругление углов

Как уже ранее упоминалось, браузеры рассматривают все элементы как строго прямоугольные блоки. Это становится очевидным при установке границы вокруг абзаца или элемента `div`. На этот случай есть возможность сгладить острые углы таких прямоугольников, добавив к стилям скругленные углы (рис. 7.8). В языке CSS представлено свойство `border-radius`, позволяющее дизайнерам добавлять скругления к одному или нескольким углам элемента. В самом простом варианте свойство `border-radius` получает одно значение, которое затем применяется ко всем четырем углам элемента:

```
.specialBox {  
  background-color: red;  
  border-radius: 20px;  
}
```

Браузер использует предоставленное значение радиуса для выписывания круговой траектории в каждом углу элемента. Как показано на рис. 7.9, значение, равное расстоянию от центра окружности до ее края, является ее радиусом. В качестве единиц измерения чаще всего применяются пикселы и `em`, но можно использовать и проценты (хотя они ведут себя немного неожиданно).



Рис. 7.8. В языке CSS позволяет скруглять углы элемента. Чтобы увидеть любое из этих удивительных закруглений, у элемента должны быть цветной фон или граница

При использовании единственного значения браузер рисует закругления одинакового радиуса для каждого угла элемента. Например, для верхнего левого изображения на рис. 7.8 используется следующее объявление:

```
border-radius: 30px;
```

Но указанием одного и того же значения для каждого угла настройки не ограничиваются. Для каждого угла можно предоставить отдельные значения, задав четыре параметра. Например, у верхнего правого блока на рис. 7.8 четыре разных угла. Объявление имеет следующий вид:

```
border-radius: 0 30px 10px 5px;
```

Сначала задается числовое значение для левого верхнего угла блока, а затем по часовой стрелке для всех остальных углов. То есть первое значение (0 в примере на рис. 7.8) применяется к левому верхнему, второе (30px) — к правому верхнему, третье (10px) — к правому нижнему и четвертое (5px) — к левому нижнему углу. Можно задать только два значения, тогда первое число будет применено к левому верхнему и правому нижнему углам, а второе — к правому верхнему и левому нижнему углам.

Кроме рассматриваемых до сих пор абсолютно круглых углов (то есть представляющих собой часть окружности), можно задавать эллиптические углы, подобные тем, что показаны в двух нижних примерах на рис. 7.8. Для эллиптической формы угла требуется два значения радиуса: первое для радиуса от центра до левого или правого края, а второе — для определения расстояния от центра до верхнего или нижнего края. Например, чтобы добавить углы, показанные в левом нижнем углу на рис. 7.8, нужно создать следующее объявление:

```
border-radius: 40px/20px;
```

Значение 40px задает горизонтальный радиус, а значение 20px — вертикальный. Слеш между ними оповещает браузер о создании эллиптического угла. Можно сделать так, что у каждого из четырех углов будут разные вытянутые формы, указав различные значения каждого из радиусов эллипса. Это может показаться немного странным:

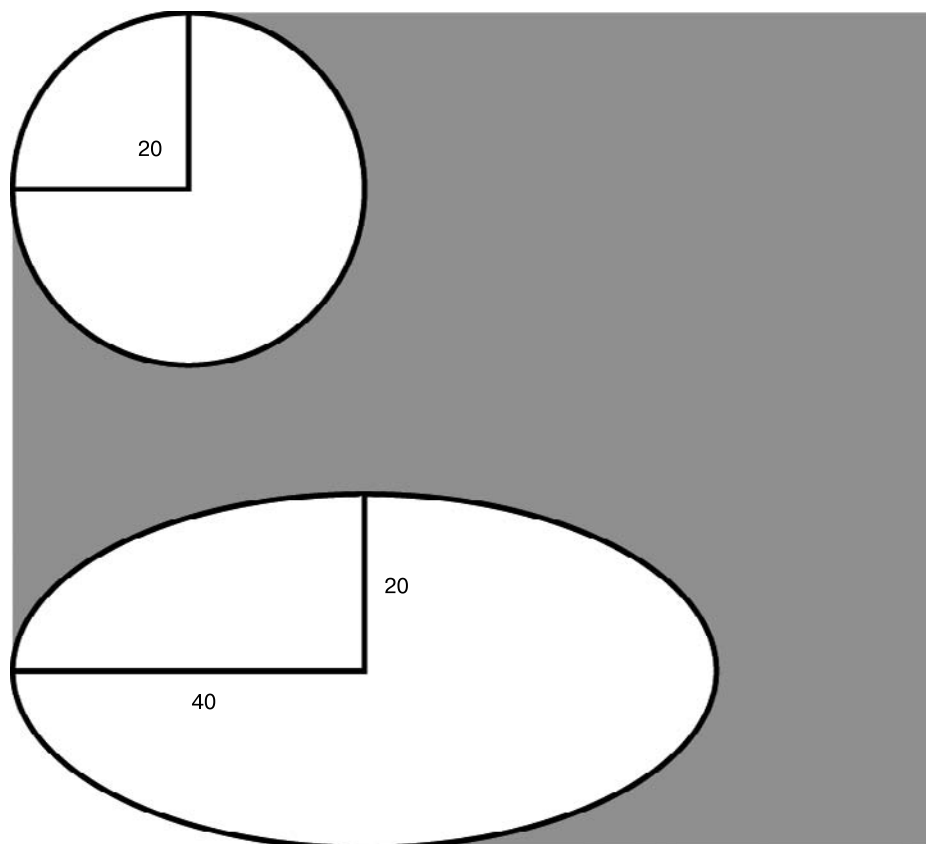


Рис. 7.9. Можно создавать не только скругленные (*вверху*), но и эллиптические углы (*внизу*)

сначала указываются четыре значения для горизонтальных радиусов каждого из углов (начиная с левого верхнего угла), а затем после каждого из значений указываются обратный слеш (/) и четыре значения вертикального радиуса каждого угла:

```
border-radius: 40px 10px 20px 10px / 20px 30px 40px 20px;
```

С помощью аналогичного синтаксиса можно даже смешивать эллиптические и скругленные углы. Горизонтальный и вертикальный радиусы у закругленных углов одинаковые:

```
border-radius: 10px 10px 20px 10px / 10px 30px 40px 10px;
```

И наконец, если нужно пойти по более длинному пути, то для определения формы каждого угла можно воспользоваться отдельными свойствами. Например, правило:

```
border-radius: 1em 2em 1.5em .75em;
```

можно также записать следующим образом:

```
border-top-left-radius: 1em;
border-top-right-radius: 2em;
```



```
border-bottom-right-radius: 1.5em;  
border-bottom-left-radius: .75em;
```

ПРИМЕЧАНИЕ

Браузер Internet Explorer 8 и более ранние версии не поддерживают свойство `border-radius`, поэтому при его объявлении будут показаны прямые углы.

Добавление тени

Как уже упоминалось, чтобы сделать текст объемным, к нему можно добавить еле заметную (или весьма заметную) отбрасываемую тень. Для добавления теней к блоку, обрамляющему элемент, используется свойство `box-shadow`, позволяющее, к примеру, блоку `div` казаться парящим над страницей (рис. 7.10). По сравнению с `text-shadow`, у этого свойства есть несколько дополнительных настроек. Например, можно сделать так, чтобы тень появлялась *внутри* блока, как показано в нижней части рис. 7.10.

Свойство `box-shadow` работает в большинстве современных браузеров, включая Internet Explorer 9. При этом IE 9 рисует тени заметно тоньше, чем другие браузеры. Кроме того, браузер IE 8 и более ранние версии не поддерживают это свойство и не отобразят тени у элементов.

Основной синтаксис свойства `box-shadow` показан на рис. 7.11. Первое значение задает горизонтальное смещение, то есть это значение приводит к перемещению тени влево или вправо от элемента. Положительное число приводит к перемещению тени вправо (верхний блок на рис. 7.10), а отрицательное число — влево.

Второе значение задает вертикальное смещение — позицию тени либо над элементом, либо под ним. При положительном значении тень помещается ниже нижнего края блока (верхний блок на рис. 7.10), а при отрицательном значении тень помещается над верхним краем блока.

ПРИМЕЧАНИЕ

Для задания значений тени нужно использовать пиксели или единицы `em`. Проценты не поддерживаются.

Третье значение задает радиус размытия тени. Оно определяет степень размытости и ширины тени. Значение 0 не придает эффекта размытости, то есть получается тень с четкими краями. Чем выше значение, тем более размытой и тусклой становится тень.

И наконец, последнее значение задает цвет отбрасываемой тени. Можно воспользоваться любым обозначением цвета, принятым в языке CSS, но RGBA-значения смотрятся намного лучше, потому что позволяют управлять прозрачностью тени, делая ее более реалистичной.

Для свойства `box-shadow` доступны два дополнительных значения: ключевое слово `inset` и значение расширения. Ключевое слово `inset` помещает тень внутрь блока (см. рис. 7.10, *внизу*). Для создания внутренней тени нужно добавить слово `inset` в качестве первого значения свойства `box-shadow`:

```
box-shadow: inset 4px 4px 8px rgba(0,0,0,.75);
```

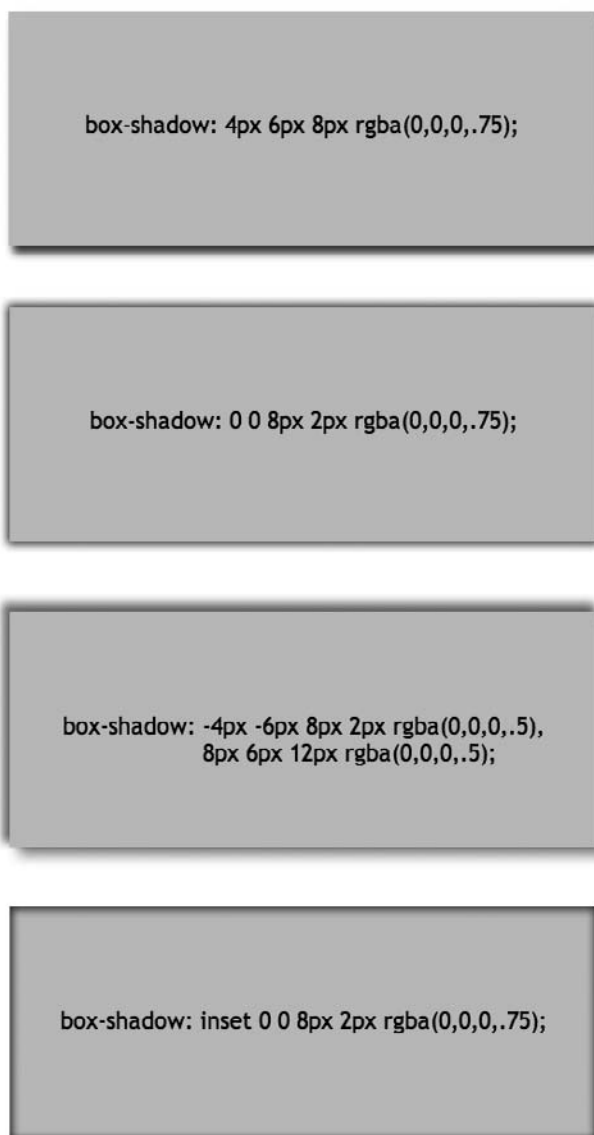


Рис. 7.10. Свойство `box-shadow` позволяет добавлять тени к элементам, создавая эффект парения над страницей

В качестве четвертого значения (между радиусом размытия тени и ее цветом) можно также добавить *расширение*. Это приведет к расширению тени на указанное значение. Иначе говоря, если добавить значение расширения, равное `10px`, браузер расширит тень на 10 пикселей в каждом направлении (в целом тень получается на 20 пикселей шире и на 20 пикселей выше). Это значение также задает позицию, в которой применяется радиус размытия. То есть, если добавлено расширение,

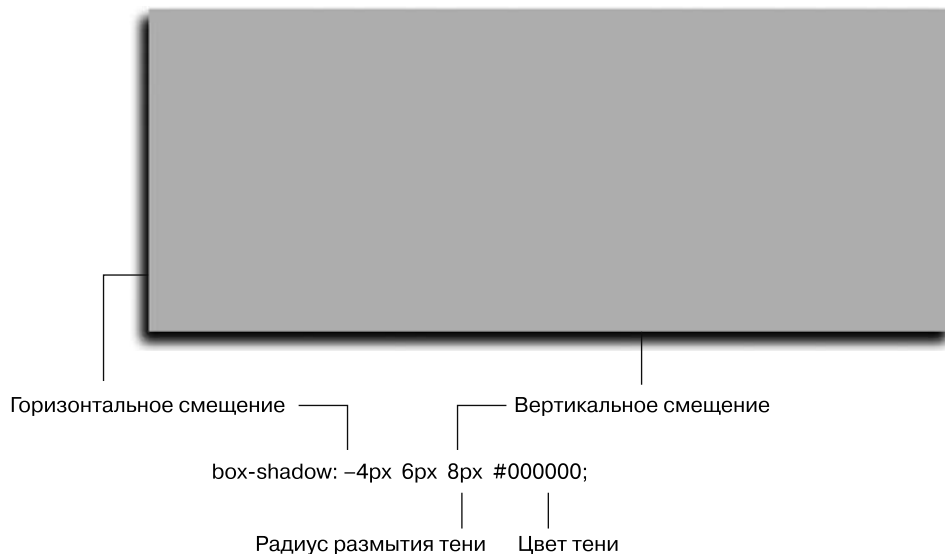


Рис. 7.11. Чаще всего тень размещается либо слева (при отрицательном значении, как показано на рисунке) элемента, либо справа (при положительном значении), либо выше верхнего края (при отрицательном значении), либо ниже нижнего края (при положительном значении) с ее размытием согласно указанному радиусу

размытие тени не начинается до тех пор, пока не будет применено значение расширения. В частности, это пригодится в том случае, когда нужно добавить тень вокруг всего элемента, добиваясь эффекта *свечения*.

Например, во втором сверху блоке на рис. 7.10 горизонтальное и вертикальное смещение равны 0, для радиуса тени установлено значение 8px, а для расширения — 2px. Значение расширения выталкивает тень на 2 пиксела наружу относительно всех четырех сторон блока, а затем радиус, равный 8 пикселям, расширяет размытие еще на 8 пикселей. Значение расширения можно даже использовать для создания вокруг существующей границы второй, по-другому расцветченной тени. Вот как выглядит пример кода для этого эффекта:

```
border: 10px solid rgb(100,255,30);
box-shadow: 0 0 0 10px rgb(0,33,255);
```

И наконец, можно даже применить к элементу несколько теней (третий блок на рис. 7.10). Для этого нужно указать запятую после первого набора установок тени, а затем добавить еще одну тень:

```
box-shadow: 10px 5px 8px #FF00FF,
-5px -10px 20px 5px rgb(0,33,255);
```

Количество теней не ограничено (только здравым смыслом).

ПРИМЕЧАНИЕ

Эффект тени заставляет браузер выполнять дополнительные вычисления. Используйте тень аккуратно. Кроме того, не забудьте проверить работу страницы с эффектом тени на мобильных устройствах, вычислительные ресурсы которых гораздо ниже, чем у настольных компьютеров и ноутбуков.

Изменение высоты и ширины

Рассмотрим еще два свойства, являющихся частью блочной модели CSS. Они предназначены для установки размеров объектов, таких как таблица, колонка, баннер, боковая панель. Свойства `height` и `width` определяют высоту и ширину области форматированного элемента. Мы будем часто пользоваться ими при создании разметки, макета веб-страниц, как описано в части III. Они также применяются для разработки базового дизайна: назначения ширины таблиц, создания простейших боковых панелей или галерей эскизов.

Разработка стилей с этими свойствами не составляет сложностей. Наберите их со значением в любой единице измерения языка CSS, которые мы изучили. Например:

```
width: 300px;  
width: 30%;  
height: 20em;
```

Пиксели просты в использовании, понятны и удобны, обеспечивают точные ширину или высоту. Единица измерения *em* — это примерно то же самое, что и размер шрифта текста. Допустим, вы устанавливаете размер шрифта 24 пиксела; единица *em* для этого форматированного элемента будет равна 24 пикселям, а если вы установите ширину равной 2 *em*, она составит 2×24 или 48 пикселей. Если вы не определите в стиле размер шрифта текста, то он будет взят из унаследованных параметров.

Процентные значения свойства `width` рассчитываются на основании ширины элемента-контейнера. Если вы установите ширину заголовка равной 75 % и этот заголовок не вложен ни в какие другие элементы веб-страницы с явно определенной шириной, то ширина заголовка составит 75 % от ширины окна браузера. Если посетитель изменит размер окна браузера, то ширина заголовка тоже изменится. Однако если заголовок заключен в блок `div` шириной 200 пикселей (к примеру, для создания колонки), то ширина этого заголовка составит 150 пикселей. Процентные значения в свойстве `height` работают точно так же, но расчет базируется на высоте элемента-контейнера, а не на его ширине.

Вычисление фактических размеров блочных элементов

Свойства `width` и `height` на первый взгляд кажутся довольно простыми и понятными, однако есть несколько нюансов, вводящих начинающих веб-дизайнеров в заблуждение. Прежде всего, существует различие между значениями ширины и высоты, которые вы явно указываете при написании стилей, и размером пространства, которое браузер фактически выделяет и использует для отображения элементов блочной модели CSS. Свойства `width` и `height` устанавливают ширину и высоту *области содержимого* форматированного элемента — пространства, в котором заключены текст, изображения или другие вложенные элементы (см. рис. 7.1, чтобы вспомнить о том, где именно в блочной конструкции элементов CSS находится область содержимого).

Фактическая ширина элемента веб-страницы представляет собой область экрана (окна браузера), выделяемую для отображения. Она состоит из ширины полей, границ, отступов и явно указанного значения ширины в свойстве стиля, как показано на рис. 7.12.

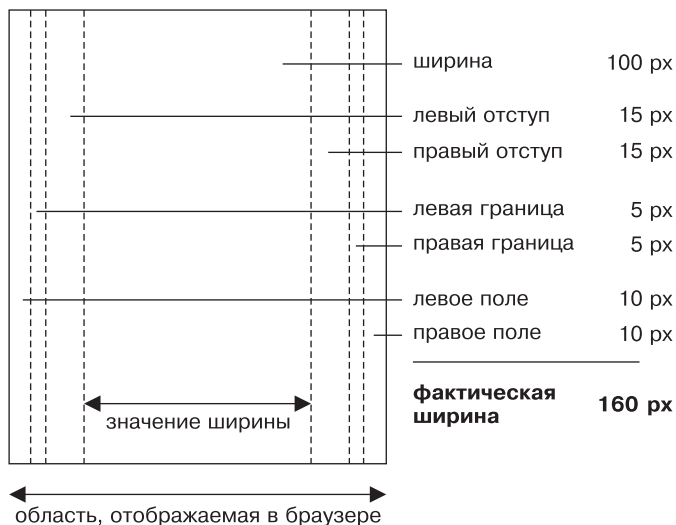


Рис. 7.12. Фактическая ширина блока формируемого элемента вычисляется путем сложения значений свойств `margin`, `border`, `padding` и `width`. Высота экранного элемента рассчитывается исходя из суммы значения свойства `height`, верхних и нижних полей, верхней и нижней границ и верхнего и нижнего отступа

Допустим, вы назначили следующие свойства:

```
width: 100px;
padding: 15px;
border-width: 5px;
margin: 10px;
```

Если определено свойство `width`, вы всегда точно знаете, сколько места займет содержимое элемента — текст и изображения, заполняющие основное пространство элемента, — независимо от любых других установленных свойств. Вам не нужно ничего вычислять, потому что значение `width` и есть размер основного пространства для размещения содержимого (в представленном выше примере ширина равна 100 пикселям). Конечно, *придется* выполнить несложные арифметические операции, чтобы выяснить общий точный размер. В представленном выше примере для размещения формируемого элемента отводится пространство шириной 160 пикселей: 20 пикселей для левого и правого полей, 10 — для левой и правой границ, 30 — для левого и правого отступа и 100 пикселей в качестве ширины основного содержимого.

Общее правило по регулированию высоты элементов на странице гласит так: *не делайте* этого! Многие подающие надежды веб-дизайнеры пытаются задать высоту абсолютно для всего, желая получить полный контроль над пикселями.

Но если вы полностью не уверены в точных размерах содержимого внутри элемента, то можете столкнуться с некоторыми нежелательными результатами (рис. 7.13). В этом примере блок с цитатой, который используется для того, чтобы акцентировать внимание на интересном отрывке из статьи, имеет ширину и высоту 100 пикселей. Когда в блок добавляется больше текста, чем может уместиться в такую высоту, его содержимое выходит за пределы. Даже если вы уверены, что текст, который вы разместили в блоке с фиксированной высотой, соответствует его размерам, посетитель может увеличить размер шрифта в своем браузере и высота текста, соответственно, станет больше высоты блока.

Другими словами, свойство `height` полезно для контроля высоты элемента `div`, содержащего, например, изображения, потому что в таком случае вы можете правильно определить его высоту. Однако если вы используете это свойство для элементов с текстом, не забудьте не только протестировать свои страницы в основных браузерах, но и проверить их при различных установленных размерах шрифта, изменяя его в браузерах.



Рис. 7.13. Когда устанавливается высота элемента (например, высота контейнера `div` правой боковой панели, как в данном примере), содержимое которого выше самого элемента, браузеры позволяют содержимому выйти за нижнюю границу элемента

СОВЕТ

Область баннера на странице — еще один подходящий кандидат для установки высоты. Обычно баннер имеет ограниченное содержимое: логотип, поле поиска, может быть, какие-нибудь навигационные кнопки. Зачастую у баннеров остается довольно много пустого пространства (помогающего привлечь внимание посетителя к ключевым элементам навигационной панели), поэтому указание высоты для баннера обычно не вызывает проблем.

Переопределение ширины блока с помощью свойства `box-sizing`

Как уже упоминалось, браузеры традиционно вычисляют ширину элемента, складывая значения свойств `border`, `padding` и `width`. Это не только заставит вас проводить математические вычисления, чтобы определить реальную ширину отображаемого элемента, но и может вызвать ряд других проблем. Особенно это касается случаев создания макетов с плавающими элементами с использованием процентных отношений. Подробности создания макетов с плавающими элементами нам еще предстоит изучить, но если говорить вкратце, каскадные таблицы стилей позволяют с помощью свойства `float` помещать элементы бок о бок, что дает возможность создавать разметки, состоящие из нескольких колонок.

Когда для нескольких колонок используется процентное отношение, могут возникать довольно странные проблемы. Предположим, есть две колонки (на самом деле два таких элемента, как `div`) и нужно, чтобы каждая занимала 50 % ширины окна. Соответственно, для двух колонок устанавливается ширина 50 %, но на тот момент, когда добавляется отступ или граница к одной из колонок, вы увеличиваете ее ширину, которая становится больше 50 % (если точнее, она составляет 50 % плюс значение левого и правого отступов и значение ширины левой и правой границ). В большинстве случаев это заставит вторую колонку опуститься *ниже* первой.

На этот случай каскадные таблицы стилей предлагают свойство, позволяющее изменить порядок вычисления браузером экранной ширины (и высоты) элемента. Свойство `box-sizing` предоставляет три варианта.

- `content-box` устанавливает ранее рассмотренный способ, с помощью которого браузеры всегда определяют экранную ширину и высоту элемента. То есть браузер добавляет ширину границ и толщину отступа к значениям, установленным для свойств ширины и высоты, чтобы определить экранную ширину и высоту заданного элемента. Поскольку это поведение используется по умолчанию, указывать что-либо для реализации варианта `content-box` не нужно.
- `padding-box` сообщает браузеру, что при установке для стиля свойства ширины или высоты они должны включать отступы как часть своего значения. Предположим, что есть элемент с отступами слева и справа, равными 20 пикселям. Его ширина установлена равной 100 пикселям. Браузер будет рассматривать ту часть, что приходится на отступы, частью этого 100-пиксельного значения. То есть ширина области содержимого будет равна всего лишь 60 пикселям ($100 - 20$ [левый отступ] $- 20$ [правый отступ]).
- `border-box` сообщает браузеру о необходимости включения в качестве составляющей части значений свойств `width` и `height` толщину как отступа, так и границ. Эта настройка решает ту проблему использования значений, выраженных в процентном отношении, о которой говорилось выше. То есть когда свойству `box-sizing` присвоено значение `border-box`, при установке ширины элемента, равной 50 %, этот элемент будет занимать до 50 % пространства, даже если к нему будут добавлены отступы и границы.

Если вам не нравится стандартный способ вычисления браузером элементов ширины и высоты, добавьте значение `border-box`. (Если, конечно, у вас нет какой-нибудь причины, по которой вы хотите включить в расчеты отступ, но не хотите включать туда еще и границу.) Чтобы воспользоваться свойством `box-sizing`, предоставьте ему одно из трех значений из списка. Например:

```
box-sizing: border-box;
```

Между прочим, многие веб-разработчики настолько оценили пользу настройки `border-box`, что создали универсальный селектор, применяемый к каждому элементу на странице:

```
* {  
  box-sizing: border-box;  
}
```

В главе 14 будет показано, что особая польза от применения значения `border-box` проявится при решении определенных проблем, возникающих при верстке CSS-кода.

Управление блочными элементами с помощью свойства `overflow`

Когда содержимое форматированного элемента имеет размеры больше определенных свойствами `width` и `height`, происходят странные вещи. На рис. 7.13 показано, что в браузерах содержимое отображается за пределами (выступает наружу) границ элемента, часто накладываясь на него.

Браузер использует в этой ситуации свойство `overflow`. В качестве значения можно указать одно из четырех ключевых слов, определяющих, как должно отображаться содержимое, выходящее за пределы блочного элемента.

- `visible` — это значение, используемое браузером по умолчанию. Указание этого ключевого слова имеет тот же эффект, что отсутствие установки свойства `overflow` (рис. 7.14, *вверху*).
- `scroll` — позволяет добавить полосы прокрутки (см. рис. 7.14, *посередине*). Параметр создает своего рода окно мини-браузера внутри веб-страницы, которое выглядит подобно устаревшим HTML-фреймам. Вы можете использовать ключевое слово `scroll`, чтобы вместить объемное содержимое в ограниченную область. При таком варианте полосы прокрутки отображаются *всегда*, даже если содержимое помещается внутри блока.
- `auto` — пользуйтесь этим значением, чтобы сделать полосы прокрутки необязательными. Оно выполняет ту же функцию, что и `scroll`, с одним исключением — полосы прокрутки в данном случае появляются только при необходимости.
- `hidden` — скрывает любое содержимое, выходящее за пределы блочного элемента (см. рис. 7.14, *внизу*). Это значение небезопасно, поскольку может привести к тому, что часть содержимого будет скрыта.

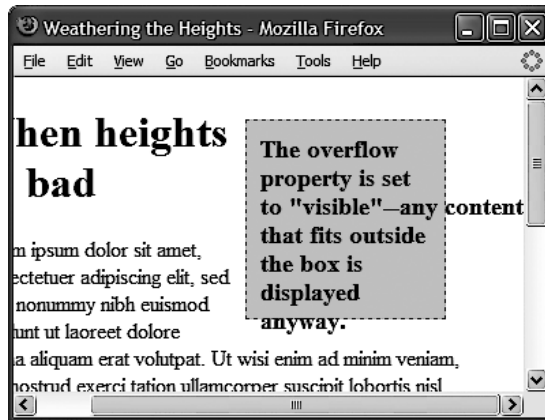


Рис. 7.14. Свойство `overflow` предоставляет три простых способа отображения текста, размеры которого не позволяют браузеру отобразить его внутри блочного элемента: `visible` показывает весь контент (вверху), `scroll` и `auto` добавляют полосы прокрутки (посередине), а `hidden` отображает только тот контент, который не выходит за границы области (внизу)

Задание максимальных и минимальных значений высоты и ширины

Если вы еще не поняли, хочу вас заверить, что каскадные таблицы стилей предлагают множество гибких решений. Кроме стандартных свойств `width` и `height`, можно воспользоваться еще четырьмя их вариантами.

- Свойство `max-width` устанавливает максимальную ширину элемента. Этот элемент может быть уже установленного предела, но не может быть шире. Такой вариант пригодится в том случае, когда нужно, чтобы ваша страница меняла свои размеры, чтобы поместиться на экранах разной ширины, но при этом нужно, чтобы она не становилась настолько широкой, что это затруднит ее чтение на мониторах с большой диагональю. Предположим, на страницу добавлен следующий стиль:

```
body {  
  max-width: 1200px;  
}
```

Этот стиль позволяет странице перестраивать контент, чтобы поместиться по ширине на небольших экранах смартфонов или планшетных компьютеров. Но на действительно больших настольных мониторах страница не должна быть шире 1200 пикселей, то есть она не может разрастаться в ширину, при которой ее контент уже невозможно читать.

- Свойство `max-height` работает во многом похоже на `max-width`, за исключением того, что оно применяется к высоте элемента. Но, как уже упоминалось, с высотой элемента лучше все же не связываться.
- Свойство `min-width` устанавливает минимальную ширину элемента. Элемент может растянуться шире значения минимальной ширины, но никогда не может стать уже этого значения. Если, к примеру, вы заметили, что при изменении размеров окна вашего браузера элементы становятся настолько узкими, что макет рассыпается, можно с помощью следующего кода установить ограничение минимальной ширины элемента:

```
body {  
  min-width: 760px;  
}
```

Если посетитель уменьшит окно своего браузера менее чем до 760 пикселей по ширине, браузер добавит полосу прокрутки, не сужая элементы на странице до крайности.

- Свойство `min-height` работает точно так же, как и `min-width`, но применительно к высоте. Оно позволяет решить проблему, показанную на рис. 7.13. Ограничивая минимальную высоту, вы заставляете браузер сделать элемент по крайней мере имеющим определенную высоту. Если содержимое элемента выше, то браузер сделает выше весь элемент.

Управление обтеканием контента с помощью плавающих элементов

HTML-контент в обычном порядке отображается, начиная с верхнего края окна браузера по направлению к нижнему: один заголовок, абзац, блочные элементы друг за другом. Такой способ представления неинтересен, но благодаря каскадным таблицам стилей предоставляется возможность изменить дизайн в лучшую сторону. В части III мы рассмотрим много новых методов компоновки, размещения, упорядочения элементов, но уже сейчас вы можете добавить привлекательности своим веб-страницам посредством свойства `float`.

Свойство `float` перемещает любой элемент в нужную позицию, выравнивая его по левому или правому краю веб-страницы. В процессе перемещения содержимое, находящееся ниже позиционируемого плавающего элемента, смещается вверх и плавно обтекает сам плавающий элемент (рис. 7.15, *внизу*). Плавающие (или перемещаемые) элементы — идеальное средство для того, чтобы выделить дополнительную информацию из основного текста. Изображения могут смещаться к любому краю веб-страницы, обеспечивая перенос расположенного рядом текстового контента и его изящное обтекание вокруг изображений. Точно так же вы можете переместить боковую панель с относящейся к тексту информацией и элементы навигации к одной из сторон веб-страницы.

На рис. 7.15 приоритет элементов следующий: заголовок, абзац, контейнер `div` и т. д. Свойство `float` позволяет нарушать этот порядок, ввиду чего и является одним из самых мощных инструментов каскадных таблиц стилей. Широкий диапазон его применения не только включает простое позиционирование, перемещение изображений к одной из сторон абзаца, но и обеспечивает полный контроль по управлению размещением баннеров, меню, панелей навигации и других элементов веб-страниц.

Несмотря на то что свойство `float` может быть использовано для сложного, комплексного форматирования (об этом вы прочитаете в главе 13), его применение не представляет никаких сложностей. В качестве значения указывается одно из трех ключевых слов — `left`, `right` или `none`. К примеру:

```
float: left;
```

- `left` — перемещает формируемый элемент влево, при этом содержимое, находящееся ниже, обтекает правый край элемента.
- `right` — перемещает элемент вправо.
- `none` — отменяет обтекание и возвращает объект в его обычную позицию. Это свойство определяет нормальное поведение элемента, поэтому используйте его, если хотите отменить обтекание слева или справа, примененное к другому стилю (правила взаимодействия нескольких стилей обсуждались в главе 5).

Плавающие элементы могут быть расположены у левого или правого края содержащего их *элемента-контейнера*. В некоторых случаях это означает, что элемент перемещается к левому или правому краю окна браузера. Однако если вы

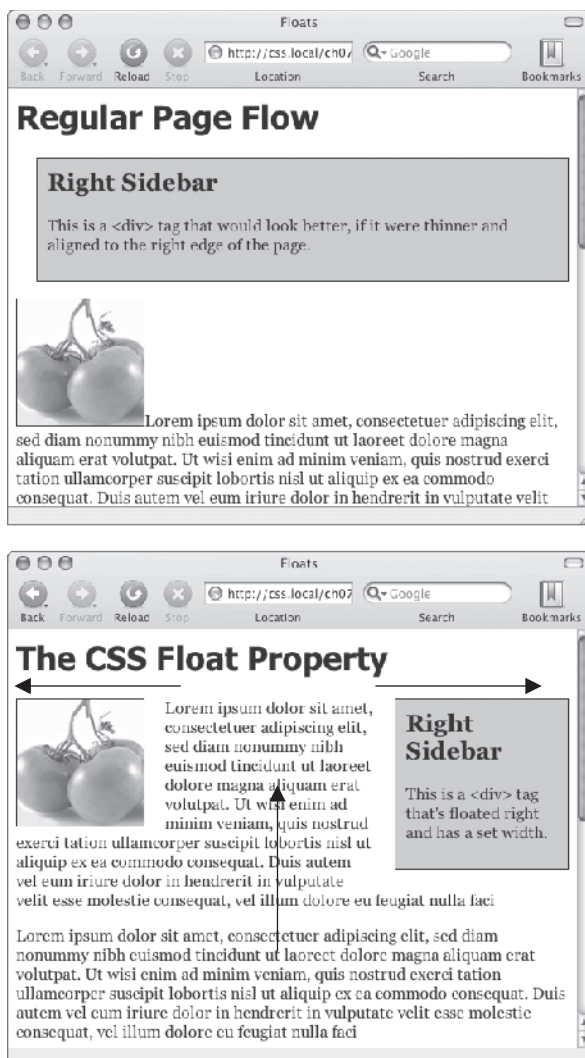


Рис. 7.15. Стандартная последовательность отображения HTML-контента слева направо, сверху вниз

перемещаете элемент, находящийся внутри другого элемента, для которого установлены значения ширины или позиции на веб-странице, то перемещение будет осуществлено к левому или правому краю *этого элемента*, который является контейнером для плавающего элемента. Например, на веб-странице может быть блочный элемент шириной 30 % от ширины окна браузера, который прилегает к правому краю окна браузера. Внутри может располагаться изображение, которое выровнено по левому краю. Иными словами, изображение примыкает к левому краю этого блока, а не окна браузера.

Вы можете применять свойство `float` к строчным элементам, например `img`. Надо сказать, использование выравнивания по левому или правому краю для фотографий — самый обычный, даже наиболее распространенный способ применения свойства `float`. Браузер обрабатывает строчные элементы точно так же, как блочные. Поэтому, добавляя отступы и поля в строчных элементах, вы избежите проблем, которые обычно возникают в такой ситуации.

Вы можете также использовать свойство `float` для блочных элементов, таких как заголовки или абзацы текста (или HTML5-элементы, например `article`, `section` и `aside`). Общая методика применения свойства `float` к элементу `div`, содержащему другие HTML-элементы, заключается в создании своего рода блока-контейнера. Так вы можете создавать боковые панели, плавающие цитаты и другие обособленные элементы веб-страниц (пример приведен в практикуме этой главы). При использовании свойства `float` для блочных элементов рекомендуется установить для них свойство `width` (на самом деле правила CSS требуют установки ширины для всех плавающих элементов, кроме изображений). Таким образом, вы можете управлять размером блока по горизонтали, оставив промежуток для текстового контента, расположенного после этого блочного элемента, и, соответственно, задать его поведение — обтекание.

ПРИМЕЧАНИЕ

Порядок написания HTML-кода оказывает большое влияние на отображение плавающих элементов веб-страницы. HTML-код плавающих элементов должен находиться прежде любого обтекающего HTML-контента. Допустим, вы создали веб-страницу, на которой имеется элемент заголовка `h1`, а за ним расположен абзац `p`. Внутри абзаца `p` вы вставили фотографию с помощью элемента `img`. Теперь, если вы установите выравнивание фотографии по правому краю, заголовок `h1` и часть содержимого абзаца `p` будут по-прежнему отображены над фотографией. Только содержимое, следующее за элементом `img`, будет обтекать изображение с левой стороны.

Фон, границы и обтекание

К недовольству многих веб-дизайнеров, фон и границы не переносятся таким же образом, как другие элементы веб-страницы. Допустим, у вас есть боковая панель, примыкающая к правому краю веб-страницы. Содержимое страницы, расположенное под ней, как ему и положено, приподнимается выше и обтекает панель. Однако если этому контенту страницы назначены настройки фона или границ, они отображаются, фактически *проступая* под боковой панелью (рис. 7.16, *слева*). По сути, браузер окружает плавающий элемент только текстом, но не границами или фоном. Как это ни удивительно, но именно так работает механизм обтекания. Возможно, вы не будете следовать этим правилам; а захотите, чтобы границы, фоновый цвет или рисунок не отображались при достижении плавающего элемента (см. рис. 7.16, *справа*). С помощью каскадных таблиц стилей вы можете сделать и это.

Первый метод заключается в добавлении еще одного свойства в стиль, устанавливающий параметры фона и границ и оказывающий воздействие на плавающий элемент. Нужно добавить в этот стиль следующий код: `overflow: hidden;`. Свойство `overflow` (описано в предыдущем разделе) отменяет отображение фона или границ под плавающими элементами.

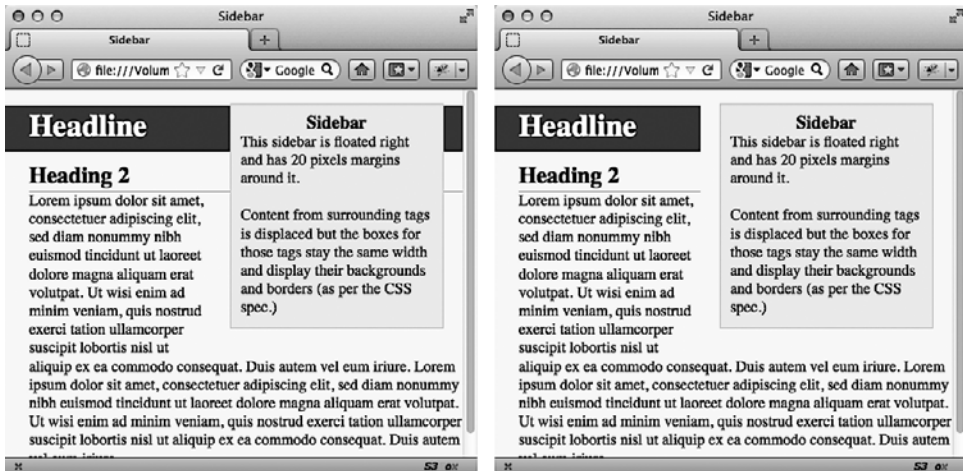


Рис. 7.16. Добавление свойства `overflow: hidden;` в стиль элемента `h1` препятствует тому, чтобы атрибуты заголовка повлияли на отображение фона и границ под плавающим элементом (боковая панель)

Другой подход состоит в добавлении границ вокруг плавающего элемента. Если вы создаете достаточно толстую границу цвета фона страницы, то не будете даже догадываться о существовании границ, находящихся под плавающим элементом, так как они будут скрыты.

Запрет обтекания

Иногда требуется отобразить содержимое элемента таким образом, чтобы на него не влияла способность обтекания плавающего элемента. Например, веб-страница может содержать примечание с текстом авторских прав, которое в любом случае должно отображаться в самой нижней части окна браузера. Если у вас имеется высокая боковая панель, примыкающая к левому краю веб-страницы, то примечание с авторскими правами может подняться вверх и отобразиться с обтеканием вокруг плавающего элемента. Таким образом, вместо того чтобы быть внизу страницы, текст появится гораздо выше, на одном уровне с боковой панелью. Вам же нужно, чтобы для примечания с авторскими правами было отменено обтекание, установленное свойством плавающей боковой панели.

Сложности другого рода возникают при наличии нескольких плавающих элементов, расположенных рядом. Когда они имеют небольшую ширину, то поднимаются вверх, располагаясь на одном уровне. В противном случае элементы могут образовать некрасивую «пустоту» (рис. 7.17, *вверху*). В данном случае плавающие элементы *не должны* отображаться рядом друг с другом. Для решения этой проблемы в языке CSS предусмотрено свойство `clear`.

Свойство `clear` сообщает браузеру о том, что формируемый элемент *не должен обтекать* плавающий элемент. Оно приводит к тому, что элемент смещается вниз

по тексту веб-страницы, располагаясь ниже плавающего элемента. Кроме того, вы можете запретить обтекание с любой стороны элемента (слева или справа) или полностью удалить все параметры обтекания.

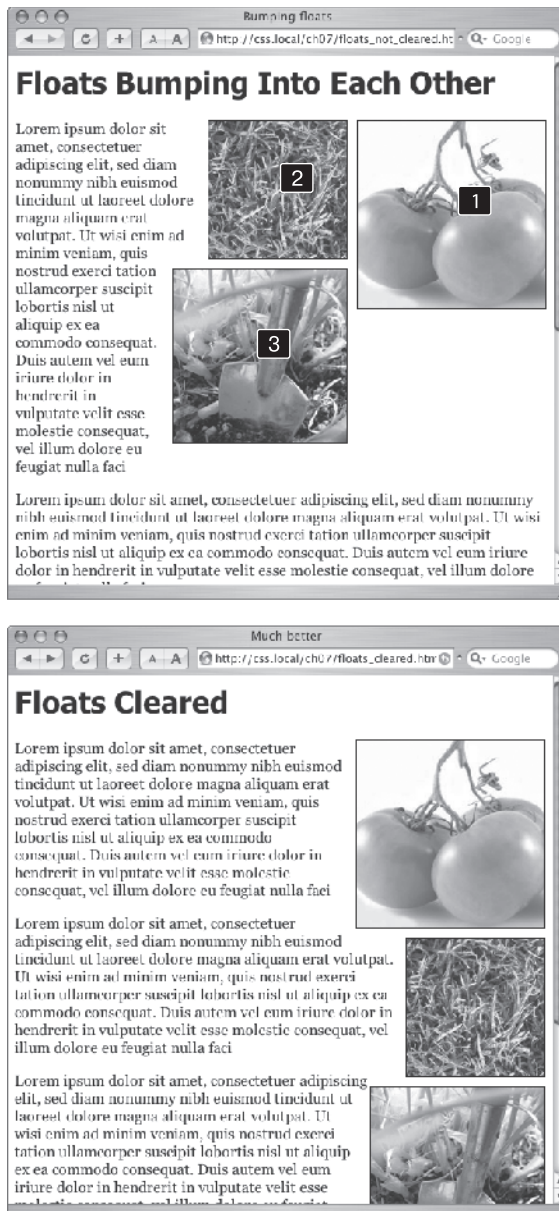


Рис. 7.17. Свойство `clear`, примененное к изображению (2), предотвращает наложение на другое изображение (1). Применение свойства `clear: right`; к фотографии (3) предотвращает ее размещение под фото 2

Свойство `clear` допускает следующие значения:

- `left` — формируемый элемент смещается вниз относительно плавающего с установленным обтеканием слева, но обтекание справа остается в силе;
- `right` — формируемый элемент смещается вниз относительно плавающего с установленным обтеканием справа, обтекание слева остается в силе;
- `both` — вызывает перемещение формируемого элемента вниз относительно плавающего с установленным обтеканием и слева и справа;
- `none` — полностью отменяет запрет обтекания — сообщает браузеру, что формируемый элемент должен вести себя так, как предусмотрено, то есть он может обтекать плавающие элементы как с левой, так и с правой стороны.

В нашем примере текст примечания с авторским правом должен всегда отображаться внизу веб-страницы и не должен обтекать другое содержимое. Ниже приведен код класса, который делает это:

```
.copyright {  
  clear: both;  
}
```

На рис. 7.17 показано, как свойство `clear` может препятствовать беспорядочному отображению плавающих элементов различной высоты. Для всех трех представленных на рисунке фотографий установлено обтекание по правому краю. В верхней части рисунка фото помидоров (1) — первое изображение на веб-странице, оно расположено в верхнем правом углу. На отображение второй фотографии (2) оказывает влияние параметр обтекания первого изображения, и оно обтекает слева первое фото. Последнее изображение (3) слишком широкое, чтобы отобразиться рядом (на одном уровне) со вторым (2), но все-таки оно пытается расположиться слева по отношению как к первому (1), так и ко второму (2) изображениям. Однако ширина рисунка не позволяет этого сделать.

Применение здесь свойства `clear: right;` препятствует размещению фотографий на одном уровне (см. рис. 7.17, *внизу*). Запрет обтекания для второй фотографии не допускает ее отображения рядом с первой, а запрет обтекания для третьей предотвращает ее отображение на одном уровне со второй.

ПРИМЕЧАНИЕ

Применение параметров левого и правого обтеканий, а также запрет обтекания кажется достаточно сложным механизмом. Вообще, так оно и есть. В этом разделе приведены основные понятия. Мы снова вернемся к этой теме в главе 13, там вы познакомитесь с использованием обтекания в более сложных реализациях.

Практикум: поля, фон и границы

В этом практикуме мы исследуем элементы блочной модели CSS, потренируемся в настройке пространства вокруг объектов веб-страниц, применим к элементам красочные границы, научимся управлять размерами и обтеканием элементов веб-страниц.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практи-

кума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку Download ZIP в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 07.

Управление фоном и полями страницы

Начнем с простого HTML-файла, содержащего внутреннюю таблицу стилей со сбросом стандартных настроек стилей CSS. Пока на странице нет ничего интересного (рис. 7.18).

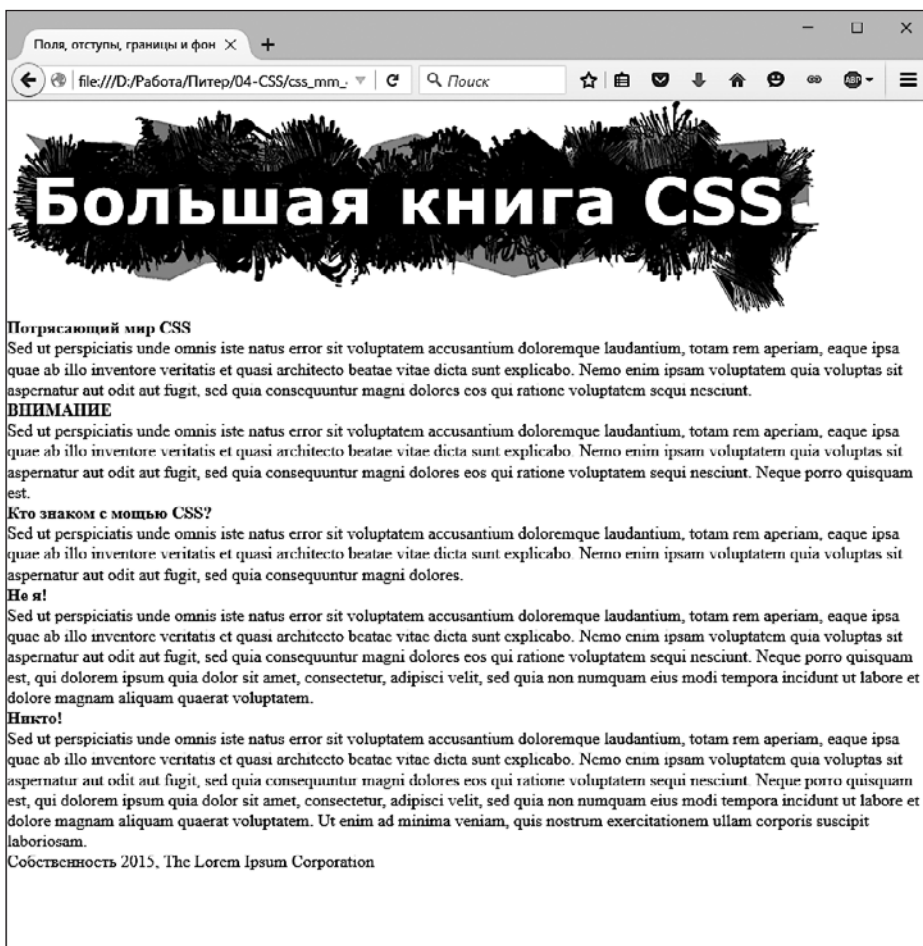


Рис. 7.18. Страница представляет собой простой HTML-документ и содержит единственный стиль, сбрасывающий множество встроенных настроек форматирования браузера. Она будет выглядеть намного лучше после изменения с использованием блочной модели CSS

СОВЕТ

Увидеть конечный результат практикума можно на рис. 7.21.

1. Откройте в редакторе HTML-кода страницу `main.html` из папки `07`.

Эта таблица стилей уже связана с файлом `index.html`, поэтому стили, добавленные в него, применяются и к странице. В таблице используется набор стилей, который рассматривался в главе 5, в подразделе «С чистого листа» раздела «Управление каскадностью». Он удаляет все поля, отступы, установленные значения размеров шрифта, начертания шрифта из наиболее значимых блочных элементов и устраняет множество кросс-браузерных проблем с отображением элементов, с которыми вы могли столкнуться из-за этих свойств.

Наверное, наиболее важные свойства в нем — `margin` и `padding`, установленные в первом стиле. Существует достаточное количество кросс-браузерных странностей, связанных с этими двумя свойствами, поэтому вы должны всегда обнулять их и начинать с чистого листа. Другой распространенной альтернативой является таблица стилей, которая устраняет различия в отображении страницы в различных браузерах, но сохраняет основные поля (tinymce.com/cop5s89).

2. В нижней части файла `main.css` щелкните кнопкой мыши сразу после комментария `/* end reset browser styles */` и добавьте селектор тега:

```
html {  
    background-color: rgb(253,248,171);  
}
```

Этот стиль устанавливает светло-желтый фон для страницы. Если вы хотите задать цвет для фона веб-страницы, свойство `background-color` можно добавлять либо к элементу `html`, либо к `body`. Далее мы добавим поля, границы и другие свойства элемента `body`.

ПРИМЕЧАНИЕ

Возможно, вы привыкли вместо задания цвета по модели RGB использовать шестнадцатеричные значения (например, `#FDF8AB`). Чтобы осуществлять преобразования между этими двумя моделями, можно воспользоваться онлайн-конвертером по адресу colorhexa.com. Но предпочтительнее применять модель RGB, поскольку ее вариация RGBA, имеющая дополнительный параметр прозрачности, достаточно полезна, чтобы остановиться на одной системе указания цвета (RGB), а не смешивать две (RGB и шестнадцатеричные значения).

3. Добавьте еще один стиль в таблицу:

```
body {  
    background-color: rgb(255,255,255);  
    border: 3px solid rgb(75,75,75);  
}
```

Этот стиль добавляет к элементу `body` белый цвет фона и темно-серую сплошную границу шириной 3 пиксела. Поскольку `body` расположен внутри элемента `html`, браузер считает, что он находится «поверх» элемента `html`, и белый цвет покрывает фон желтого цвета, установленный на предыдущем шаге. Далее мы зададим ширину элемента `body` и настроим его отступы и поля.

СОВЕТ

Обычно, если вы добавляете свойство `background-color` к элементу `body`, цвет заполняет все окно браузера. Однако если вы также добавите цвет фона для элемента `html`, фон `body` будет заполнять

только область контента. Чтобы увидеть это в действии, просмотрите веб-страницу после шага 4, затем удалите стиль для элемента `html` и просмотрите страницу снова. Странная, но полезная мелочь каскадных таблиц стилей.

4. Измените стиль элемента `body`, который вы только что создали, добавив пять новых свойств (изменения выделены полужирным шрифтом):

```
body {  
  background-color: rgb(255,255,255);  
  border: 3px solid rgb(75,75,75);  
  max-width: 760px;  
  margin-top: 20px;  
  margin-left: auto;  
  margin-right: auto;  
  padding: 15px;  
}
```

Свойство `max-width` ограничивает тело (элемент `body`) страницы 760 пикселями по ширине: если окно браузера посетителя окажется шире, чем 760 пикселей, он увидит фоновый цвет элемента `html` и область шириной 760 пикселей с белым фоном элемента `body`. Если же окно браузера станет уже указанного размера, блок текста также сузится, заполняя окно, что упрощает просмотр страницы на небольших экранах планшетов и смартфонов.

Свойство `margin-top` добавляет 20 пикселей пространства от верхнего края окна браузера, смещая содержимое элемента `body` немного вниз, а левое и правое поля центрируют его, размещая в середине окна браузера. Значение `auto` — это еще один способ сообщить браузеру: «Разбирайся в этом сам», и, поскольку оно применяется как к левому, так и к правому полям, браузер создает одинаковые промежутки слева и справа.

ПРИМЕЧАНИЕ

Вы можете также использовать сокращенную запись свойства `margin`, устанавливая настройки полей, если наберете одну строку кода:

```
margin: 20px auto 0 auto;
```

Наконец, для того, чтобы предотвратить прикосновение содержимого элемента `body` к линии границы, отступ шириной 15 пикселей добавлен к внутренней части содержимого с помощью свойства `padding`. Другими словами, для изображения и текста был задан отступ 15 пикселей от всех четырех краев. Затем мы добавим свечение вокруг блока, воспользовавшись свойством `box-shadow`.

5. Отредактируйте только что созданный стиль элемента `body`, добавив к нему еще одно свойство после строки `border`, но перед `max-width` (изменения выделены полужирным шрифтом):

```
body {  
  background-color: rgb(255,255,255);  
  border: 3px solid rgb(75,75,75);  
  box-shadow: 0 0 15px 5px rgba(44,82,100,.75);  
  max-width: 760px;
```

```
margin-top: 20px;
margin-left: auto;
margin-right: auto;
padding: 15px;
}
```

Этот стиль добавляет к блоку свечение путем создания 15-пиксельной тени, помещенной непосредственно за блоком (то, что параметры тени начинаются с 0 0, означает, что тень не имеет смещения вправо-влево или вверх-вниз и представляет собой фон). Значение 5px определяет расширение тени, выталкивая ее на 5 пикселей дальше всех четырех углов. И наконец, значение rgba устанавливает темно-синий цвет с уровнем непрозрачности 75 % (то есть сквозь него можно видеть желтый фон).

В вашей таблице стилей уже много всего, поэтому пришло время проверить, как выглядит страница.

6. Сохраните файл и просмотрите веб-страницу в браузере.

Вы должны увидеть белый блок с изображением, фрагмент текста и серый контур с синеватым свечением, плавающие на желтом фоне (рис. 7.19). В данном случае настройка ширины элемента `body`, а также добавление кода `margin-left: auto;` и `margin-right: auto;` помещает его прямо в центре окна браузера. Для выравнивания объекта по вертикали (сохранения одинаковых расстояний сверху и снизу от объекта до краев страницы), необходимо использовать нечто большее, чем свойства `margin`. Чтобы узнать способы выравнивания по вертикали и горизонтали, посетите страницу tinyurl.com/qhad4mq.

Теперь следует уделить внимание тексту, чем мы и займемся далее.

Настройка интервалов между элементами

Поскольку стили, сбрасывающие стандартные настройки CSS, «оголили» текст на странице, вам необходимо создать новые стили, которые преобразили бы заголовки и абзацы. Начнем с элемента `h1` в верхней части страницы.

1. Вернитесь к файлу `main.html` в редакторе HTML-кода. Установите курсор после закрывающей скобки селектора `body`, нажмите клавишу `Enter` и добавьте следующий стиль:

```
h1 {
  font-size: 2.75em;
  font-family: Georgia, "Times New Roman", Times, serif;
  font-weight: normal;
  text-align:center;
  letter-spacing: 1px;
  color: rgb(133,161,16);
  text-transform: uppercase;
}
```

Он использует множество свойств для форматирования текста, которые мы обсуждали в предыдущей главе. Верхний заголовок имеет высоту 2,75 em (44 пиксела в большинстве браузеров) и набран прописными буквами. Для него уста-

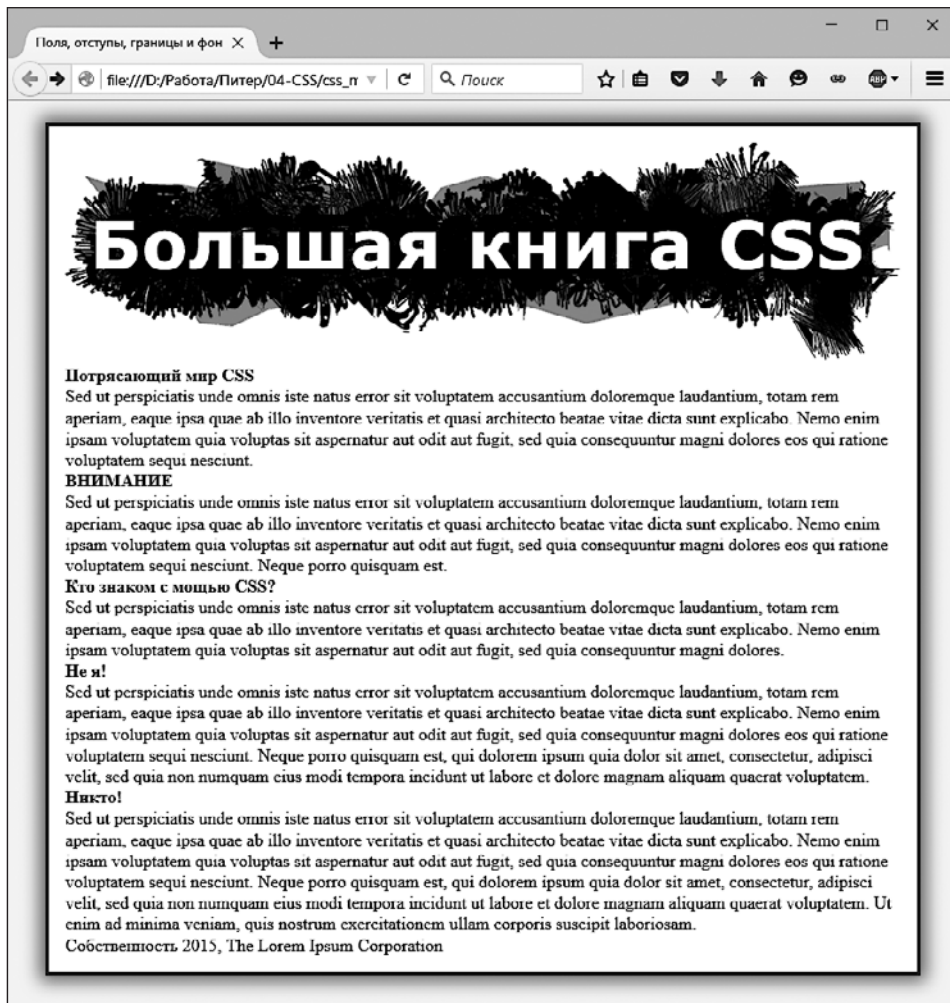


Рис. 7.19. Устанавливая значение `auto` для левого и правого полей любого элемента, имеющего заданную ширину, вы центрируете этот элемент

новлен шрифт *Georgia* зеленого цвета, есть небольшой промежуток между буквами. Свойство `text-align` обеспечивает выключку текста по центру блока. Очень интересным получается добавление фонового цвета для выделения заголовка.

СОВЕТ

Сохраняйте файл и просматривайте страницу в браузере после каждого шага из этого примера. Таким образом вы поймете, как приведенные здесь свойства CSS воздействуют на элементы, которые они формируют.

2. Добавьте новое свойство к селектору тега `h1`, чтобы он выглядел следующим образом (изменения выделены полужирным шрифтом):

```
h1 {
  font-size: 2.75em;
  font-family: Georgia, "Times New Roman", Times, serif;
  font-weight: normal;
  text-align:center;
  letter-spacing: 1px;
  color: rgb(133,161,16);
  text-transform: uppercase;
  background-color: rgb(226,235,180);
}
```

Если вы просмотрите страницу сейчас, то увидите, что заголовок обрел светло-зеленый фон. Когда фон применяется к блочным элементам, таким как заголовок, он заполняет все доступное горизонтальное пространство (другими словами, цвет не только появляется за текстом «Потрясающий мир CSS», но и простирается вплоть до правого края окна).

Текст заголовка немного стесненный — буква «П», с которой он начинается, кажется края фона. С помощью небольших отступов вы можете исправить это.

3. Добавьте еще одно свойство к селектору тега `h1`, чтобы он выглядел следующим образом (изменения выделены полужирным шрифтом):

```
h1 {
  font-size: 2.75em;
  font-family: Georgia, "Times New Roman", Times, serif;
  font-weight: normal;
  text-align:center;
  letter-spacing: 1px;
  color: rgb(133,161,16);
  text-transform: uppercase;
  background-color: rgb(226,235,180);
  padding: 5px 15px 2px 15px;
}
```

Сокращенное свойство `padding` предоставляет быстрый способ добавить отступы вокруг всех четырех сторон контента — в данном случае отступ шириной 5 пикселей добавляется над текстом, 15 пикселей — справа, 2 пиксела — внизу и 15 пикселей — слева.

Существует еще одна проблема с заголовком: из-за отступов, которые добавлены к элементу `body` (см. шаг 4 предыдущего задания), заголовок (включая фоновый цвет) отодвинут на 15 пикселей от левого и правого краев серой границы, окружающей содержимое. Заголовок станет выглядеть лучше, если его фоновый цвет будет касаться этого контура. Это не проблема: на помощь приходят отрицательные значения полей.

4. Добавьте последнее свойство к селектору тега `h1`, чтобы он выглядел следующим образом (изменения выделены полужирным шрифтом):

```
h1 {
  font-size: 2.75em;
  font-family: Georgia, "Times New Roman", Times, serif;
```

```
font-weight: normal;
text-align:center;
letter-spacing: 1px;
color: rgb(133,161,16);
text-transform: uppercase;
background-color: rgb(226,235,180);
padding: 5px 15px 2px 15px;
margin: 0 -15px 20px -15px;
}
```

Здесь сокращенное свойство `margin` устанавливает следующие размеры полей: 0 пикселей для верхнего, –15 пикселей для правого, 20 пикселей для нижнего и –15 пикселей для левого. Нижнее поле добавляет немного пространства между заголовком и абзацем, который идет за ним. Следующий прием заключается в использовании отрицательных значений для левого и правого полей. У нас есть возможность назначить отрицательные значения полей для какого-либо элемента. Это свойство «тянет» элемент по направлению поля — в данном случае, заголовок продлевается на 15 пикселей влево и 15 пикселей вправо, расширяясь и вытягиваясь поверх отступов элемента `body`.

5. Теперь вы немного отформатируете элемент `h2`. Добавьте следующий стиль после стиля `h1`:

```
h2 {
font-size: 1.5em;
font-family: "Arial Narrow", Arial, Helvetica, sans-serif;
color: rgb(249,107,24);
border-top: 2px dotted rgb(141,165,22);
border-bottom: 2px dotted rgb(141,165,22);
padding-top: 5px;
padding-bottom: 5px;
margin: 15px 0 5px 0;
}
```

Этот стиль добавляет базовое форматирование текста и пунктирные границы сверху и снизу заголовка. Чтобы добавить немного пространства между текстом заголовка и строками, используются небольшие отступы сверху и снизу. Наконец, свойство `margin` добавляет поля шириной 15 пикселей над заголовком и 5 пикселей под ним.

6. Сохраните файл и просмотрите страницу в браузере.

Заголовки выглядят прекрасно (рис. 7.20). Далее вы создадите боковую панель в правой части страницы.

Создание боковой панели

Боковые панели — элементы, применяемые в большинстве печатных журналов и газет. Они отделяют небольшие блоки информации, например перечни статей, и акцентируют на них внимание. Но для того, чтобы боковые панели были достаточно эффективными и полезными, они не должны прерывать потока основного

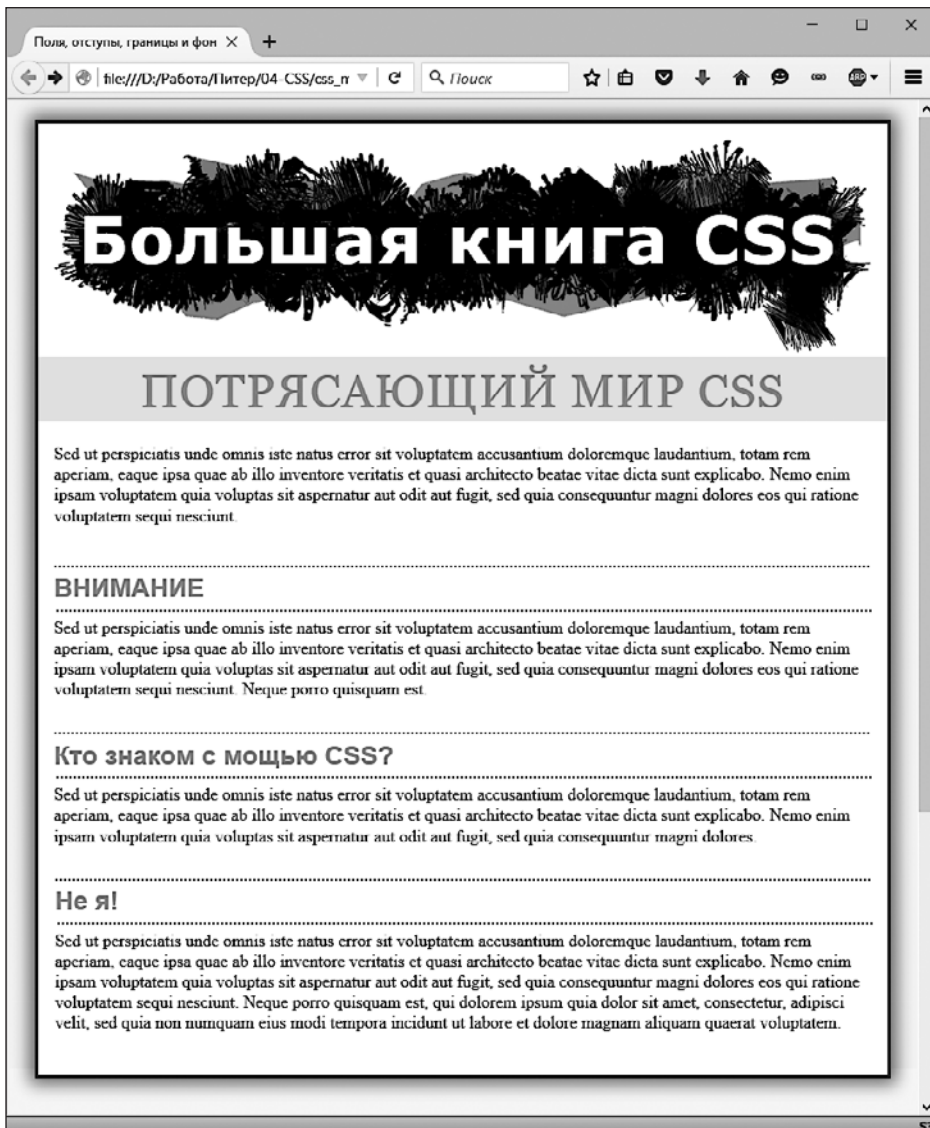


Рис. 7.20. С помощью нескольких стилей вы можете изменить фоновый цвет веб-страницы, добавить поля, регулировать интервалы между заголовками и абзацами

контента. Даже название «боковая панель» говорит о том, что этот блок должен быть расположен обособленно и примыкать к краю веб-страницы, что можно легко сделать средствами каскадных таблиц стилей.

1. Вернитесь к файлу `index.html` в редакторе HTML-кода.

Сначала нужно изолировать область веб-страницы, составляющую боковую панель. Для этого прекрасно подходит элемент `div`. С его помощью можно заключить любой объем HTML-кода в отдельный блок.

2. Прокрутите страницу вниз и щелкните кнопкой мыши перед первым элементом h2 (с заголовком **Внимание**). Введите код `<div class="sidebar">` и нажмите клавишу **Enter**.

Этот HTML-код отмечает начало блока боковой панели и применяет к нему класс. Мы создадим класс `.sidebar` позже, а сначала нужно определить завершение блока боковой панели, закрыв контейнер тегом `</div>`.

3. Перейдите к закрывающему тегу `</p>` абзаца, который следует сразу за элементом h2 (это тот тег `</p>`, который расположен перед строкой `<h2>Кто знаком с мощностью CSS?</h2>`). Нажмите клавишу **Enter** и введите тег `</div>`.

Мы только что заключили заголовок и маркированный список в элемент `div`. Теперь создадим для него стиль.

4. Вернитесь к файлу `main.css` и добавьте после созданного ранее селектора h2 следующий код:

```
.sidebar {  
  width: 30%;  
  float: right;  
  margin: 10px;  
}
```

Этот стиль устанавливает ширину области контента (в которой отображается текст) равной 30%. В этом случае ширина боковой панели составляет 30 % от ширины ее контейнера. Контейнером является элемент `body`, и его ширина не превысит 760 пикселей. Свойство `float` перемещает боковую панель в правую часть блока, а свойство `margin` добавляет 10 пикселей пространства вокруг панели.

Если вы просмотрите веб-страницу в браузере, то увидите, что форма и положение блока боковой панели определены, но есть одна проблема: границы элементов h2 отображаются *под* самим блоком. Несмотря на то что плавающая боковая панель смещает текст заголовков, границы остаются на том же месте. Один из способов решить эту проблему — добавить цвет фона для боковой панели, чтобы не видеть границ h2 (но есть и другой способ, который вы будете использовать на шаге 8).

5. Добавьте еще одно свойство к стилю `.sidebar`, чтобы он выглядел следующим образом (изменения выделены полужирным шрифтом):

```
.sidebar {  
  width: 30%;  
  float: right;  
  margin: 10px;  
  background-color: rgb(250,235,199);  
  padding: 10px 20px;  
}
```

Это свойство добавляет светло-оранжевый цвет к боковой панели и смещает текст от границ боковой панели, чтобы он не касался границ, которые вы собираетесь добавить.

6. Добавьте еще два свойства к стилю `.sidebar`, чтобы он выглядел следующим образом (изменения выделены полужирным шрифтом):

```
.sidebar {
  width: 30%;
  float: right;
  margin: 10px;
  background-color: rgb(250,235,199);
  padding: 10px 20px;
  border: 1px dotted rgb(252,101,18);
  border-top: 20px solid rgb(252,101,18);
}
```

Это пример удобной методики, описанной ранее. Если вы хотите, чтобы бóльшая часть границ вокруг элемента была одинаковой, можно сначала определить границу для всех четырех краев — в данном случае пунктирную оранжевую линию толщиной 1 пиксел вокруг всей боковой панели. Затем можно применить новые свойства для отдельных границ, которые вы хотите изменить, — в данном примере верхняя граница будет сплошной и будет иметь высоту 20 пикселей. Такой способ позволяет использовать всего две строки кода, а не четыре (`border-top`, `border-bottom`, `border-left` и `border-right`).

Затем мы добавим скругленные углы и тень, чтобы выделить эту боковую панель.

7. И наконец, добавьте еще два свойства к стилю `.sidebar`, придав ему следующий вид (изменения выделены полужирным шрифтом):

```
.sidebar {
  width: 30%;
  float: right;
  margin: 10px;
  background-color: rgb(250,235,199);
  padding: 10px 20px;
  border: 1px dotted rgb(252,101,18);
  border-top: 20px solid rgb(252,101,18);
  border-radius: 10px;
  box-shadow: 5px 5px 10px rgba(0,0,0,.5);
}
```

Свойство `border-radius` позволяет создать скругленные углы. В данном случае значение `10px` предоставляет заметное скругление. Свойство `box-shadow` добавляет тень, отбрасываемую вниз и вправо от блока, придавая ему вид парящего над страницей. Теперь вы близки к завершению работы.

Заголовок внутри боковой панели выглядит не совсем так, как должен. К нему применяются те же свойства, что и к другим элементам `h2` (из-за стиля тега `h2`, который вы создали в шаге 4). Границы отвлекают внимание, а верхнее поле излишне смещает заголовок вниз от верхней части боковой панели. Для решения проблемы вы можете использовать селектор потомков, чтобы переопределить эти свойства.

8. После стиля `.sidebar` в файле `main.css` добавьте селектор потомков следующим образом:

```
.sidebar h2 {  
  border: none;  
  margin-top: 0;  
  padding: 0;  
}
```

Из-за селектора потомков `.sidebar` этот стиль является доминирующим, то есть имеет большую *специфичность* по сравнению с простым стилем `h2`. Он стирает границу, полученную от оригинального стиля элемента `h2`, вместе с верхним полем и всеми отступами. Тем не менее, поскольку в этом стиле не определены размер, цвет и начертание шрифта, эти свойства по-прежнему передаются от стиля `h2` — каскадность в действии!

Страница стала хорошо выглядеть, но границы элементов `h2` все еще появляются под боковой панелью. На это не очень приятно смотреть, но все можно легко исправить.

9. Найдите стиль `h2` и добавьте свойство `overflow`:

```
h2 {  
  font-size: 1.5em;  
  font-family: "Arial Narrow", Arial, Helvetica, sans-serif;  
  color: rgb(249,107,24);  
  border-top: 2px dotted rgb(141,165,22);  
  border-bottom: 2px dotted rgb(141,165,22);  
  padding-top: 5px;  
  padding-bottom: 5px;  
  margin: 15px 0 5px 0;  
  overflow: hidden;  
}
```

Присвоив свойству `overflow` значение `hidden`, вы скроете границы, которые проходят за пределами текста заголовка и под плавающим элементом.

10. Сохраните файл и просмотрите веб-страницу в браузере.
Таблица должна иметь вид, представленный на рис. 7.21.

Дополнительное задание

Чтобы закрепить полученные знания и навыки, попробуйте выполнить следующее практическое задание самостоятельно. Создайте для элемента `p` стиль, который бы смог приукрасить абзац: попробуйте добавить поля, указать цвет шрифта и т. д. Затем создайте класс для форматирования примечания с информацией об авторском праве, которое должно отображаться в нижней части страницы `index.html` (например, с именем `.copyright`). Добавьте в этот стиль верхнюю границу (над текстом примечания), измените цвет текста, уменьшите размер шрифта и измените регистр букв на прописные (используйте для этого свойство `text-transform`, описанное в разделе «Форматирование символов и слов» главы 6). После создания стиля добавьте соответствующий атрибут класса к элементу `p` в HTML-коде.

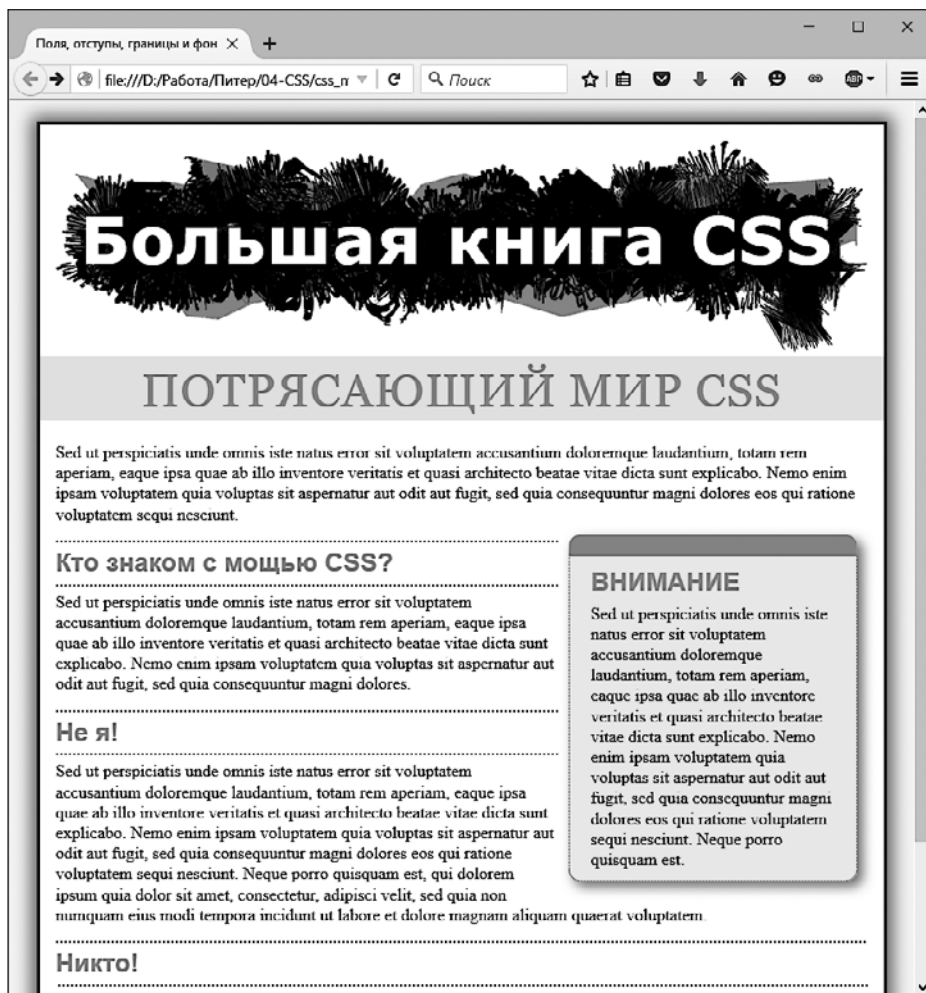


Рис. 7.21. Небольшой набор каскадных стилей придает непривлекательному HTML-коду элегантный дизайн. Обратите внимание, как плавающая боковая панель привлекает внимание и смещает в сторону основной текст

8 Добавление графики на веб-страницы

Как бы вы ни украшали текст веб-страниц с помощью границ и полей, ничто так не влияет на внешний вид сайта, как изображения. Каскадные таблицы стилей предлагают исчерпывающий набор средств управления ими. В веб-дизайне можно работать с графическими элементами двумя способами: посредством элемента `img` и свойства `background-image` (которое позволяет применить изображение к фону любого тега на странице).

В этой главе углубленно рассмотрены некоторые оригинальные методы применения изображений средствами языка CSS. На самом деле лучший способ узнать о возможностях графики и научиться ее использовать — увидеть результаты практических примеров, поэтому в конце главы в практикуме приведены три урока. В них мы создадим веб-страницу фотогалереи и попрактикуемся в быстрой и профессиональной разработке дизайна с применением изображений.

Каскадные таблицы стилей и элемент `img`

Элемент `img` представляет собой средство добавления изображений в веб-страницы фотогалерей еще со времен появления Всемирной паутины. Даже сайты без фотографий используют его для добавления логотипов, кнопок навигации и иллюстраций. В CSS нет свойств, специально предназначенных для форматирования изображений. Однако вы можете использовать для этого общие свойства, с которыми познакомились в предыдущих главах книги. Например, свойство `border` обеспечивает простой и быстрый способ заключить изображение в рамку или унифицировать вид галереи. Ниже представлен список свойств, наиболее часто используемых в отношении изображений.

- `border` — можно указать любое из множества свойств границ (см. раздел «Добавление границ» главы 7) для обрамления изображений. Пример приведен в практикуме этой главы. Поскольку каждая сторона изображения может иметь свои параметры границ: цвет, стиль, толщину линии — ваши творческие возможности расширяются.
- `padding` — добавляет пустой промежуток между границей и изображением. Отделение рамки от фотографии небольшим промежутком имитирует подложку, которая традиционно используется для обрамления рисунков с целью отделе-

ния их друг от друга. Устанавливая цвет фона, вы даже можете изменить цвет самой подложки.

- `float` — выравнивание изображения с помощью свойства `float` перемещает его к левому или правому краю веб-страницы. Или, если изображение расположено внутри другого блочного элемента, например боковой панели, — к левой или правой стороне этого элемента. Таким образом, текст и другие элементы веб-страницы обтекают изображение с другой свободной стороны и отображаются под ним. Вы можете создать многократное обтекание рисунков для отображения фотогалереи в несколько строк и столбцов.
- `margin` — чтобы добавить пустой промежуток между изображением и остальным содержимым веб-страницы, используйте свойство `margin`. Когда вы устанавливаете для изображения обтекание, текст обычно располагается слишком близко. Добавление левого поля (с выравниванием по правому краю) или правого поля (с выравниванием по левому краю) создает пустой отделяющий промежуток между текстом и графическим элементом.
- `border-radius` — свойство `border-radius` позволяет делать углы элементов, включая изображения, закругленными. С его помощью можно «закруглить» углы изначально прямоугольной фотографии с прямыми углами.

В большинстве случаев вам не потребуется создавать стиль для самого элемента `img`. Его форматирование затрагивает слишком большой диапазон элементов веб-страницы и изменит *все* изображения, включая элементы, обеспечивающие совершенно другие функции. Так, наряду с изображениями одинаковое форматирование будут иметь логотип, навигационные кнопки, фотографии и даже рекламные баннеры. Наверное, вы не хотели бы видеть одну и ту же черную рамку, обрамляющую все изображения. Вместо этого следует использовать классы, например `.galleryImage` или `.logo`, для применения стилей к нужным элементам выборочно.

Другой подход заключается в использовании селекторов потомков для целевого форматирования изображений, сгруппированных вместе в одном фрагменте веб-страницы. Если у вас есть галерея фотографий, можно заключить весь контент внутрь элемента `div` с классом `.gallery` и затем создать стиль только для изображений, расположенных внутри этого блока: `.gallery img`.

СОВЕТ

Присвойте значение `50%` свойству `border-radius`, чтобы сделать из прямоугольного изображения круглое. Например, вы можете создать класс `round` и применить его к рисунку, который должен отображаться в виде круга:

```
.round { border-radius: 50%; }
```

Добавление фоновых изображений

Свойство `background-image` — ключ к созданию визуально потрясающих сайтов. Стоит только научиться применять его и родственные свойства, и вы сможете заставить сайт выделяться среди прочих. Чтобы убедиться в эффективности фоновых изображений, перейдите по адресу csszengarden.com. HTML-код обеих веб-

страниц, показанных на рис. 8.1, совершенно одинаков; наибольшие визуальные различия достигаются благодаря использованию фоновых изображений (на самом деле, если вы взглянете на HTML-код этих веб-страниц, вы увидите, что в них нет ни единого элемента `img`).

Если вы уже создавали сайты раньше, то, наверное, использовали изображения в качестве фоновых рисунков веб-страниц. Скорее всего, они были небольших размеров, повторяющиеся на заднем плане окна браузера с едва различимым узором. Этот проверенный временем метод языка HTML использовал атрибут `background` элемента `body`. Каскадные таблицы стилей сделают все то же самое, но гораздо лучше.

ПРИМЕЧАНИЕ

Далее в книге вы найдете описание трех свойств фоновых изображений и изучите CSS-код каждого из них. Позже в этой главе вы познакомитесь с сокращенным методом набора — вариантами свойств, которые сэкономят немало времени.

Свойство `background-image` добавляет графический файл в качестве фона элемента. Чтобы поместить его на заднем плане всей веб-страницы, можно создать стиль для элемента `body`:

```
body {  
  background-image: url(images/bg.png);  
}
```

Свойство принимает единственное значение: ключевое слово `url`, за которым следует путь к графическому файлу, заключенный в круглые скобки. Вы можете использовать абсолютный URL-адрес, например, так: `url:(http://www.cosmofarmer.com/image/bg.gif)` — или относительный путь от документа или корневого каталога сайта:

```
url(../images/bg.gif) /* относительный путь от документа */  
url(/images/bg.gif) /* относительный путь от корневого каталога */
```

Как описывается в следующей врезке «В курс дела!», относительный путь указывает адрес файла таблицы стилей от документа CSS, а *не* формируемой HTML-страницы. Конечно, пути будут совпадать при использовании внутренней таблицы, но вы должны помнить об этом поведении, применяя *внешнюю* таблицу стилей. Предположим, у вас есть папка `styles` (содержащая таблицы стилей сайта) и папка `images` (с изображениями сайта). Обе расположены в главном (корневом) каталоге вместе с начальной (домашней) страницей сайта (рис. 8.2). Когда посетитель просматривает ее, загружается также внешняя таблица стилей (шаг 1 на рис. 8.2). Теперь допустим, что таблица включает стиль `body` с атрибутом фонового изображения, в котором в качестве рисунка выбран файл `bg.png` из папки `images`. Относительный путь по отношению к документу приведет от таблицы стилей к рисунку (шаг 2 на рис. 8.2). Это будет выглядеть следующим образом:

```
body {  
  background-image: url(../images/bg.png);  
}
```

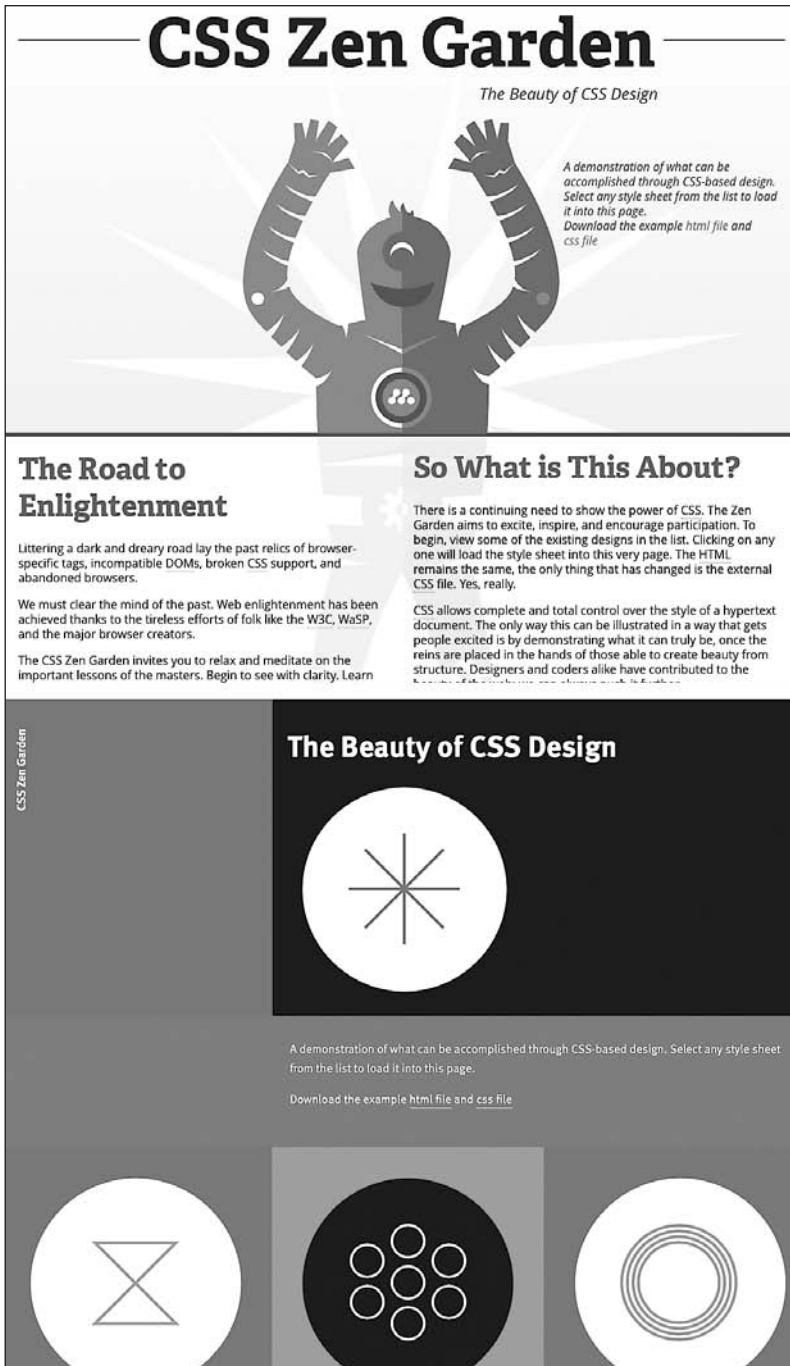


Рис. 8.1. Сайт csszengarden.com демонстрирует мощь каскадных таблиц стилей, показывая, как с помощью языка CSS вы можете превратить один и тот же HTML-файл в две совершенно разные веб-страницы

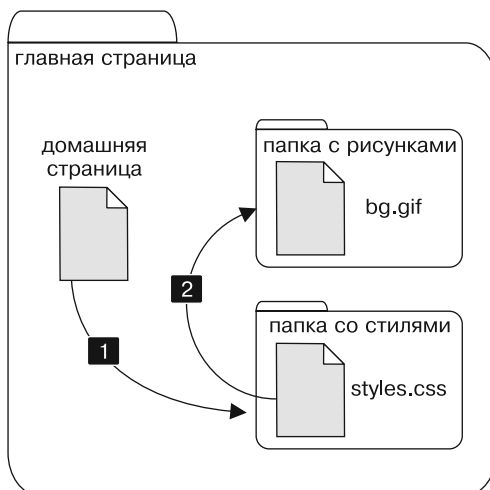


Рис. 8.2. Относительный путь от документа вычисляется применительно к файлу таблицы стилей, но не к формируемой веб-странице

Этот путь интерпретируется так: символы `../` означают «подняться вверх на один уровень», то есть к корневому каталогу, к папке, содержащей папку `styles`; `images/` означает «перейти к папке `images`», а `bg.png` определяет имя файла изображения.

В КУРС ДЕЛА!

GIF-, JPEG- и PNG-файлы: веб-графика

Компьютерная графика представлена сотнями различных форматов файлов с такими замысловатыми аббревиатурами, как JPEG, GIF, TIFF, PICT, BMP, EPS и т. д.

Веб-графика немного проще. Современные браузеры работают только с четырьмя графическими форматами: GIF, JPEG, PNG и SVG. Каждый из них обеспечивает хорошее *сжатие*. Благодаря возможностям компьютера сжатие уменьшает размер графического файла и тот может «путешествовать» через Интернет быстрее. Какой из этих форматов лучше выбрать, зависит от изображения, которое вы хотите добавить на свою страницу.

- **GIF** (Graphics Interchange Format). Такие файлы обеспечивают хорошее сжатие для изображений, в которых есть области со сплошным цветом: логотипы, текст, простые баннеры и т. д. GIF-файлы также предоставляют возможность установить один из цветов прозрачным, а это означает, что

сквозь такую область изображения будет виден фон веб-страницы. Кроме того, GIF-изображения могут включать в себя простую анимацию.

Изображения GIF содержат не более 256 оттенков, вследствие чего фотографии обычно выглядят *постеризованными* (пятнистыми и нереалистичного цвета, как плакат). Другими словами, фото заката, которое вы сделали цифровой камерой, в формате GIF будет выглядеть не очень хорошо. Если вам не нужно анимировать изображение, то формат PNG8, который мы обсудим ниже, станет лучшим выбором по сравнению с GIF.

- **JPEG** (Joint Photographic Experts Group). Такая графика имеет сильные стороны там, где у GIF наблюдаются недостатки. Изображения JPEG могут содержать миллионы различных цветов, что делает их идеальными для фотографий. Однако файлы JPEG имеют преимущества не только в плане

В КУРС ДЕЛА!

фотографий. Они способны сжимать изображения с множеством различных цветов гораздо лучше, чем GIF, потому что алгоритм сжатия JPEG предусматривает то, как человеческий глаз воспринимает смежные значения цветов. Когда графическое приложение сохраняет файл JPEG, происходит комплексный анализ цвета с целью снижения объема данных, необходимых для точного представления образа. С другой стороны, сжатие JPEG приводит к тому, что текст и большие области со сплошным цветом покрываются пятнами (*артефактами*).

- **PNG** (Portable Network Graphics) включает в себя лучшие черты GIF-и JPEG-форматов, но вы должны узнать, какую именно версию PNG использовать в конкретной ситуации. PNG8 в основном заменяет GIF. Как и GIF, он предлагает 256 цветов и базовую возможность сделать один цвет прозрачным. Тем не менее PNG8 обычно *эффективнее* сжимает изображения и делает их размер несколько меньше, чем GIF. По этой причине изображения PNG8 загружаются чуть быстрее, чем такие же изображения, сохраненные в формате GIF. То есть лучше использовать PNG8, а не GIF.

Форматы PNG24 и PNG32 (также известный как PNG24 с альфа-прозрачностью) предлагают расширенную цветовую палитру JPEG-изображений без потери качества. Это означает, что фотографии сохраняются в форматах PNG24 или PNG32 и, как правило, имеют более высокое качество, чем JPEG. Но прежде, чем перейти к формату PNG, следует помнить, что JPEG-изображения предла-

гают очень хорошее качество и меньший размер файла, чем PNG24 либо PNG32. В общем, JPEG станет гораздо лучшим выбором для фотографий и других изображений, которые состоят из множества цветов.

Однако PNG32 имеет одну особенность, которой нет у других форматов: это 256 уровней прозрачности (также называемой *альфа-прозрачностью*). Она позволяет оформить фон веб-страницы как тень или задать для графики прозрачность 50 %, чтобы можно было видеть нижележащий элемент сквозь нее, создавая эффект полупрозрачности.

- **SVG** — еще один графический формат, отличающийся от PNG, GIF и JPEG тем, что не является растровым. Вместо того чтобы размещать маленькие пиксели рядом друг с другом для формирования изображения, формат SVG (или Scalable Vector Graphics — масштабируемая векторная графика) определяет набор математических инструкций, которые описывают изображение на экране. Формат SVG, как и другие векторные форматы, не предназначен для фотографий или изображений. Вместо этого он используется для векторной графики, например логотипов и значков. Чтобы создать файл формата SVG, можно использовать программу Adobe Illustrator или аналогичный ей векторный редактор. Преимущество таких файлов заключается в том, что, как правило, при небольшом размере изображение можно масштабировать, увеличивая его размер без потери резкости и четкости. На сайте tinypurl.com/p7w6zkd приведен длинный список ресурсов, посвященных формату SVG.

В приведенных примерах путь не заключен в кавычки, как требует HTML; они необязательны, хотя и могут указываться. В CSS все три следующие строки кода функционируют идентично:

```
background-image: url(images/bg.png);
background-image: url("images/bg.png");
background-image: url('images/bg.png');
```

ПРИМЕЧАНИЕ

Файлы формата SVG можно использовать в качестве фоновых изображений во всех браузерах, за исключением Internet Explorer 8. Сохраните файл формата SVG на своем сайте и ссылайтесь на него, как на любой другой графический файл:

```
background-image: url(images/icon.svg);
```

Управление повтором фоновых изображений

Если свойство `background-image` используется без указания дополнительных атрибутов, то фоновый рисунок многократно повторяется в виде мозаики, заполняя всю веб-страницу. На этот случай можно воспользоваться свойством `background-repeat`, чтобы определить, каким образом будет повторяться фоновый рисунок:

`background-repeat: no-repeat;`

Свойство может принимать четыре значения. Рассмотрим их по порядку.

- `repeat` — параметр по умолчанию, обеспечивает повторное отображение фонового рисунка слева направо и сверху вниз до полного заполнения всего пространства веб-страницы (рис. 8.3).



Рис. 8.3. Применяйте мозаичное отображение фоновых рисунков с осторожностью: выбирайте не слишком контрастные, удачно стыкующиеся изображения (*слева*); четкие, яркие, высококонтрастные изображения (*справа*) делают основной текст веб-страниц нечитаемым

- `no-repeat` — отображает фоновый рисунок один раз, без повторения и мозаики. Этот параметр используется на практике довольно часто, и мы также будем применять его для установки фоновых изображений элементов, отличных от `body`. Вы можете пользоваться им, чтобы поместить графический логотип в верхней части веб-страницы или создать собственные маркеры в списках (пример такого маркера приведен в практикуме этой главы).
- `repeat-x` — вызывает повторение фонового изображения горизонтально вдоль оси X (по всей ширине веб-страницы). Это замечательное средство для добавления графического баннера (заголовка) в верхней части веб-страницы (*рис. 8.4, слева*), декоративной границы сверху или снизу заголовка.

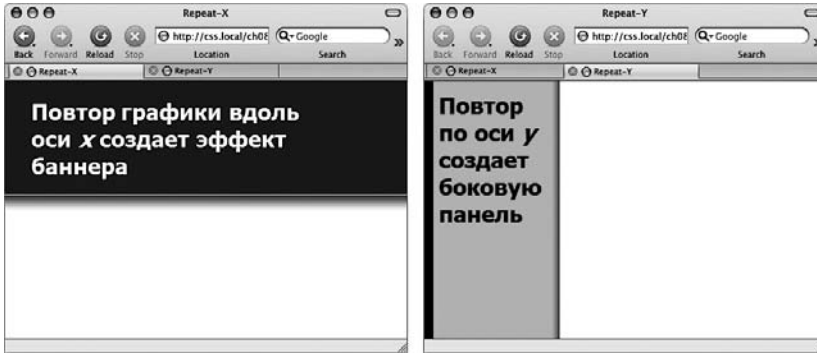


Рис. 8.4. Управляя повтором фоновых изображений, можно создать графический фон для заголовков (слева) и боковых панелей (справа) веб-страницы

- `repeat-y` — повторяет фоновое изображение вертикально вдоль оси Y (по всей высоте веб-страницы). Вы можете использовать этот параметр, чтобы добавить слева или справа веб-страницы графическую боковую панель (см. рис. 8.4, *справа*) или создать эффект тени с любой стороны элемента веб-страницы (возможно, той же боковой панели).
- `round` — мозаично повторяет фоновое изображение, как и предыдущие параметры, но никогда не обрезает его. Изображения масштабируются, чтобы заполнить область по вертикали и горизонтали без отсечения (рис. 8.5, *слева*). Для достижения указанного эффекта браузеры искажают копии изображений, нарушая исходные пропорции изображения.
- `space` — мозаично повторяет фоновое изображение, как и предыдущие параметры, но предотвращает искажение и обрезание изображений сверху или снизу. Другими словами, параметр `space` всегда отображает изображение без искажений. Соответственно, браузеры добавляют пространство между копиями изображения (см. рис. 8.5, *справа*).

Позиционирование фоновых изображений

Размещение фоновых изображений и управление их повтором — это только половина дела. Свойство позиционирования фонового изображения `background-position` позволяет несколькими способами управлять точным расположением графики. Вы можете определить начальную позицию фонового изображения по горизонтальной и вертикальной координате посредством трех ключевых слов, точных абсолютных и процентных значений.

Ключевые слова

Вы можете работать с двумя наборами ключевых слов: один из них имеет три параметра управления горизонтальным позиционированием: `left`, `center`, `right`, а другой — три параметра вертикального: `top`, `center`, `bottom` (рис. 8.6). Предположим, вы

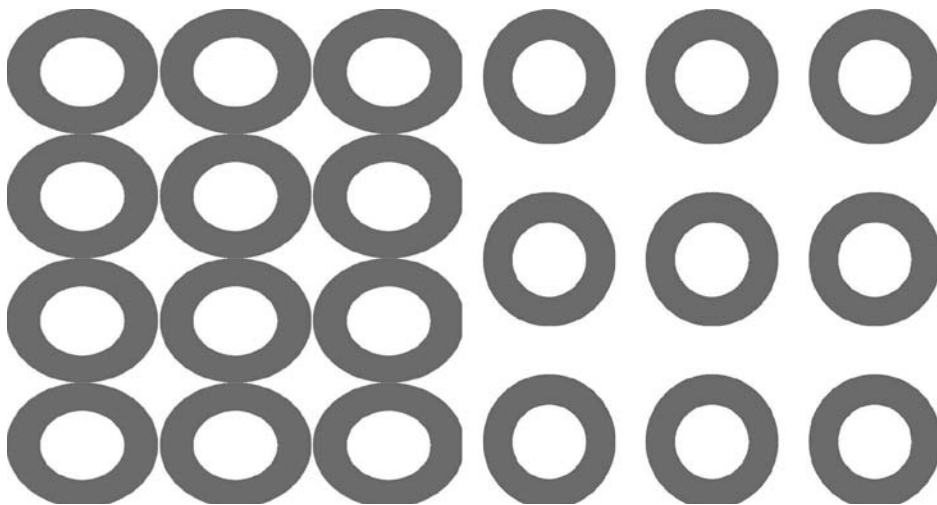


Рис. 8.5. Параметры `round` (слева) и `space` (справа) для узора `background-repeat` гарантируют, что ни одна из копий изображения не будет обрезана. Эта комбинация параметров заставляет браузер искажать пропорции изображения (слева) или добавлять дополнительное пространство между его копиями (справа)

хотите поместить рисунок прямо в центре веб-страницы. Для этого можно создать следующий стиль:

```
body {
  background-image: url(bg_page.jpg);
  background-repeat: no-repeat;
  background-position: center center;
}
```

В КУРС ДЕЛА!

Типы URL-адресов

В языке CSS при добавлении фонового изображения вы должны определить URL-адрес (*Uniform Resource Locator* — унифицированный указатель ресурса) — путь к файлу, расположенному в Интернете. Существует три типа путей: *абсолютный*, *корневой относительный* и *относительный от документа*. Все три варианта указывают, где браузер может найти определенный файл (например, другую веб-страницу, рисунок или внешнюю таблицу стилей).

Абсолютный путь похож на почтовый адрес — он содержит всю информацию, необходимую браузеру для нахождения файла. Абсолютный путь включает `http://`, имя компьютера (хоста) в Сети, папку и имя самого фай-

ла. Например: `http://www.cosmofarmer.com/images/bluegrass.jpg`.

Корневой относительный путь (относительный путь от корневого каталога) указывает, где находится файл самого верхнего уровня — корневой папки сайта. Он не содержит ни `http://`, ни доменного имени. Начинается с символа `/`, указывающего на корневой каталог сайта (папка, в которой располагается главная страница). Например, `/images/bluegrass.jpg` обозначает, что файл `bluegrass.jpg` находится в папке `images`, которая расположена в корневом каталоге. Самый простой способ определить корневой относительный путь — взять абсолютный и отбросить `http://` и имя хоста.

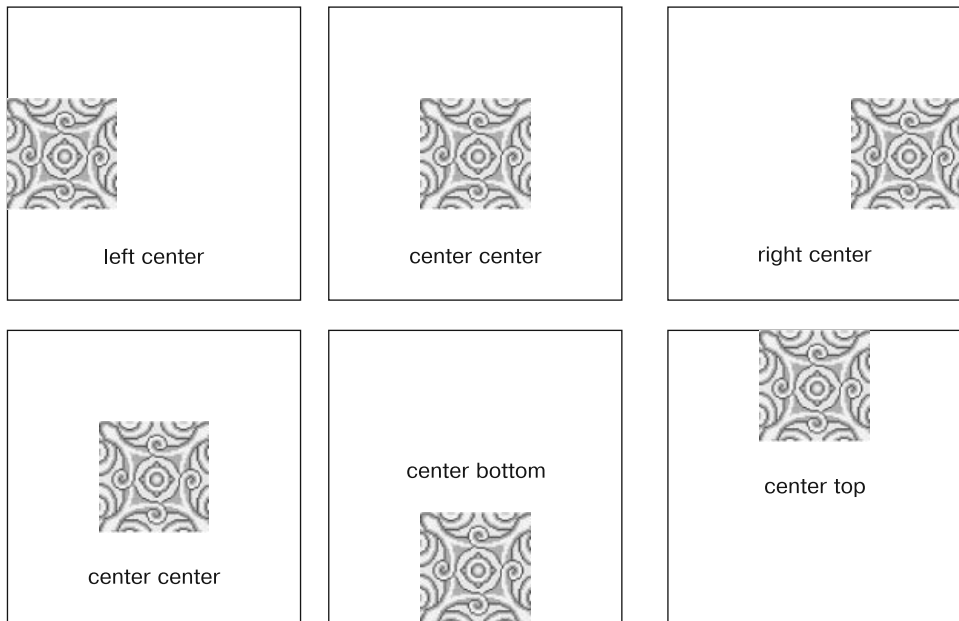


Рис. 8.6. Вы можете использовать для позиционирования фоновых изображений ключевые слова, последовательность написания которых не имеет значения. Записи `top center` или `center top` идентичны

В КУРС ДЕЛА!

Относительный от документа определяет путь к файлу от текущего документа. Когда он указан в таблице стилей, он определяет путь *до указанного файла от таблицы стилей*, а не от текущей веб-страницы.

Ниже приведено несколько подсказок относительно того, какой тип URL-адреса нужно использовать в каждом конкретном случае.

- Если вы обращаетесь к файлу, который находится на сервере, отличном от того, где размещена ваша таблица стилей, то должны использовать абсолютный путь. Это единственный тип URL-адреса, который может указывать на другой сайт.
- Корневой относительный путь хорош для доступа к изображениям, хранящимся на вашем собственном сайте. Поскольку отсчет всегда начинается с корневого каталога, вы можете перемещать таблицу стилей в любое другое место, не затрагивая при этом пути от корневого каталога до изображения сайта. Однако этим способом затруднительно

пользоваться, если вы только учитесь веб-дизайну: вы не сможете предварительно просмотреть страницу в браузере, пока не будете просматривать страницу с сервера в Интернете или установленного на компьютере для тестирования и отладки. Другими словами, если вы попытаетесь открыть веб-страницу на своем компьютере с помощью меню **Файл** ▶ **Открыть** (**File** ▶ **Open**), то вообще не увидите никаких изображений, которые помещены на веб-страницу с применением относительного пути от корневого каталога.

- Использование относительного пути от документа — лучший способ разработки веб-дизайна на собственном компьютере, без привлечения веб-сервера. Вы можете создать CSS-файлы и затем просмотреть их в браузере, открывая страницу, сохраненную на жестком диске компьютера. Они будут работать и тогда, когда вы загрузите их на рабочий сайт, но вам придется переписать URL к изображениям, если вы захотите переместить таблицу стилей в другое расположение на сервере.

Чтобы переместить изображение в верхний правый угол веб-страницы, измените параметр позиции фонового рисунка:

```
background-position: right top;
```

ПРИМЕЧАНИЕ

Если вы решили применить повторяющееся фоновое изображение (установив для свойства `background-repeat` одно из значений, описанных ранее), то начальную точку или координату первого отображаемого фонового рисунка определит свойство `background-position`. Например, если вы примените параметр `repeat`, то весь фон веб-страницы будет заполнен фоновым изображением. Но именно `background-position` укажет, с какой позиции нужно начать повторное отображение рисунка.

Ключевые слова полезны, когда необходимо создать вертикальные или горизонтальные баннеры-заголовки. Если вы хотите, чтобы фоновый рисунок был горизонтально центрирован и повторялся от верхнего до нижнего края веб-страницы, создавая фон для всего содержимого (рис. 8.7, *слева*), то можно создать следующий стиль:

```
body {  
  background-image: url(background.jpg);  
  background-repeat: repeat-y;  
  background-position: center top;  
}
```

На рис. 8.7 на левом скриншоте фоновое изображение представляет собой широкий белый блок с тенями с левой и правой сторон. Цвет фона веб-страницы — серый, и текст выглядит так, как будто он написан на белом листе бумаги, помещенном на экран компьютера.

Аналогично, применяя ключевые слова `bottom`, `center` и `top`, вы можете установить горизонтальное повторение фонового изображения в определенном месте веб-страницы (или внутри форматизируемого элемента) при использовании параметра `repeat-x` (см. рис. 8.4, *слева*).

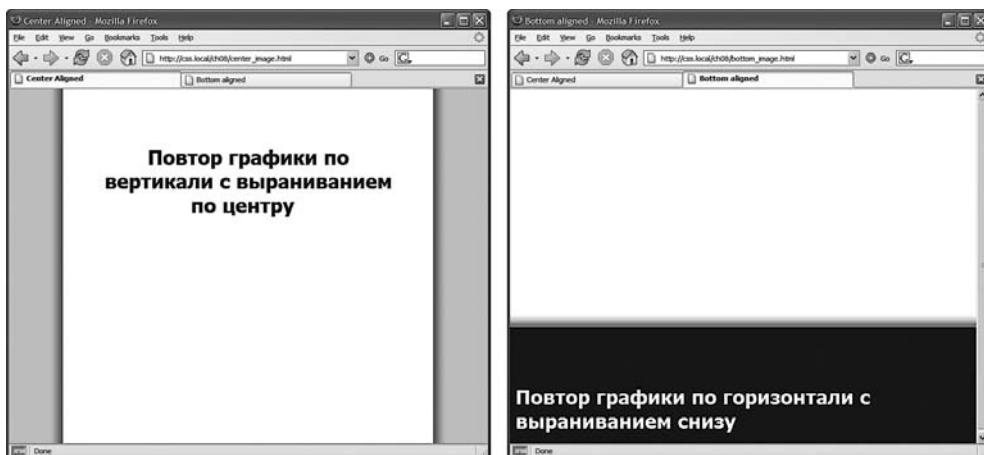


Рис. 8.7. При повторении фонового изображения вертикально (*слева*) или горизонтально (*справа*) пользуйтесь свойством `background-position`

СОВЕТ

Вы можете добавить фоновый рисунок к обоим элементам: и к `html`, и к `body`. Если вы повторяете оба изображения по горизонтали и располагаете рисунок элемента `body` вверху, а изображение элемента `html` внизу, то можете достичь эффекта двух полос, идущих через верхнюю и нижнюю части страницы, вне зависимости от высоты страницы.

ОШИБКИ БРАУЗЕРОВ

Проблема с отображением фоновых рисунков в нижней части окон браузеров

Отображая рисунок в качестве фона всей веб-страницы, большинство дизайнеров часто неправильно устанавливают вертикальное позиционирование изображения. Например, если вы используете вертикальную позицию `bottom`, изображение не всегда появляется в нижней части окна браузера. Это случается, когда основное содержимое веб-страницы меньше по высоте.

Если веб-страница имеет всего несколько абзацев текста и отображается на большом мониторе, то многие браузеры принимают за нижний край окна браузера нижний край последнего абзаца текста веб-страницы. Если вы столкнетесь с такой ситуацией, добавьте следующий стиль:

```
html { height: 100%; }
```

Точные значения

Позиционировать фоновые изображения можно, используя точные значения в пикселах или единицах `em`. При этом нужно указать два значения: одно из них определяет расстояние между левым краем изображения и левым краем элемента-контейнера, а другое — расстояние между верхним краем изображения и верхним краем элемента-контейнера (другими словами, первое значение указывает горизонтальную координату, а второе — вертикальную).

Допустим, вы хотите установить собственные маркеры для списка. При добавлении в элемент `li` фонового рисунка маркеры не всегда будут правильно выровнены (рис. 8.8, *вверху*). Придется выравнять их с помощью свойства `background-position` (см. рис. 8.8, *внизу*). В нашем случае элементы списка будут выглядеть гораздо симпатичнее, если расположить маркеры на 5 пикселей правее и на 8 пикселей ниже. Для этого добавим в стиль фонового изображения следующее свойство:

```
background-position: 5px 8px;
```

Вы не сможете указать расстояние относительно нижнего (`bottom`) или правого (`right`) края веб-страницы или формируемого элемента в пикселах или `em`, поэтому, если хотите быть уверенными в том, что изображение помещено точно в правый нижний угол, используйте в качестве единиц измерения либо ключевые слова (`bottom right`), либо процентные значения, как описывается далее. Однако вы можете использовать отрицательные значения, чтобы сместить изображение относительно левого края или поднять относительно верхнего, скрывая часть. Возможно, вы захотите применить значения для обрезки части изображения. Или, если у фонового изображения имеется большое пустое поле с левой или верхней стороны, можно использовать отрицательные значения для его устранения.

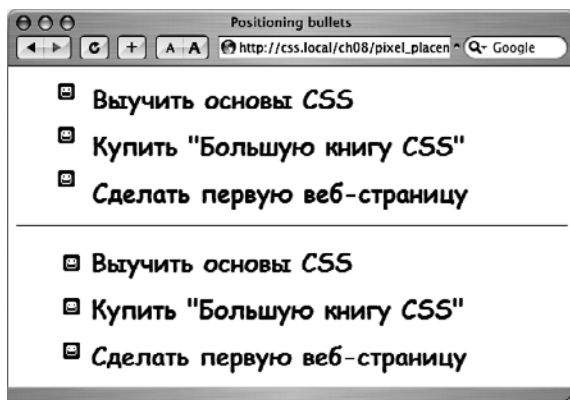


Рис. 8.8. Использование собственных изображений маркеров иногда требует их точного позиционирования так, чтобы они имели соответствующее центрирование и правильно отображались относительно текста элементов списка

Процентные значения

Для позиционирования фоновых изображений вы также можете применять процентные значения и тогда добьетесь интересного эффекта. В данном случае можно позиционировать элемент на расстояние, пропорциональное его ширине. Например, вам может понадобиться переместить рисунок на расстояние в три четверти размера элемента от заголовка, но вы не знаете ширины.

Как и в случае с пикселями или значениями в единицах `em`, вы должны указать два процентных значения: одно представляет собой горизонтальную координату, а второе — вертикальную. Относительно чего рассчитывается значение, указанное в процентах? Если объяснить в двух словах, то оно приравнивается к процентному значению форматируемого элемента.

Лучший способ понять — рассмотреть несколько практических примеров. Чтобы позиционировать изображение в центре веб-страницы (аналогично изображению в центре на рис. 8.8), необходимо использовать следующий код:

```
background-position:50% 50%;
```

Свойство устанавливает координату x таким образом, что она указывает на точку изображения в 50 % от его левой стороны, которая находится в 50 % от левого края веб-страницы (или любого элемента, к которому применяется фоновое изображение). Координата y устанавливается так, что она указывает на точку изображения в 50 % от верхней стороны, которая находится в 50 % от соответствующего края веб-страницы или форматируемого элемента. Другими словами, центр изображения совпадает с центром элемента, для которого оно определено в качестве фонового. Это означает, что при использовании процентных значений точные координаты позиционирования динамически изменяются и похожи на «движущуюся мишень» (поэтому координаты позиционирования форматируемого элемента в процентах могут изменяться, если посетители веб-страницы меняют размеры окна браузера).

При использовании процентных значений в первую очередь необходимо определить *точку привязки* (координату) — позицию изображения, которую вы хотите закрепить. В этом примере отметки «50% 50%» на центральном изображении отображают его точку привязки. Далее определите точку 50/50 на самой странице (это должен быть центр страницы). Это место, в которое помещается точка привязки изображения. Три остальных изображения позиционируются аналогичным образом.

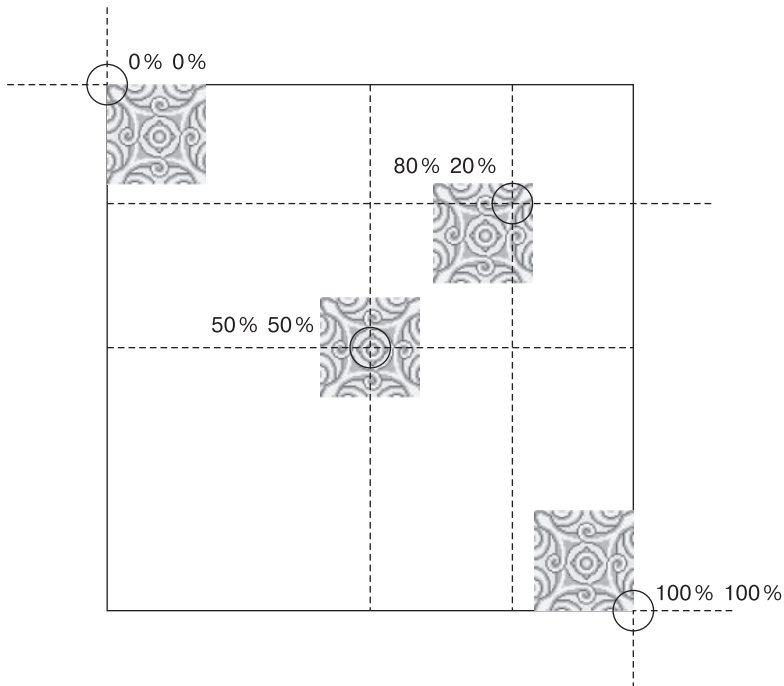


Рис. 8.9. Каждое из этих четырех изображений было позиционировано на странице (представленной большой черной рамкой) с помощью процентных значений

ПРИМЕЧАНИЕ

Позиционирование изображения по вертикали в фоне страницы с помощью процентов необязательно поместит его в правильное место, если содержимое страницы не заполняет всю высоту окна браузера.

Как и в случае с пикселями и `em`, можно назначать отрицательные процентные значения, но результаты трудно предугадать. Вы можете также комбинировать и подбирать значения в пикселях, `em` и процентах одновременно. Например, чтобы поместить изображение на расстоянии 5 пикселей от левого края элемента, но в центре элемента по высоте, применяют свойство со следующими параметрами:

```
background-position: 5px 50%;
```

ПРИМЕЧАНИЕ

Безусловно, фоновые изображения улучшают визуальное восприятие веб-страниц, но они, как правило, не отображаются при распечатке. В большинстве браузеров фоновые изображения могут распечатываться, но по умолчанию это требует дополнительной настройки параметров вывода на печать со стороны пользователя. Если вы хотите предоставить возможность посетителям сайта печатать веб-страницы с рисунками в таком виде, как они отображаются в браузере, то вместо фоновых изображений используйте элемент `img` для вставки таких важных фрагментов, как логотип сайта или схема проезда к интернет-магазину.

Фиксация позиции изображения

Когда посетитель прокручивает содержимое веб-страницы, чтобы увидеть остальную ее часть, фоновое изображение прокручивается вместе с содержимым. В результате рисунок на заднем плане перемещается вместе с текстом. Кроме того, если он не повторяется, то в процессе прокрутки исчезнет из виду. Когда вы устанавливаете в качестве фонового изображения веб-страницы логотип сайта или водяной знак, вы, вероятно, хотите, чтобы он всегда *оставался* в пределах видимости во время просмотра.

В CSS такая проблема решается с помощью свойства `background-attachment`, которое может принимать два значения: `scroll` и `fixed`. Значение по умолчанию, `scroll`, определяет такое поведение браузера, при котором фоновое изображение прокручивается вместе с текстом и другим контентом. Значение `fixed` предотвращает перемещение, жестко фиксируя его на заднем плане (рис. 8.10). Так, если вы хотите поместить логотип компании в левом верхнем углу веб-страницы и зафиксировать его там даже при прокрутке посетителями контента, можно создать следующий стиль:

```
body {  
  background-image: url(images/logo.gif);  
  background-repeat: no-repeat;  
  background-attachment: fixed;  
}
```

Определение начальной позиции фонового изображения и настройка отсечения

В CSS есть возможность сообщить браузеру, где фоновое изображение должно начаться по отношению к границам, отступам и содержимому элемента. Например, когда изображение используется для повторяющегося заполнения, оно появляется в верхнем левом углу отступа элемента (верхнее среднее изображение на рис. 8.11). Но начальную позицию изображения можно изменить, воспользовавшись свойством `background-origin`. Этому свойству присваивается одно из трех значений.

- `border-box` — изображение помещается в верхний левый угол области границы (см. рис. 8.11, *слева вверху*).
- `padding-box` — изображение помещается в верхний левый угол области отступа (см. рис. 8.11, *посередине вверху*). Это обычное место, куда браузер помещает фоновое изображение.

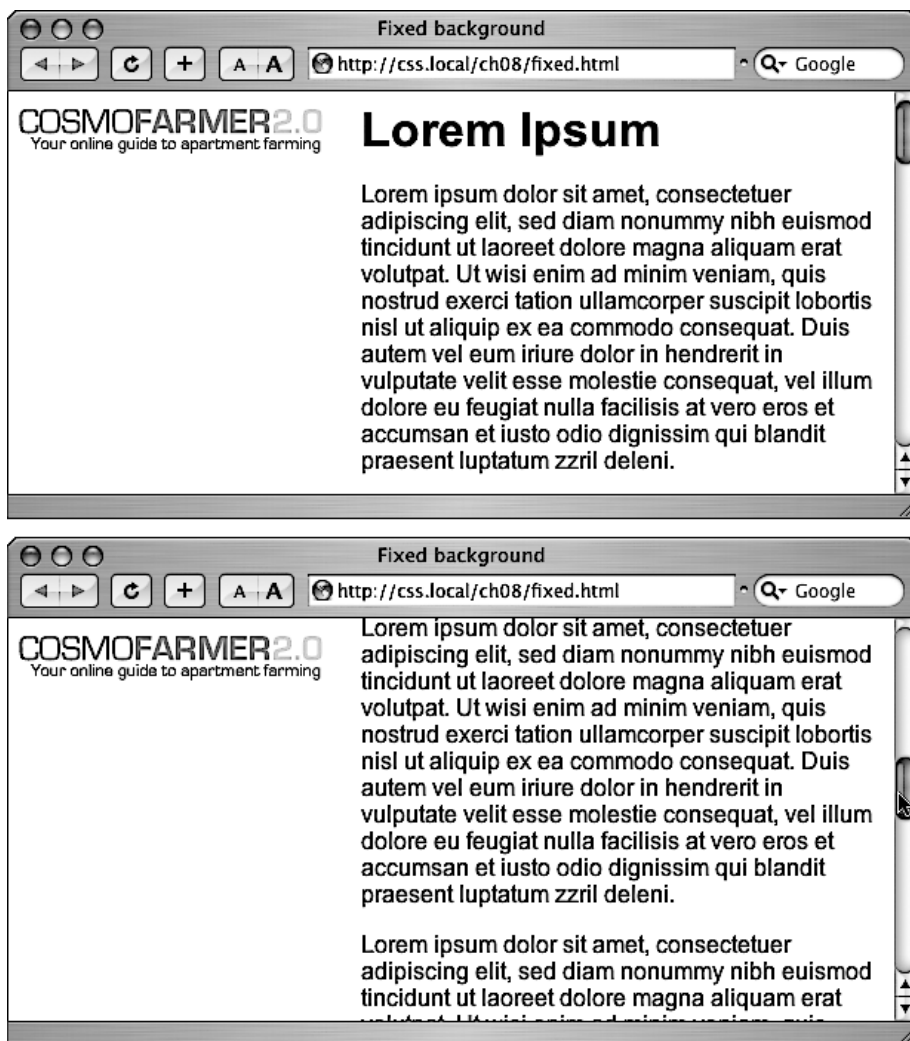


Рис. 8.10. Ищете способ закрепить логотип сайта (в примере изображение CosmoFarmer 2.0), чтобы он при прокрутке страницы остался на месте? Этого можно добиться с помощью свойства `background-attachment` и значения `fixed`

- `content-box` — изображение помещается в левый верхний угол области содержимого (см. рис. 8.11, *справа сверху*).

Разумеется, эти настройки не имеют смысла, если вокруг элемента нет ни отступа, ни границы. Кроме того, эффект может быть практически незаметен, особенно если у заполняющего повторяющегося изображения не видно швов.

ПРИМЕЧАНИЕ

Свойства `background-origin` и `background-clip` в Internet Explorer 8 и его более ранних версиях не работают.

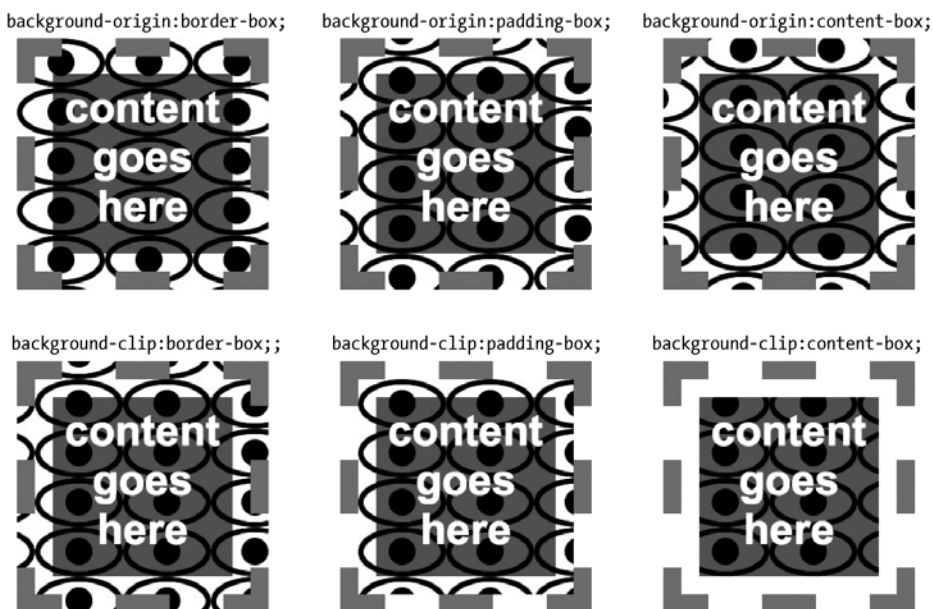


Рис. 8.11. Свойство `background-clip` можно также применить к фоновому цвету. Если добавить прерывистую границу (точечную, пунктирную или двойную), фоновый цвет обычно появляется позади линий границы в пустых областях. Но чтобы ограничить отображение цвета, не позволяя ему пересекать пределы границ, нужно присвоить свойству `background-clip` значение `padding-box`

Но эффект может быть весьма заметен, если изображение не повторяется. Например, если вы выберете настройку `no-repeat` и фоновое изображение появится только один раз, его можно поместить в верхний левый угол области (внутри любых отступов):

```
background-image: url(logo.png);
background-repeat: no-repeat;
background-origin: content-box;
```

Кроме того, свойство `background-origin` может эффективно использоваться с другим свойством — `background-clip`, которое ограничивает область появления фонового изображения. Обычно фоновые изображения заполняют *всю* область элемента, включая пространство позади границ и отступов (см. рис. 8.11, *слева внизу*). Но область отображения рисунка можно ограничить с помощью трех значений.

- `border-box` — позволяет рисунку отображаться позади содержимого и любых границ. Заметить это можно только при использовании линий границ с промежутками, например пунктирных (см. рис. 8.11, *слева внизу*). Это обычное поведение браузеров, поэтому использовать эту настройку нет смысла.
- `padding-box` — дает возможность ограничить любое фоновое изображение областью отступов и содержимого элемента (см. рис. 8.11, *посередине внизу*). Эта

настройка пригодится при использовании линий границ с промежутками, например пунктирных, и нежелании видеть рисунок в разрывах границы.

- `content-box` — позволяет ограничить фоновое изображение областью содержимого элемента (см. рис. 8.11, *справа внизу*).

Сочетая свойства `background-origin` и `background-clip`, можно отобразить рисунок в верхнем левом углу области содержимого и позади только содержимого:

```
background-origin: content-box;  
background-clip: content-box;
```

Масштабирование фоновых изображений

Обычно изображение, помещенное в качестве фона элемента страницы, принимает размер, который был задан при его создании. Однако свойство `background-size` позволяет управлять размером фонового рисунка. Чтобы установить размер, можно использовать как точные значения, так и ключевые слова.

- Для того чтобы установить размер изображения, нужно задать его высоту и ширину. Для этого можно воспользоваться абсолютными значениями, выраженными в пикселах:

```
background-size: 100px 200px;
```

Этот код устанавливает для фонового изображения ширину 100 пикселей и высоту 200 пикселей. Можно также установить только значения ширины или высоты, задав для оставшегося значения ключевое слово `auto`:

```
background-size: 100px auto;
```

В таком случае фоновое изображение будет иметь ширину 100 пикселей, а браузер автоматически установит его высоту, сохраняя пропорции (чтобы не возникло искажений).

Можно также использовать процентные значения. Если нужно масштабировать рисунок для полноценного заполнения фона, можно для обеих настроек использовать значения 100% (рис. 8.12, *слева*):

```
background-size: 100% 100%;
```

- Ключевое слово `contain` приводит к изменению размеров изображения с целью заполнения фона элемента с сохранением пропорций (см. рис. 8.12, *посередине*). В зависимости от формы изображения и элемента рисунок увеличивается, чтобы поместиться либо по ширине, либо по высоте элемента.

```
background-size: contain;
```

- Ключевое слово `cover` заполняет изображением фон элемента, соотнося высоту/ширину элемента и изображения, не искажая пропорции (см. рис. 8.12, *справа*).

```
background-size: cover;
```

Использование свойства `background-size` практически всегда приводит к изменению размеров исходного изображения: если изображение меньше заполняемого

элемента, браузер увеличивает его масштаб, что зачастую выражается заметной пикселизацией и ухудшением качества изображения (что видно по фоновым изображениям на рис. 8.12).



Рис. 8.12. Только свойство `background-size` позволяет менять размеры фонового изображения. Его поддерживает большинство браузеров, за исключением Internet Explorer 8, поэтому используйте свойство с оглядкой

СОВЕТ

Свойство `background-size` может оказаться особенно удобным при работе с элементами, размер которых определяется с использованием процентных значений, например при верстке адаптивных дизайнов, рассматриваемых в главе 14. Допустим, если поместить изображение в фон баннера, занимающего 960 пикселей при просмотре на мониторе настольного компьютера, но сжать его до 480 пикселей при просмотре на смартфоне, можно поместить в баннер большое изображение и воспользоваться следующей настройкой:

```
background-size: 100% auto;
```

Эта настройка инструктирует браузер изменить размер изображения таким образом, чтобы оно поместилось в сжатый баннер.

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Добавление специальных эффектов к фоновым изображениям

Если у вас есть опыт работы в таких графических редакторах, как программа Photoshop, вы, вероятно, знакомы с режимами наложения, которые позволяют преобразовать два изображения в одно, заставив их взаимодействовать визуально. Рассмотрим пример, как это работает. Представьте, что вы поместили в фоновый элемент фотографию, а также задали для нее фоновый цвет. Как правило, любое изображение, не имеющее прозрачных областей, перекроет фоновый цвет, расположенный позади него.

Но как быть, если вы хотите смешать изображение и фоновый цвет? Например, вы хотели бы, чтобы фоновый цвет был виден *только* сквозь белые области фотогра-

фии. Относительно новое свойство `background-blend-mode` предоставляет такую возможность. Необходимо применить это свойство к элементу с фоновым цветом и изображением. Оно использует один из 16 режимов наложения на выбор, каждый из которых позволяет добиться различных результатов.

Еще один пример. Предположим, вы поместили фотографию помидора в качестве фонового объекта элемента `div` и хотите добавить оранжевый цвет, чтобы смешать его с фотографией. В данном случае необходимо установить оранжевый цвет в качестве фонового, а затем задать режим наложения следующим образом:

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

```
background-image: url(tomato.png);
background-color: orange;
background-blend-mode: screen;
```

Режим наложения можно применять к нескольким изображениям (см. раздел «Использование множественных фоновых изображений» главы 8). Это позволит создать очень интересный визуальный эффект.

По адресу tinyurl.com/nllpow7 опубликован видеоролик использования режима наложения в Photoshop. Свойство `background-blend-mode` в CSS работает так же.

По адресу tinyurl.com/mnm6kzn есть пример использования режима наложения в CSS (сейчас ни одна версия Internet Explorer (а также Edge) не поддерживает эту функцию).

Сокращенная запись свойства background

Как вы видите из примеров предыдущих разделов, чтобы воспользоваться всеми преимуществами фоновых изображений, необходимо применять свойства, управляющие параметрами фоновых изображений. Но многократный повторный набор таких длинных свойств, как `background-image`, `background-attachment` и т. д., отнимает много времени. Существует более простой метод — применение сокращенного свойства `background`.

Фактически вы можете объединить и перечислить все свойства фона (включая `background-color`) в единственной строке CSS-кода. Используйте свойство `background`, за которым должны следовать значения свойств `background-image`, `background-position`, `background-size`, `background-repeat`, `background-origin`, `background-clip`, `background-attachment` и `background-color`.

ПРИМЕЧАНИЕ

Браузер Internet Explorer 8 не поддерживает свойства `background-size`, `background-origin` и `background-clip`. Если вы используете их в сокращенном свойстве `background`, IE 8 не будет применять их к элементу, проигнорировав.

Следующий стиль добавляет только одно изображение (`no-repeat`) по центру веб-страницы, уменьшает его размер до 50 %, фиксирует изображение (чтобы оно не прокручивалось вместе со страницей) и устанавливает фон белого цвета:

```
body {
  background: url(bullseye.gif) center center / 50% no-repeat fixed #FFF;
}
```

Если вы указываете положение (например, в предыдущем примере это `center center`), свойство `background-size` (значение которого в примере равно 50%), используйте между ними разделитель `/`.

Вовсе не обязательно указывать абсолютно все параметры свойства. Вы можете использовать один из них или любое сочетание. Например, `background: yellow` равнозначно `background-color: yellow`. Все те параметры свойства, которые вы не определите сами, будут иметь стандартные значения по умолчанию. Допустим, вы определили только само фоновое изображение:

```
background: url(image/bullseye.gif);
```


Этот код эквивалентен следующему стилю:

```
background: url(image/bullseye.gif) left top / 100% repeat scroll border-box border-box transparent;
```

Поведение, когда при отсутствии определения значения происходит возвращение к значениям по умолчанию, может привести к весьма неожиданным результатам. Предположим, что к стилю добавляются два объявления:

```
background-color: yellow;  
background: url(image/bullseye.gif) no-repeat;
```

Возможно, вы ожидали увидеть GIF-изображение на желтом фоне. Но вы его не увидите, потому что при обнаружении свойства `background` без указания цвета браузер присваивает свойству `background-color` значение `transparent` (невидимый). Чтобы выйти из этой сложной ситуации, нужно указать свойство `background-color` вторым:

```
background: url(image/bullseye.gif) no-repeat;  
background-color: yellow;
```

Кроме того, когда к одному и тому же элементу применяется сразу несколько свойств, можно в конечном итоге случайно стереть фоновые изображения. Предположим, необходимо добавить фоновое изображение к каждому абзацу страницы, для чего создается следующий стиль:

```
p {  
  background: url(icon.png) left top no-repeat rgb(0.30.0);  
}
```

Затем принимается решение, что после заголовка второго уровня каждый первый абзац должен иметь синий фон, и создается следующий стиль:

```
h2 + p {  
  background: blue;  
}
```

В этом втором стиле используется сокращенная запись, сбрасывающая все остальные свойства фона к их значениям по умолчанию. Что касается изображения, то по умолчанию оно вообще не используется, поэтому вместо простого добавления к абзацу синего фона с оставлением изображения без изменений этот стиль вообще удаляет!

Иначе говоря, сокращенная запись свойства `background` может сэкономить время на написании кода, но может и принести проблемы, о чем не нужно забывать.

ЧАВО

Поиск бесплатных коллекций изображений

Я не художник, не умею рисовать, и у меня нет даже цифровой камеры. Где же можно найти графический материал для сайта?

Нужно отдать должное Всемирной паутине. Она представляет собой универсальное средство поиска, благодаря которому можно выйти из любой ситуации.

ЧАВО

Существует множество платных сайтов, содержащих коллекции готовых фотографий и иллюстраций, но наряду с ними также имеется довольно много бесплатного материала.

Ресурс foter.com содержит свыше 228 миллионов бесплатных фотографий. Он собирает лицензионные фотографии фонда Creative Commons из всех уголков Всемирной паутины. Ресурс также содержит бесплатный WordPress-плагин, с помощью которого можно легко добавлять стоковые изображения для вашего блога. На сервисе unsplash.com вы найдете потрясающие бесплатные фотографии природы, архитектуры и животных.

Образцы фотографий можно найти на сайте morguefile.com, где есть множество замечательных снимков, сделанных людьми, любящими природу. Сайт freeimages.com — еще один отличный фоторесурс. Сайт openphoto.net предлагает коллекции изображений, предоставляя возможность их применения с некоторыми ограничениями. Вы также можете воспользоваться поиском иллюстраций с лицензией Creative Commons и найти изображения (а также видео и музы-

ку), которые могут быть задействованы в личных или коммерческих проектах: search.creativecommons.org. Кроме того, вы можете искать иллюстрации с лицензией Creative Commons на сайте flickr.com/creativecommons (надо отметить, что за фотографии на бесплатных сайтах не нужно платить, но не все они могут использоваться в коммерческих проектах; прежде всего читайте комментарии мелким шрифтом ко всем снимкам, с которыми собираетесь работать).

Если вы ищете маркеры для списков, значки для панели навигации, файлы фоновых узоров для заполнения страницы, то вам доступно несколько сайтов. Ресурс flaticon.com предоставляет доступ к огромной базе бесплатных векторных знаков, и вы можете использовать их и в виде PNG- и SVG-файлов на своем сайте. Сайт tinyurl.com/k9zfhfz предлагает бесплатный набор из 121 значка. Если вас интересуют мозаичные фоновые узоры, посетите один из следующих сайтов: tinyurl.com/5kwqqd, tinyurl.com/6dva2f или tinyurl.com/6dva2f. Можете также создать собственный мозаичный фон с помощью следующих инструментов: bgpatterns.com, stripegenerator.com или patterncooler.com.

Использование множественных фоновых изображений

Хотя обычно достаточно и одного фоновое изображения, доступна возможность размещения сразу нескольких изображений в виде слоев. Предположим, что нужно добавить фоновое изображение к боковой панели, чтобы придать ей видимость свитка (рис. 8.13). Если поместить в качестве фона одно изображение, то сначала оно может работать (см. рис. 8.13, *слева вверху*), но если добавить к боковой панели слишком много текста, внешний вид ухудшится (см. рис. 8.13, *справа вверху*). Причина в том, что у изображения один размер и оно не будет становиться больше или меньше, чтобы подстроиться под размер боковой панели.

На этот случай вы можете добавить к фону элемента несколько изображений. В случае со свитком можно использовать три фоновых изображения: одно для верхней части свитка, одно для нижней и одно для его текстовой области. Последнее изображение представляет собой бесшовную плитку, поэтому при увеличении высоты боковой панели изображение заполняет новое пустое пространство плитками.

Разумеется, несколько изображений можно использовать и для решения менее сложных задач. Предположим, что к фону элемента нужно добавить красочное текстурированное изображение, а также двухцветный логотип. Как сказано во врез-

ке из раздела «Добавление фоновых изображений», для многоцветного изображения больше подходит формат JPEG, а для областей, содержащих сплошные цвета, к которым можно отнести логотип, больше подходит формат PNG8. В данном случае можно объединить преимущества, используя JPEG для многоцветного фона и PNG8 для логотипа.

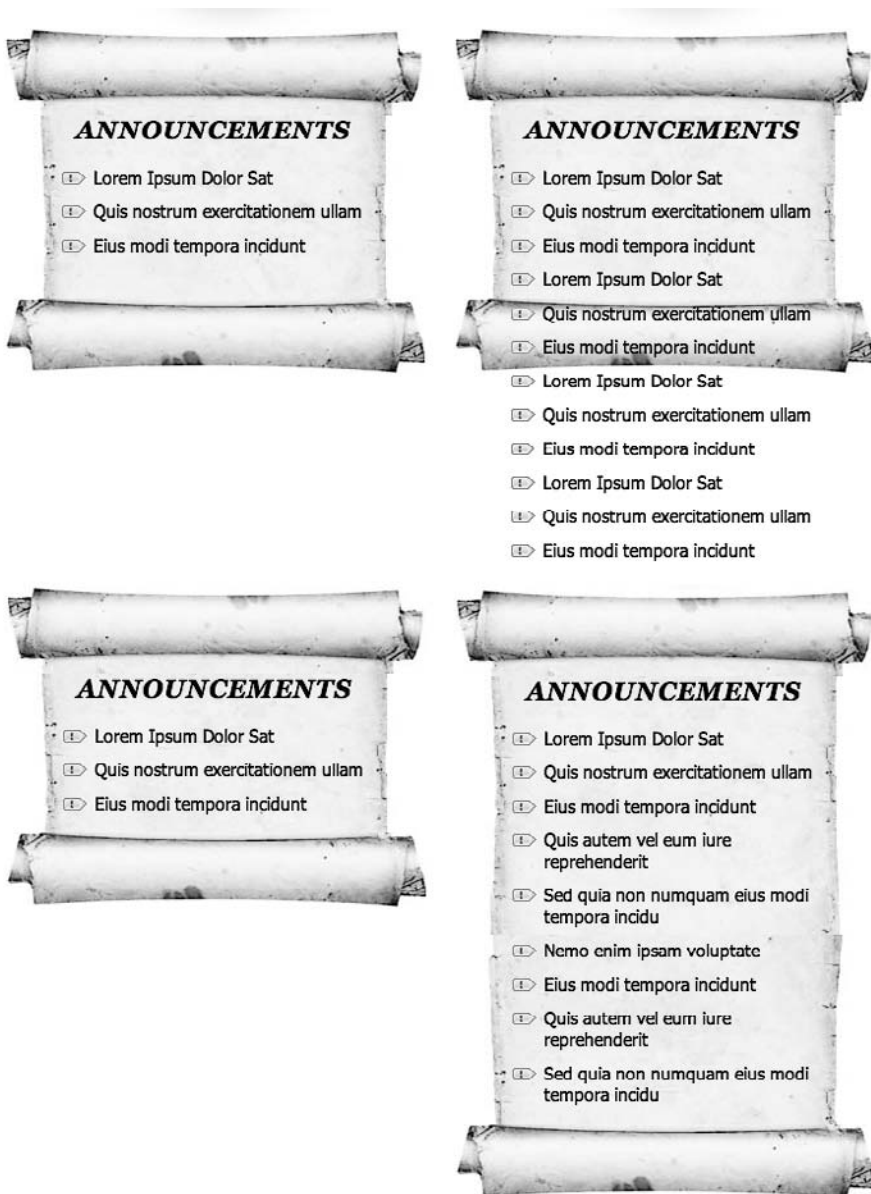


Рис. 8.13. Учтите, что такой способ создания нескольких фоновых изображений не работает в Internet Explorer 8

Для создания нескольких фоновых изображений нужно воспользоваться свойством `background-image` со списком изображений с запятой в качестве разделителя:

```
background-image: url(scrollTop.jpg),  
                 url(scrollBottom.jpg),  
                 url(scrollMiddle.jpg);
```

ПРИМЕЧАНИЕ

В данном примере кода каждый URL-адрес находится на отдельной строке, но это соблюдать не обязательно. Все это можно набрать и одной (но очень длинной) строкой. Однако многие разработчики считают, что код будет легче читаться, если ссылку на каждое изображение поместить в отдельной строке и воспользоваться пробелами (или табуляцией) для отступа строк, располагая ссылки друг под другом. Нужно лишь помнить о необходимости указать запятую после каждой ссылки, за исключением последней, для которой обычно требуется точка с запятой, завершающая объявление.

Поскольку фоновые изображения обычно выстраиваются в плитку, зачастую необходимо также включать и объявление свойства `background-repeat` (если этого не сделать, изображения начнут выстраиваться в плитку одно над другим, перекрывая собой все нижние изображения). Можно добавить и другие свойства фона, состоящие из простого набора значений, для которого в качестве разделителя используется запятая:

```
background-repeat: no-repeat,  
                 no-repeat,  
                 repeat-y;
```

При использовании нескольких значений, подобных этим, первое значение (в примере — `no-repeat`) применяется для первого изображения, указанного в свойстве `background-image` (`scrollTop.png`); второе применяется со вторым изображением и т. д. Поскольку во всем этом можно быстро запутаться, многие веб-разработчики используют для указания нескольких изображений сокращенный метод записи:

```
background: url(scrollTop.jpg) center top no-repeat,  
           url(scrollBottom.jpg) center bottom no-repeat,  
           url(scrollMiddle.jpg) center top repeat-y;
```

ПРИМЕЧАНИЕ

Несколько фоновых изображений кладутся стопкой друг на друга, как слои в программе редактирования изображений. Какое из них появится в верхнем слое, определяется порядком перечисления фоновых изображений. Изображение, указанное первым, появляется в верхнем слое элемента, второе — во втором слое и последнее появляется в нижнем слое. В предыдущем примере кода верхняя часть свитка (`scrollTop.jpg`) находится выше его нижней части (`scrollBottom.jpg`), которая, в свою очередь, находится выше текстовой области свитка (`scrollMiddle.jpg`).

Использование градиентных фонов

Градиент — это плавный переход цвета, к примеру, от синего к красному или от черного к белому. Градиенты входят в состав основных элементов любой программы редактирования изображений. Создание тонких переходов от одного схожего цвета к другому придает изображению своеобразную расплывчатость;

Apple использует градиенты в изображениях кнопок и других элементов пользовательского интерфейса операционных систем OS X и iOS. Раньше приходилось обращаться к программе Photoshop и создавать огромные файлы с градиентом для последующего использования. Теперь задачу создания градиента можно возложить на браузер.

В CSS поддерживаются фоновые градиенты, представляющие собой, по сути, фоновые изображения, создаваемые браузером на лету. Фактически для создания градиента используется обычное свойство `background-image`. Существует несколько типов градиентов.

Линейные градиенты

Самым простым является *линейный градиент*. Он распространяется по прямой от одного конца элемента к другому, демонстрируя плавный переход от одного цвета к другому (рис. 8.14). Нужно только указать его направление в градусах или с помощью ключевых слов `top`, `bottom`, `right`, `left` либо их комбинаций, например `bottom left`.



Рис. 8.14. Линейные градиенты позволяют отказаться от устаревшего метода добавления градиентов к фону элемента: создания градиента в виде графического изображения в программе Photoshop или Fireworks с последующим использованием свойства `background-image` для помещения этого изображения в качестве фона элемента

Например, чтобы нарисовать градиент, изменяющий цвет от черного к белому от левого угла к правому, нужно воспользоваться следующим кодом:

```
background-image: linear-gradient(to right, black, white);
```

Чтобы нарисовать градиент, изменяющий цвет от черного к белому от верхней части элемента к нижней, нужно воспользоваться таким кодом:

```
background-image: linear-gradient(to bottom, black, white);
```

Вы также можете нарисовать градиент под углом, используя ключевые слова, такие как `bottom right`, для указания направления градиента. Чтобы нарисовать градиент, изменяющий цвет от оранжевого к красному от верхнего левого угла элемента к его правому нижнему углу, нужно воспользоваться следующим кодом:

```
background-image: linear-gradient(to bottom right, orange, red);
```

Градиенты можно создавать с помощью любых значений цвета, принятых в языке CSS (см. раздел «Форматирование текста цветом» главы 6), например ключевых слов, таких как `white` или `black`, шестнадцатеричного кода (`#000000`) или значений RGB, к примеру `rgb(0,0,0)`, и т. д.

Но вы не ограничены применением ключевых слов. Можно указать угол, определяющий направление градиента. Угол записывается в виде значения в диапазоне от 0 до 360, а за ним указывается ключевое слово `deg`: `0deg` (рис. 8.15). Например, значение `0deg` соответствует верхнему краю элемента, поэтому градиент начнется в нижней части элемента и закончится в верхней. Другими словами, код выглядит следующим образом:

```
background-image: linear-gradient(to top, black, white);
```

или

```
background-image: linear-gradient(0deg, black, white);
```

Значения в углах повторяют часовую стрелку на циферблате, поэтому значение `90deg` соответствует правому краю элемента (то же самое, что ключевое слово `right`), `180deg` — нижнему краю элемента (ключевое слово `bottom`), а `270deg` — левому краю элемента (`left`).

При использовании значений углов браузер проводит воображаемую линию через центр элемента. Указанный угол является углом наклона этой линии, а также определяет точку, в которой заканчивается градиент. Например, градиент, показанный на рис. 8.15, создан под углом 48° . Браузер проводит воображаемую линию через центр элемента, отмечая угол 48° . Это означает, что градиент начинается у левой нижней границы элемента и движется к правой верхней границе.

Например, для градиента, показанного на рис. 8.14, *слева внизу*, используется следующее объявление:

```
background-image: linear-gradient(135deg, rgb(0,0,0), rgb(204,204,204));
```

По адресу tinyurl.com/nu47ro6 вы найдете интерактивный пример, как различные значения углов влияют на градиент.

Узлы градиента

В рассматриваемых выше градиентах использовались только два цвета, но их можно добавить любое количество. Дополнительные цвета называются *цветовыми*

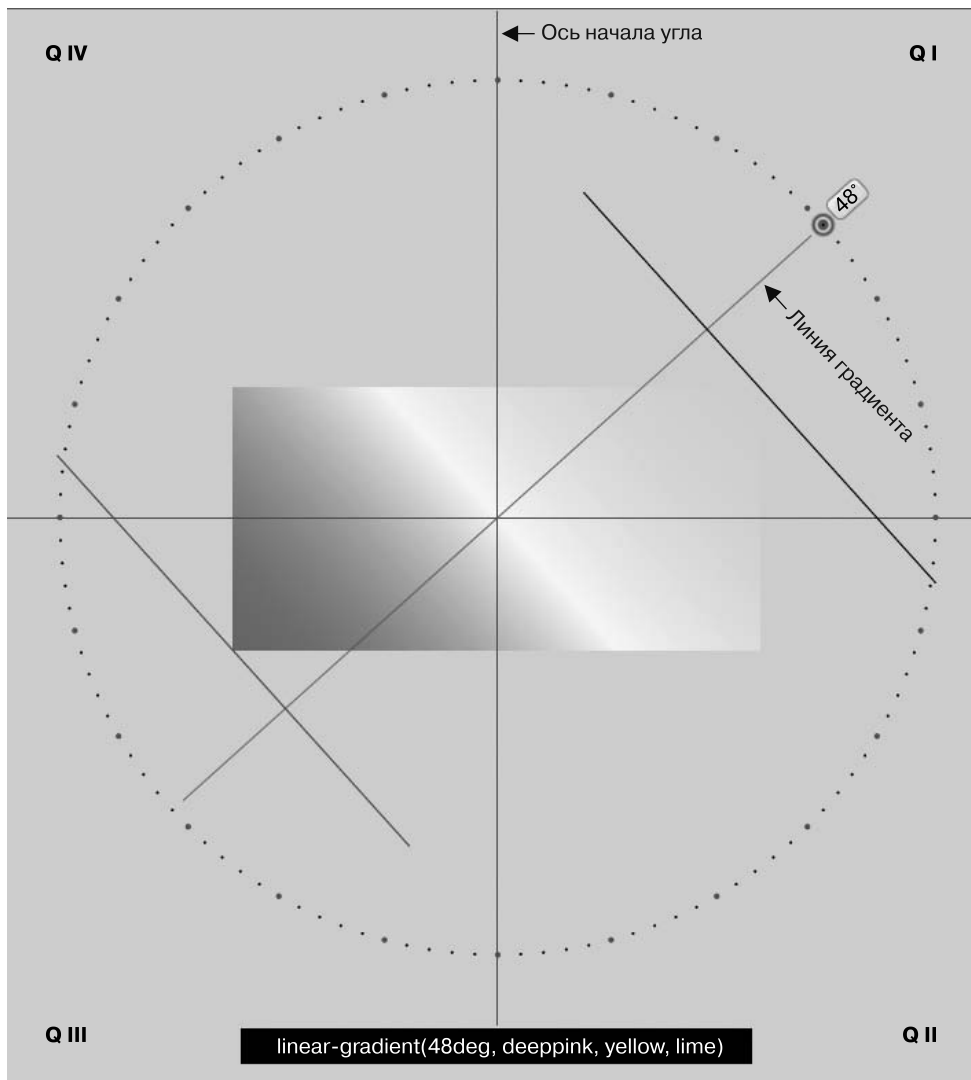


Рис. 8.15. Значение в градусах, указанное для линейного градиента, применяется для создания линии, проходящей через середину воображаемой точки центра элемента. Градиент рисуется вдоль этой линии. Значение определяет как точку старта, так и наклон градиента

узлами и позволяют фону осуществлять плавный переход от одного цвета ко второму, затем от второго к третьему и т. д., пока градиент не будет завершён последним цветом. Для добавления цветowych узлов нужно добавить к градиенту дополнительные значения цвета. Например, нижнее правое изображение на рис. 8.14 состоит из трех цветов и градиент переходит от белого к черному и от черного к белому цвету:

```
background-image: linear-gradient(to right, black, white, black);
```

Разумеется, можно применять любые значения цвета (имена, шестнадцатеричные или RGB-значения) и даже перемешивать их друг с другом. Следовательно, предыдущую строку кода можно записать следующим образом:

```
background-image: linear-gradient(to right, black, rgb(255,255,255), #FFFFFF, HSL(0,0%,0%));
```

Браузеры распределяют цвета равномерно, поэтому в этом примере в самой крайней левой точке фон черный, в центре фон белый, а в самой крайней правой точке он снова черный. Но можно разместить разные цветовые узлы и в более конкретно указанных местах фона, добавив после цвета второе значение.

СОВЕТ

Если добавить к элементу с градиентным фоном свойство `background-position`, можно получить весьма необычные эффекты. По сути, можно изменить то, что браузером рассматривается в качестве стартовой точки градиента.

Предположим, нужно получить в начале градиента темно-красный цвет, затем быстро перейти к ярко-оранжевому примерно на 10 % протяженности элемента, оставляя оранжевый цвет на большем протяжении элемента, а затем быстро вернуться к темно-красному цвету, как на верхнем изображении на рис. 8.16. Для этого нужно использовать четыре цветовых узла — красный, оранжевый, оранжевый и красный, и указать, где должны располагаться два оранжевых узла:

```
background-image: linear-gradient(to right, #900, #FC0 10%, #FC0 90%, #900);
```

Обратите внимание на то, что значение 10% применяется ко второму цвету (оранжевому): тем самым браузеру сообщается, что он должен достичь этого цвета к 10 % ширины элемента. Точно так же значение 90% показывает, что браузеру нужно оставить оранжевый цвет до достижения 90 % ширины элемента, а затем приступить к плавному переходу к темно-красному цвету на правом краю элемента.

ПРИМЕЧАНИЕ

Для указания позиций цветовых узлов не обязательно указывать проценты. Можно также использовать пиксели или единицы `em`. Но указание процентных значений является более гибким решением, приспособившимся к изменениям ширины и высоты элемента. Одним из исключений является рассматриваемый далее повторяющийся линейный градиент. При создании этого типа плиточных градиентов очень хорошо подойдут пиксельные значения.

Как для первого, так и для последнего цвета указывать какие-либо значения положения не нужно, поскольку браузер предполагает, что первый цвет начинается с 0 %, а последний заканчивается на 100 %. Но если нужно сохранять начальный цвет неизменным на некоторой протяженности элемента, можно после первого цвета указать значение этой протяженности. Например, нижнее изображение на рис. 8.16 создано с помощью такого объявления:

```
background-image: linear-gradient(to right, #900 20%, #FC0 30%, #FC0 70%, #900 80%);
```

Обратите внимание, что у первого цвета — `#900` — также есть позиция 20%. Это означает, что первые 20 % протяженности элемента (слева направо) будут иметь фон из сплошного красного цвета. Затем, от 20%-ной до 30%-ной точки градиент плавно перейдет от красного к оранжевому цвету.

СОВЕТ

Ключевое слово `transparent` можно использовать для любого цвета в градиенте, чтобы видеть область, расположенную позади него, например фоновый цвет элемента или даже другой линейный градиент. Помните, что к элементу может быть применено несколько фоновых изображений, поэтому вы можете использовать несколько градиентов в одном элементе. Например, попробуйте добавить следующий код для достижения невероятного эффекта:

```
background-image:  
  linear-gradient(cyan, transparent),  
  linear-gradient(225deg, magenta, transparent),  
  linear-gradient(45deg, yellow, transparent);
```



Рис. 8.16. Область появления цветового узла на градиенте можно указать вполне конкретно. В верхнем примере к 10 % ширины элемента слева направо темно-красный цвет переходит в оранжевый

Поддержка браузера Internet Explorer

Градиенты не поддерживаются в программе Internet Explorer 9 и более ранних ее версиях. Если вы решили использовать градиенты, для IE 9 и более ранних версий нужно указывать резервный цвет. Подберите сплошной цвет, соответствующий общему тону вашего градиента, и объявите его первым, а за ним разместите все объявления градиентов:

```
background-color: #FC0;
background-image: linear-gradient(to bottom, #900, #FC0, #900);
```

Браузер Internet Explorer 9 применит фоновый цвет и пропустит все остальные, неподдерживаемые объявления. Другие браузеры применяют фоновый цвет, но также создадут градиент, который перекроет фоновый цвет. Если применяются RGBA-цвета с некоторой степенью прозрачности, то вам не захочется, чтобы сквозь них просматривался фоновый цвет. В таком случае воспользуйтесь сокращенной формой записи свойства background, и свойство background-color будет замещено (благодаря особому поведению сокращенной формы записи свойства background). Тогда для объявления цвета в формате RGBA можно будет добавить следующий код:

```
background-color: #FC0;
background: linear-gradient(to bottom, rgba(153,0,0,.5), #FC0,
  rgba(153,0,0,.5));
```

Повторяющиеся линейные градиенты

Обычно линейный градиент заполняет весь элемент от первого цвета с одного края элемента до последнего цвета на его противоположном краю. Но при желании можно создать их повторяющийся вариант. По сути, при этом определяется градиент с указанными цветовыми узлами, браузер рисует градиент, а затем повторяет этот шаблон, распространяя его в виде плиток по фону элемента. Например, чтобы получить повторяющийся градиент, как на изображении на рис. 8.17, *слева*, следует написать такой код:

```
background-image: repeating-linear-gradient(45deg, #900 20px, #FC0 30px, #900 40px)
```

В подобных случаях для цветовых узлов лучше всего подойдут пиксельные значения. Получается, что браузер рисует градиент, начинающийся в нижнем левом углу с 20 пикселей темно-красного цвета, затем до 30 пикселей идет переход к оранжевому цвету, а затем до 40 пикселей происходит обратный переход к темно-красному цвету. После того как этот градиент будет нарисован, браузер разместит его в виде плитки в фоне элемента.

ПРИМЕЧАНИЕ

Для экспериментов с градиентами имеются широчайшие возможности. Чтобы посмотреть только на малую часть тех удивительных результатов, которых можно достичь в ходе таких экспериментов, посетите галереи шаблонов CSS по адресам tinyurl.com/85cb36t, tinyurl.com/Zashxxj. Кроме того, оцените удивительные возможности использования градиентов для создания флагов со всего мира по адресу tinyurl.com/qc4l6wk и некоторые невероятные градиенты, созданные с помощью режима наложения background-blend, описанного во врезке «Добавление специальных эффектов к фоновым изображениям» в разделе «Позиционирование фоновых элементов», tinyurl.com/nl9orcw.

Можно также использовать повторяющиеся градиенты для создания однотонных полос без каких-либо излишеств и переходов между цветами. Например, изображение, показанное на рис. 8.17, *справа*, создано с помощью такого объявления:

```
background-image: repeating-linear-gradient(45deg, #900 0, #900 10px, #FC0 10px, #FC0 20px);
```

Оно начинается с темно-красного цвета (#900) в точке 0 и снова переходит к красному цвету в точке 10px. Поскольку переход задается между одинаковыми цветами, браузер рисует его в виде сплошного цвета. Затем на 10 пикселах задается переход к оранжевому цвету (#FC0). Поскольку это та же самая точка, в которой заканчивается красный цвет, никакого плавного перехода не происходит, шаблон не содержит перехода от красного к оранжевому цвету. И наконец, переход к такому же оранжевому цвету осуществляется до 20 пикселей, создавая еще одну сплошную линию. Поскольку это повторяющийся линейный градиент, браузер заполняет шаблоном фон элемента.

Повторяющиеся градиенты работают в большинстве браузеров, но только не в Internet Explorer 9 и более ранних версиях (работает и в Edge). Поэтому лучше будет добавить запасной вариант фонового цвета:

```
background-color: #FC0;  
background: repeating-linear-gradient(to bottom left, #900 20px, #FC0 30px, #900 40px)
```

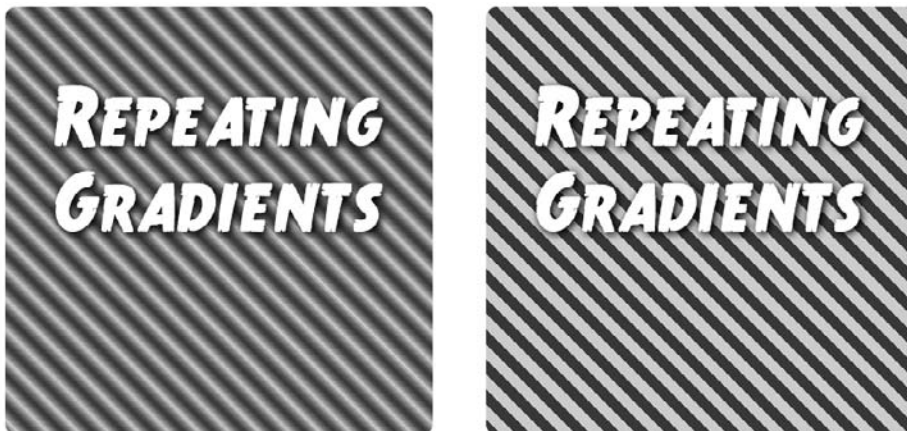


Рис. 8.17. Используя повторяющиеся линейные градиенты, можно создавать полосатые шаблоны. С помощью различных цветовых узлов можно создавать полосы из четких или расплывчатых по краям линий

Радиальные градиенты

В CSS также предоставляется способ создания *радиальных градиентов*, то есть градиентов, расходящихся наружу по круговой или эллиптической схеме (рис. 8.18). Синтаксис похож на тот, что применяется для линейных градиентов, нужно только задать начальный цвет (цвет в середине градиента) и конечный (цвет в конце

градиента). Например, верхнее левое изображение на рис. 8.18 создано с помощью следующего кода:

```
background-image: radial-gradient(red, blue);
```

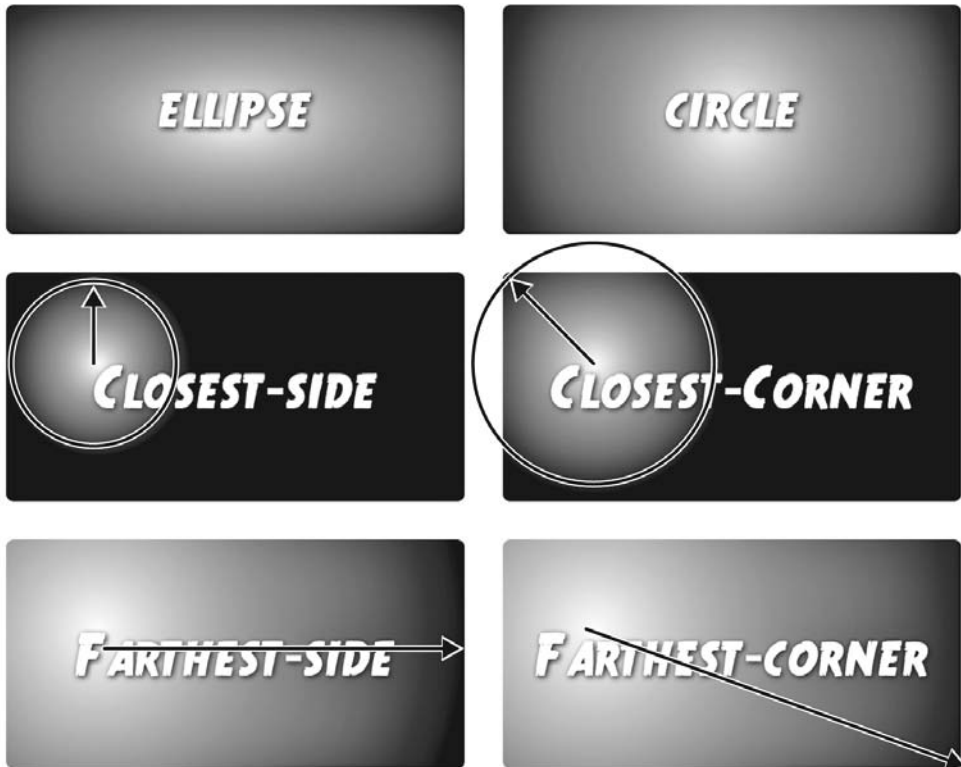


Рис. 8.18. Радиальные градиенты бывают эллиптические (слева сверху) или круговые (справа сверху). Позицией градиента в фоне элемента, а также его размерами можно управлять

Этот код создает эллиптическую форму, помещающуюся в высоту и ширину элемента с центром градиента (где начинается красный цвет) в центре самого элемента.

Можно также создать круговые градиенты, добавив перед указанием цветов ключевое слово `circle`. Например, верхнее левое изображение на рис. 8.19 создано с помощью следующего кода:

```
background-image: radial-gradient(circle, red, blue);
```

Обычно браузер рисует центр радиального градиента в центре элемента, но центр градиента можно позиционировать с помощью таких же ключевых слов позиционирования и значений, которые используются для свойства `background-position`. Например, среднее левое изображение на рис. 8.18 является радиальным градиентом, который начинается с отступом 20 % от левого края элемента и с отступом 40 % от его

верхнего края. Значения позиционирования указываются перед ключевыми словами, определяющими форму и цвета:

```
background-image: radial-gradient(circle at 20% 40%, red, blue);
```

Чтобы указать размер градиента, можно воспользоваться одним из четырех ключевых слов.

- `closest-side` — предписывает браузеру создать градиент, распространяющийся из центра только до ближайшей к центру стороне элемента. Например, в среднем левом изображении на рис. 8.18 ближайшей к центру градиента стороной является верхний край элемента, поэтому радиус окружности простирается от центра до этого края. Тем самым весь градиент остается внутри элемента. Код для создания этого градиента имеет следующий вид:

```
background-image: radial-gradient(closest-side circle at 20% 40%, red, blue);
```

Когда ключевое слово `closest-side` применяется к эллиптическому градиенту, для вычисления значений x и y радиусов эллипса используются обе ближайшие стороны (то есть верхняя или нижняя и левая или правая).

- `closest-corner` — приводит к вычислению ширины градиента из его центра до ближайшего угла элемента (см. рис. 8.18, *справа посередине*). Это может означать выход градиента за пределы элемента. Например, окружность, нарисованная в верхней части среднего правого изображения на рис. 8.18, показывает фактический размер градиента. Часть градиента выходит за границы элемента:

```
background-image: radial-gradient(closest-corner circle at 20% 40%, red, blue);
```

- `farthest-side` — приводит к вычислению радиуса окружности от ее середины до самой дальней стороны элемента. В случае эллиптического градиента это расстояние от центра до любой самой дальней верхней или нижней стороны и до любой самой дальней левой или правой стороны. Код, с помощью которого создано нижнее левое изображение на рис. 8.18, имеет следующий вид:

```
background-image: radial-gradient(farthest-side circle at 20% 40%, red, blue);
```

- `farthest-corner` — приводит к вычислению радиуса окружности от ее центра до самого дальнего угла элемента. Код, с помощью которого создано нижнее правое изображение на рис. 8.18, имеет такой вид:

```
background-image: radial-gradient(farthest-corner circle at 20% 40%, red, blue);
```

Как и в случае линейных градиентов, можно использовать несколько цветových узлов и указывать для них конкретные позиции. Предположим, нужно создать круговой градиент, составленный из темно-красного, оранжевого и желтого цветов. Нужно, чтобы красный цвет сохранялся на некотором протяжении до его перехода в оранжевый, а затем оранжевый цвет сохранялся на некотором протяжении до перехода в желтый, появляющийся в конце градиента. Чтобы указать позиции появления того или иного цвета, можно воспользоваться процентными значениями:

```
background-image: radial-gradient(circle at 20% 40%, red 20%, orange 80%, yellow);
```

Разумеется, как и в случае линейного градиента, можно воспользоваться допустимыми в языке CSS значениями цвета и переписать этот код следующим образом:

```
background-image: radial-gradient(circle at 20% 40%, rgb(255,0,0) 20%, rgb(255,165,0) 80%, #FFFF00);
```

Как и в случае с линейными градиентами, Internet Explorer 9 и более ранние его версии не поддерживают радиальные градиенты и не будут понимать, о чем идет речь. Нужно добавить резервный фоновый цвет:

```
background-color: red;
background: radial-gradient(circle at 20% 40%, red 20%, orange 80%, yellow);
```

Повторяющиеся радиальные градиенты

Как и в случае с линейными градиентами, радиальные градиенты можно создать повторяющимися, что подойдет для гипнотизирования посетителей вашего сайта. Чтобы браузер «знал» размер отдельно взятого радиального градиента и смог его повторить, нужно обязательно добавить к различным цветовым узлам процентные, пиксельные значения или значения `em`. Например:

```
background-image: repeating-radial-gradient(circle, red 20px, orange 30px, yellow 40px, red 50px);
```

Учтите, что для создания повторяющихся радиальных градиентов без резких переходов градиент нужно завершать тем же цветом, с которого он начинался (в примере это красный цвет). Тем самым будет обеспечено плавное возвращение цвета к начальному. Если этого не сделать (например, если последним цветом в показанном выше коде сделать желтый), то получится резкая граница там, где заканчивается последний цвет и в повторяющемся градиенте начинается первый цвет.

СОВЕТ

Поскольку браузеры считают линейные и радиальные градиенты фоновым изображением, для них можно использовать и другие свойства фоновых изображений, такие как `background-size`, `background-position` и т. д. Кроме того, в одном стиле можно использовать несколько градиентов, перечислив их через запятую, точно так же, как ранее это делалось с несколькими фоновыми изображениями. В одном элементе можно также смешивать изображения с градиентами.

Практикум: совершенствуем изображения

Фотогалерея — отличный пример привлекательной веб-страницы. Этот практикум воедино собирает и наглядно демонстрирует различные способы форматирования изображений. Мы попрактикуемся в добавлении рамок и подрисовочных подписей, создадим универсальную фотогалерею, легко адаптируемую для просмотра на устройствах с различными размерами экрана, применим рассмотренное ранее свойство `box-shadow` для создания профессиональных визуальных эффектов тени.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке

и загрузите ZIP-архив (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 08.

Заключение изображения в рамку

Мы будем работать над веб-страницей вымышленного сайта CosmoFarmer.com (рис. 8.19). У нас уже есть присоединенная внешняя таблица стилей, применяющая форматирование к веб-странице и обеспечивающая ей простейший дизайн.

1. Откройте файл `image.html` из папки `08\01_image_ex` в браузере.

Изображение занимает на странице слишком много места (см. рис. 8.19, *вверху*). С помощью CSS-кода вы легко можете заключить рисунок в симпатичную рамку и визуальнo выделить его из основного текстового контента (см. рис. 8.19, *внизу*).

2. Откройте файл `styles.css` из папки `01_image_ex` в редакторе HTML-кода.

Он представляет собой внешнюю таблицу стилей, используемую файлом `image.html`. Вы начнете с добавления класса в эту таблицу, а затем примените класс к элементу `img` в HTML-файле.

3. Перейдите к концу файла и наберите следующее:

```
img.figure {  
  
}
```

Селектор `img.figure` воздействует на любой элемент `img`, к которому применен класс `figure`. Вы будете использовать его для выборочного форматирования некоторых изображений (вы могли бы назвать стиль `.figure`, но в таком случае он применялся бы к *любому* элементу с этим классом, а не только к изображениям).

4. Добавим в только что созданный стиль свойства `float` и `margin`:

```
float: right;  
margin: 10px;
```

Свойство `float` смещает изображение к правой стороне веб-страницы, приподнимая текст вверх и создавая обтекание фотографии с левой стороны. Поля обеспечивают небольшие свободные промежутки, отделяющие фото от текста. Теперь добавим границу и небольшие отступы, чтобы изображение больше походило на настоящий фотоснимок.

5. Добавим границу, цвет фона и отступы. Законченный вариант стиля должен иметь следующий вид:

```
img.figure {  
  float: right;  
  margin: 10px;  
  border: 1px solid #666;  
  background-color: #CCC;  
  padding: 10px;  
}
```

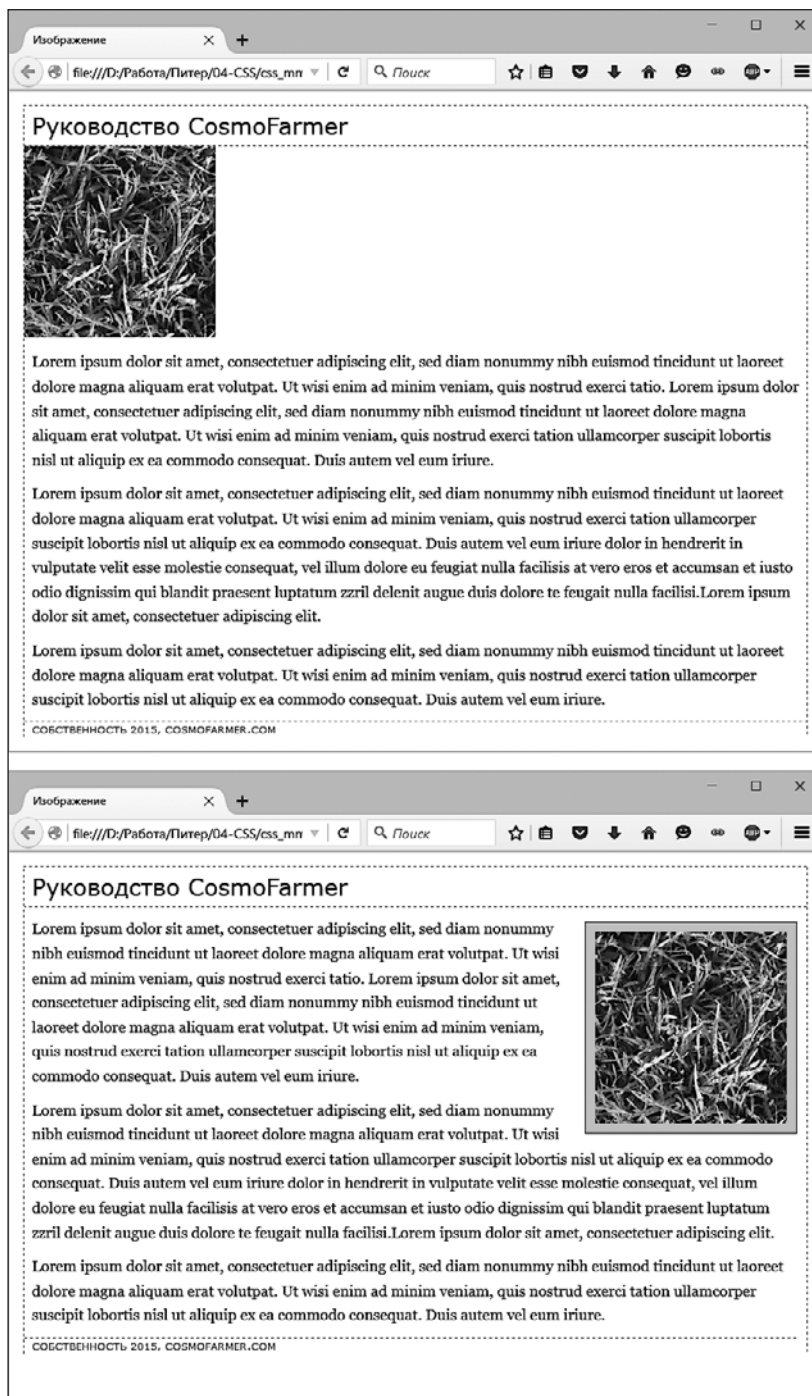



Рис. 8.19. Два варианта веб-страницы: до (вверху) и после (внизу) форматирования с помощью CSS

Если вы сохраните и просмотрите веб-страницу в браузере прямо сейчас, то не будет видно никаких изменений, поскольку класс не возымает эффекта, пока его не применить к элементу.

6. Сохраните и закройте файл `styles.css` и откройте файл `image.html`. Найдите элемент `img` и примените к нему класс `class="figure"` так, чтобы код имел следующий вид:

```

```

Теперь изображение приобретет все свойства форматирования, определенные для класса `.figure`.

7. Просмотрите веб-страницу в браузере. Она должна выглядеть так, как показано на рис. 8.19, *внизу*.

Финальную версию веб-страницы, получившейся в данном практикуме, вы сможете найти в папке `08_finished\01_image_ex`, файл `image.html`.

Изображение может заменить тысячу слов, но нередко требуется добавить к изображению текстовую подпись с подробной информацией об объекте, месте съемки и т. д. При этом вы наверняка захотите, чтобы комментарий был единым целым с изображением, чтобы соседний текст обтекал надпись так же, как и само изображение. Лучший способ — заключить изображение и текст в контейнер, для которого будет определено единое форматирование. В таком случае фотография и связанный с ней текст обрабатываются как цельный блочный элемент. Если вы позже решите изменить их размещение на веб-странице и, возможно, установить обтекание с выравниванием по левому краю страницы — никаких проблем: вам потребуется изменить форматирование для всего элемента-контейнера.

В языке HTML5 именно для решения нашей задачи включены два новых элемента: `figure` предназначен для заключения в него изображения, из которого хотят сделать иллюстрацию. В дополнение к нему может использоваться элемент `figcaption`, который позволяет добавить подрисуночную подпись под изображением. Сначала нужно немного подправить HTML-код.

8. Вернитесь в редактор HTML-кода, к файлу `image.html`. Найдите в коде элемент `img` и удалите значение `class="figure"`, которое было добавлено при выполнении шага 6. Добавьте перед элементом `img` тег `<figure>`.

Этим тегом будет помечено начало контейнера. Теперь будет добавлена подпись и элемент `figure` будет закрыт, чтобы обозначить конец контейнера.

9. После элемента `` добавьте строки, выделенные ниже полужирным шрифтом. HTML-код приобрет такой вид:

```
<figure>

<figcaption>Рисунок 1: Полевица побегообразующая лучше всего растет в открытом
грунте.</figcaption>
</figure>
```

10. Вернитесь к файлу `styles.css`. Прокрутите файл до стиля `img.figure`, созданного ранее, и удалите этот стиль.

Для элемента `figure` будет добавлен новый стиль.

11. Добавьте следующий стиль в файл `styles.css`:

```
figure {  
    float: right;  
    width: 222px;  
    margin: 15px 10px 5px 10px;  
}
```

В предыдущем уроке свойство `float: right` уже использовалось. Свойство `margin` добавит вокруг всех четырех сторон элемента `figure` небольшое пустое пространство. Возникает вопрос: а для чего нужно свойство `width`? Хотя у фотографии и есть установленная ширина (200 пикселей), ее нет у абзаца подписи. Если не указать ширину, абзац заставит элемент `figure` стать шире фотографии. В данном случае нужно, чтобы подпись была такой же по ширине, как фотография и ее рамка.

Значение 222 пикселя получилось в результате небольших математических вычислений общей области, занимаемой фотографией на странице: хотя сама фотография занимает в ширину 200 пикселей, у нее появится (вы добавите на следующем шаге) по 10 пикселей отступов слева и справа. Кроме того, у изображения имеется рамка, занимающая слева и справа по 1 пикселу, что в целом и составляет ширину фотографии от одной границы к другой, то есть 222 пикселя (см. подраздел «Вычисление фактических размеров блочных элементов» раздела «Изменение высоты и ширины» главы 7).

12. Добавьте в файл `styles.css` следующий стиль:

```
figure img {  
    border: 1px solid #666;  
    background-color: #CCC;  
    padding: 10px;  
}
```

Этот селектор потомков воздействует на любой элемент `img`, который находится внутри элемента `figure`. Поскольку здесь используется селектор потомков, класс к элементу `img` добавлять не нужно. Затем добавим стиль к подписи.

13. Добавьте в файл `styles.css` следующий стиль:

```
figcaption {  
    font: bold 1em/normal Verdana, Arial, Helvetica, sans-serif;  
    color: #333;  
    text-align: center;  
}
```

Этот стиль использует некоторые свойства, изученные вами в главе 6 и позволяющие создать подрисовочную подпись, отформатированную полужирным шрифтом `Verdana` серого цвета с выключкой по центру. Сокращенный способ записи свойства `font` позволяет записать четыре различных свойства в одном объявлении (на одной строке).

Чтобы подрисуночную подпись выделить еще больше, добавим цветной фон и границу.

- Добавьте в стиль `figcaption` следующие три свойства:

```
figcaption {
  font: bold 1em/normal Verdana, Arial, Helvetica, sans-serif;
  color: #333;
  text-align: center;
  background-image: linear-gradient(to bottom, #e6f3ff, white);
  border: 1px dashed #666;
  padding: 5px;
}
```

Чтобы создать цветную рамку вокруг заголовка, значения свойств `background-image`, `border` и `padding` должны быть сброшены. Мы используем линейный градиент с помощью свойства `background-image`, чтобы создать позади подписи градиент с переходом из светло-голубого в белый цвет.

- Сохраните файлы `image.html` и `styles.css`, а затем просмотрите файл `image.html` в браузере.

Теперь можно понять одну из причин, по которой разработку дизайна вести проще с использованием внешней таблицы стилей — приходится работать только с одним файлом, а не с двумя, и сохранять только его. Страница должна иметь вид, показанный на рис. 8.20. (Полную версию этой страницы можно найти по адресу `08_finished/01_image_ex.`)

Практикум: создание фотогалереи

Раньше веб-дизайнеры создавали фотогалереи на основе HTML-элемента `table`, помещая изображения в ячейки таблицы, образуемые строками и столбцами. Но сейчас вы можете достигнуть того же эффекта с помощью лаконичного CSS-кода в сочетании с применением гораздо менее объемного HTML-кода.

- Откройте файл `gallery.html` из папки `08\02_gallery_ex.`

Во-первых, взгляните на HTML-код, который использован для создания фотогалереи. Веб-страница содержит шесть фотографий и подписей к ним. Каждая фотография и подпись к ней хранятся в контейнере `figure`. Сама фотография находится в элементе `img`, а подпись к ней — в элементе `figcaption`.

```
<figure>
  
  <figcaption>Рисунок 6: Фотопокрытия изображают природные склоны. </figcaption>
</figure>
```

Если вы просмотрите страницу сейчас, то увидите серию фотографий, размещенных друг над другом, что выглядит не очень привлекательно. Галерея будет выглядеть лучше, если фотографии будут находиться друг рядом с другом, объединяясь в ряды. Вы можете сделать это с помощью всего нескольких стилей.



Рис. 8.20. Используя в качестве контейнера элемент `figure`, а также применив обтекание слева и некоторое форматирование, вы с легкостью добавили подрисуночную подпись к снимку

2. Откройте файл `styles.css`.

Таблица стилей уже содержит код сброса CSS и несколько стилей для заголовка и остального текста. Для начала вы добавите несколько стилей для фотографий и их заголовков.

3. Добавьте два новых стиля в верхней части файла `styles.css`:

```
figure img {
  border: 1px solid #666;
  background-color: #FFF;
  padding: 4px;
}

figcaption {
  font: 1.1em Arial, Helvetica, sans-serif;
  text-align: center;
  margin: 10px 0 0 0;
}
```

Эти стили добавляют границы-рамки ко всем изображениям галереи и устанавливают шрифт, выравнивание и поля для подрисуночных подписей. В первом стиле для выбора только тех изображений, которые находятся внутри элементов `figure`, используется селектор потомков.

Теперь поместим фотографии друг рядом с другом.

4. Добавьте в таблицу стилей такой код:

```
figure {
  float: left;
  width: 210px;
  margin: 0 10px 10px 10px;
}
```

Он создает такое обтекание, при котором все пары «фотография/подпись» выравниваются по левому краю окна браузера. На самом же деле браузер размещает фотографии на одном уровне, рядом друг с другом, пока не закончится свободное место в строке. Затем браузер переносит следующие изображения на строку ниже, пока не отобразит все, и т. д. Общая ширина *складывается* из ширины самой фотографии плюс отступы и границы-рамки. В данном примере имеем: 200 пикселей на фотографию, 8 пикселей на левый и правый отступы и 2 пиксела на левую и правую границы.

5. Сохраните файл и просмотрите веб-страницу `gallery.html` в браузере. Она должна быть похожа на страницу, представленную на рис. 8.21, *слева*.

Измените ширину окна браузера так, чтобы оно стало уже или шире, и наблюдайте, как перераспределяются изображения. Вы увидите, что не совсем правильно. Во второй строке изображений имеется два пустых места, где должны располагаться фотографии. Эта проблема возникает по той причине, что подпись ко второму изображению на первой строке больше по высоте, чем другие на этой же строке. Изображения, которые переносятся на эту подпись, не могут разместиться рядом (об этой путанице, появляющейся при использовании свойства `float`, читайте в разделе «Решение проблем с обтекаемыми элементами» главы 13). Существует простое решение такой проблемы.

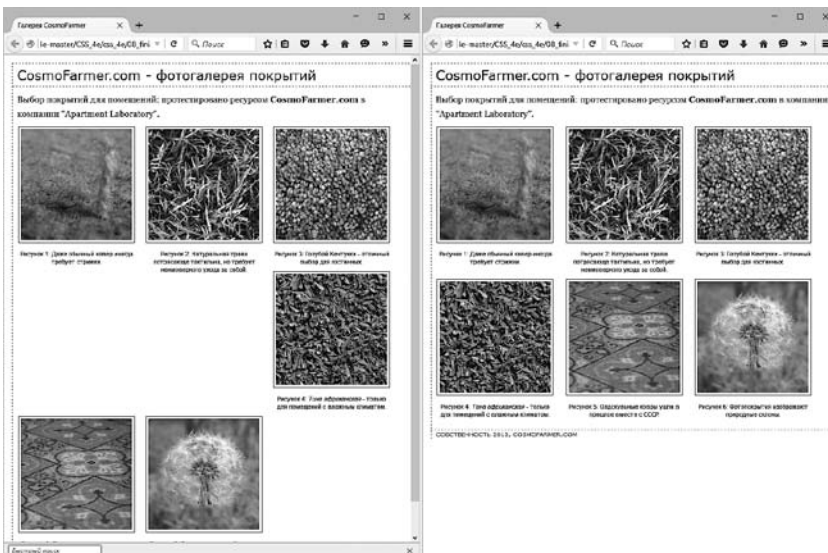


Рис. 8.21. Плавающие элементы, расположенные рядом друг с другом, — один из способов имитации столбцов и строк таблицы

- Вернитесь к файлу `styles.css` в редакторе HTML-кода. Найдите стиль `figure`. Удалите свойство `float:left` и добавьте свойства `display:inline-block;` и `vertical-align:top`. Код должен приобрести такой вид:

```
figure {
  display: inline-block;
  vertical-align: top;
  width: 210px;
  margin: 0 10px 10px 10px;
}
```

Свойство `display:inline-block` (см. пункт «Использование свойств `display:inline` и `display:inline-block`» подраздела «Горизонтальные панели навигации» раздела «Создание панелей навигации» главы 9) рассматривает каждую пару «изображение/подпись» в качестве блока (у которого есть высота и ширина), но также и в качестве строчного элемента (то есть блоки могут выстраиваться в ряд). Кроме того, свойство `vertical-align` со значением `top` гарантирует выравнивание каждого элемента `figure` по верхнему краю других элементов `figure`, имеющих-ся в данном ряду.

- Сохраните файл и воспользуйтесь предварительным просмотром веб-страницы в браузере. Взгляните на правую часть рис. 8.21.

Если вы измените размеры окна браузера, то вид фотогалереи тоже изменится. В более широкое окно может вестись четыре или даже пять изображений, но, если размер уменьшить, вы увидите, что на одной строке отображается всего один или два рисунка.

Добавление теней

Наша фотогалерея выглядит хорошо, но ее можно сделать еще выразительнее. Добавление эффектов тени к каждой фотографии придаст веб-странице иллюзию глубины и обеспечит реалистичность трехмерного пространства. Однако прежде, чем запускать графический редактор типа Photoshop, стоит узнать, что добавить эффекты тени к любому изображению веб-страницы можно средствами CSS.

- Откройте в редакторе HTML-кода файл `styles.css`, над которым вы работали в предыдущей части практикума.

Изменение коснется созданного ранее стиля `img`.

- Добавьте в конец стиля `figure img` свойство `box-shadow: 2px 2px 4px rgba(0,0,0,.5);`, чтобы придать ему следующий вид (изменения выделены полужирным шрифтом):

```
figure img {
  border: 1px solid #666;
  background-color: #FFF;
  padding: 4px;
  box-shadow: 2px 2px 4px rgba(0,0,0,.5);
}
```


Свойство `box-shadow` было рассмотрено в разделе «Добавление тени» главы 7. Здесь добавляется тень, распространяющаяся на 2 пиксела вправо от изображения и на 2 пиксела ниже его, которая расширяется наружу на 4 пиксела. Использование цветовой модели RGBA (см. подраздел «RGBA» раздела «Форматирование текста цветом» главы 6), можно установить для тени черный цвет с 50%-ной прозрачностью.

3. Сохраните файл и просмотрите веб-страницу в браузере. Она должна иметь такой же вид, как на рис. 8.22.

Каждый рисунок отображается со своей собственной тенью, и вам не пришлось даже запускать Photoshop.

Законченную версию веб-страницы вы сможете найти в папке `08_finished\02_gallery_ex` учебного материала.

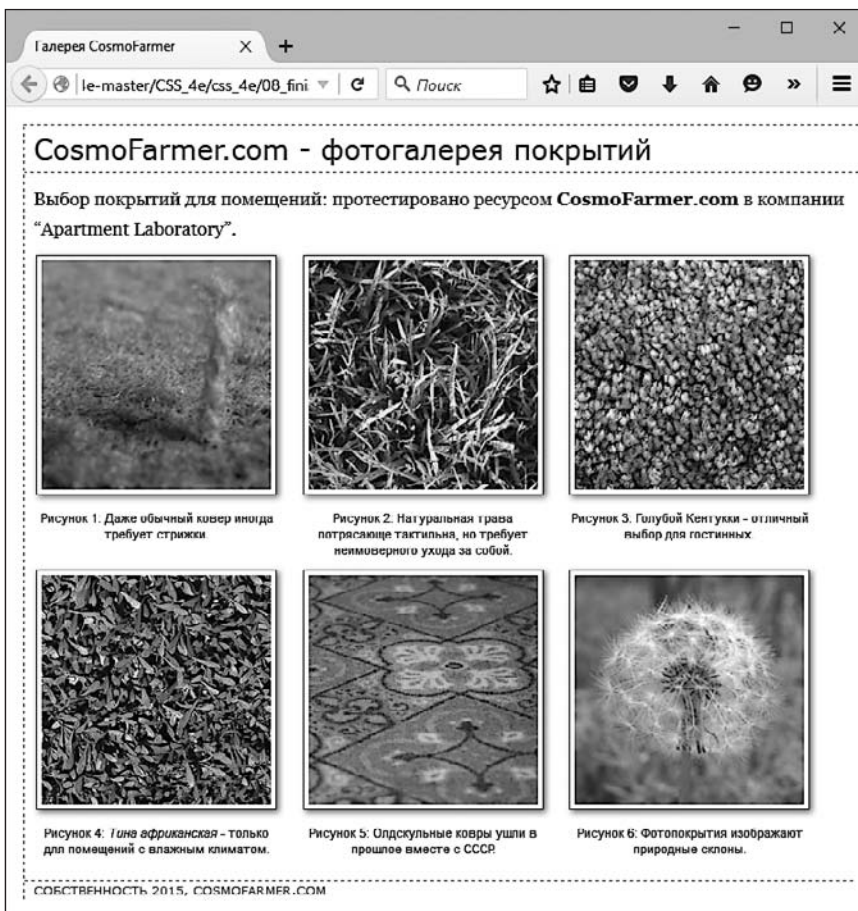


Рис. 8.22. Добавление эффекта тени к фотографиям придает веб-странице объем и улучшает визуальное восприятие любой фотогалереи. Используя таблицы каскадных стилей, можно добавить тень к любому изображению, не прибегая к помощи Photoshop

Практикум: использование фоновых изображений

Свойство `background-image` — секретное оружие современного веб-дизайна. Оно превращает скучный текст в великолепный графический образ (рис. 8.23). Вы можете использовать его для добавления фоновых изображений к любому HTML-элементу, и дизайн, который сумеете создать, ограничен только воображением. Пример эффекта тени, приведенный в предыдущем практикуме, — всего лишь один из способов креативного использования фоновых изображений. Основное их применение — в качестве общего фона веб-страницы, а также для создания собственных маркеров в списках. Эти наиболее распространенные способы применения свойства `background-image` мы и рассмотрим в текущем практикуме.

Добавление на веб-страницу фонового изображения

Что бы это ни было: сложный, замысловатый узор, логотип компании или полноэкранный фотография, — изображения очень часто используются в качестве фоновых рисунков веб-страниц. Фактически это и есть основное применение свойства `background-image`.

1. Откройте файл `bg_images.html` из папки `08\03_bg_ex` в редакторе HTML-кода.

Веб-страница имеет двухколоночный дизайн: она очень проста, содержит лишь немного отформатированного текста на белом фоне (см. рис. 8.23, *вверху*). Для начала добавим линейный градиент. К странице привязана внешняя таблица стилей с базовым форматированием. Вы используете ее, чтобы добавить новый стиль.

2. Откройте файл `styles.css`. Введите следующий код в нижней части страницы:

```
html {  
    background-image: linear-gradient(to bottom, rgb(176,194,213), white 700px);  
}
```

Этот код добавляет линейный градиент шириной 700 пикселей (см. раздел «Использование градиентных фонов» выше), начинающийся голубым цветом вверху и плавно переходящий в белый внизу.

3. Сохраните файл таблицы стилей и просмотрите страницу `bg_images.html` в браузере.

Произошло кое-что странное. Градиент переходит от голубого цвета к белому, а затем повторяет переход снова. Это неожиданно. Чтобы избежать такого эффекта, необходимо сообщить браузеру, что HTML-элемент должен заполнить окно браузера, присвоив параметру `height` значение `100%`.

4. В файл `styles.css` добавьте еще одно свойство (выделено полужирным шрифтом):

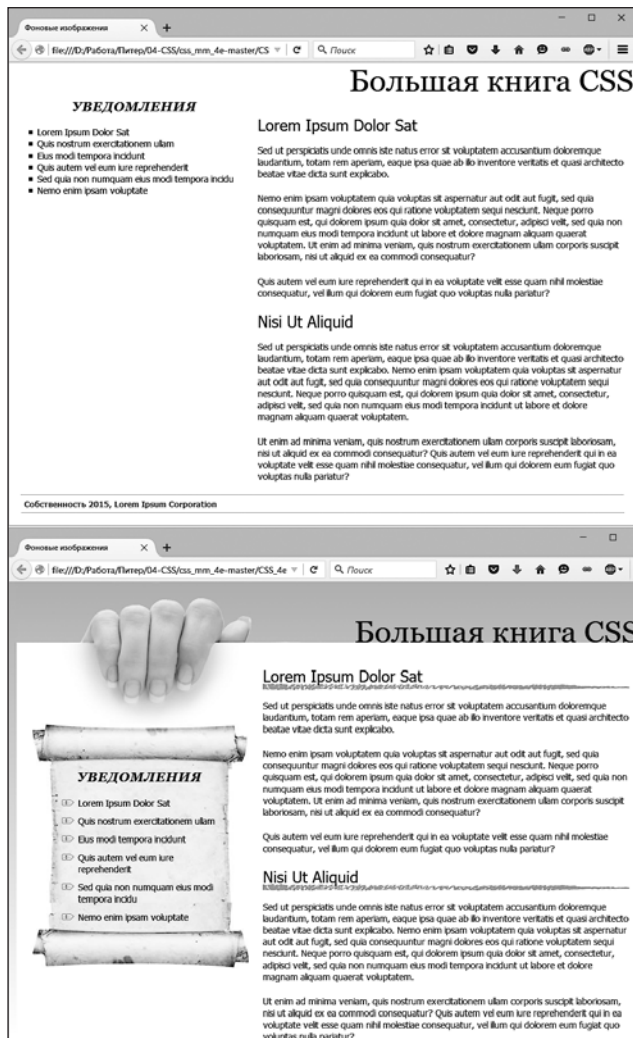


Рис. 8.23. С помощью фоновых изображений вы можете превратить упорядоченную веб-страницу (вверху) в еще более привлекательный сайт (внизу). Поскольку изображения могут добавляться в фон любого элемента на странице, возможности размещения графики на странице практически безграничны

```
html {
  height: 100%;
  background-image: linear-gradient(to bottom, rgb(176,194,213), white 700px);
}
```

Теперь, если вы сохраните файл таблицы каскадных стилей и просмотрите файл `bg_images.html`, градиент, расположенный в фоне, не повторится. Смотрится неплохо, но синий цвет есть и в фоне текста. Вы можете выделить текст, задав для него другой фоновый цвет.

- Вернитесь к редактору HTML-кода и файлу `style.css`. Добавьте еще один стиль для элемента `div`, в котором находится контент страницы:

```
.wrapper {  
  background-color: #FFF;  
}
```

У этого элемента `div` есть фиксированная ширина, он центрирован на странице и содержит весь ее текст. Стиль задает тексту белый фоновый цвет, но с помощью изображения вы можете сделать его лучше.

- Отредактируйте стиль, который вы создали в шаге 5, добавив фоновое изображение:

```
.wrapper {  
  background-color: #FFF;  
  background-image: url(images/bg_main.jpg);  
  background-position: left top;  
  background-repeat: no-repeat;  
}
```

Выделенные три строки кода добавляют фоновое изображение в левом верхнем углу контейнера `div`; значение `no-repeat` свойства `background-repeat` означает, что изображение появляется только один раз. Если вы сохраните файл и просмотрите его в браузере, то увидите изображение руки, которая словно держит эту страницу. Очень здорово. Но единственная проблема заключается в том, что текст находится слишком высоко вверху и прикрывает рисунок. Далее вы разместите на странице большой заголовок и левую боковую панель.

- Добавьте еще два стиля следующим образом:

```
.banner {  
  margin-top: 48px;  
}  
.announcement {  
  margin-top: 115px;  
}
```

Первая строка добавляет небольшие отступы, смещающие вниз баннер, содержащий заголовок, чтобы он соприкасался с белой страницей, а второй стиль смещает вниз левую боковую панель, чтобы освободить достаточно места для изображения с рукой. Страница должна выглядеть так, как показано на рис. 8.24.

Замена границ изображениями

Свойство `border` (см. раздел «Добавление границ» главы 7) — полезный инструмент, но ограниченность предлагаемых языком CSS стилей границ быстро надоедает. Линия, нарисованная карандашом от руки, привлечет внимание посетителей сайта гораздо больше, нежели прямая черная. Можете смело проигнорировать свойство

border и добавить на свой вкус любую линию в виде фонового рисунка, — это очень просто. В этом практикуме мы заменим линии подчеркивания под элементами заголовков h2 собственным рисунком, похожим на рукописную линию.

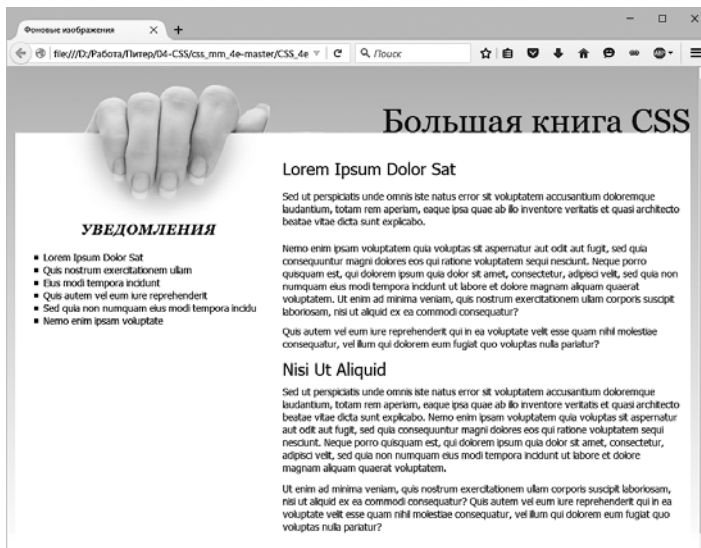


Рис. 8.24. Каскадные таблицы стилей позволяют комбинировать цвет фона и фоновое изображение. К области основного текста применен белый цвет, который выделяет текст на фоне градиента. Кроме того, изображение руки, держащей лист бумаги, вносит в дизайн страницы глубину

1. Вернитесь к файлу `styles.css` в редакторе HTML-кода. Добавьте стиль для элемента `h2` внутри контейнера `div` с классом `main`:

```
.main h2 {
  background-image: url(images/underline.png);
  background-repeat: no-repeat;
}
```

Свойство `background-image` назначает фоновое изображение заголовкам `h2`, находящимся внутри элемента с классом `main`; а значение `no-repeat` гарантирует, что изображение появится только один раз.

Если сейчас вы просмотрите файл в браузере, то увидите, что рисунок подчеркивания расположен не там, где ему положено быть. Он находится *над* заголовками!

2. Добавьте в стиль `.main h2` после свойства `background-repeat` строку со следующим кодом:

```
background-position: left bottom;
```

Мы изменили начальную позицию фонового изображения. Теперь оно выводится, начиная от левого нижнего угла элемента `h2`. Однако при просмотре

веб-страницы вы не заметите серьезных улучшений: подчеркивание сливается с текстом заголовка.

Есть простое решение. Поскольку нижняя координата фонового рисунка расположена у основания блока, образуемого элементом `h2`, необходимо всего лишь увеличить его общую высоту, чтобы сместить линию фонового рисунка немного вниз. Для этого воспользуемся небольшим отступом снизу.

3. Отредактируйте стиль `.main h2` последний раз, чтобы он выглядел таким образом:

```
.main h2 {  
  background-image: url(images/underline.png);  
  background-repeat: no-repeat;  
  background-position: left bottom;  
  padding-bottom: 7px;  
}
```

Как вы помните, отступ представляет собой промежуток между границей (или краем фона) и содержимым. Код также увеличивает суммарную высоту блока — в данном случае добавляется 7 пикселей нижнего отступа. Теперь граница изображения находится в нижней части блока `h2`, но в пустой области, созданной нижним отступом.

4. Сохраните файл и просмотрите страницу `bg_images.html` в браузере.

Теперь все элементы `h2` подчеркнуты рукописной линией. Займемся блоком боковой панели, сделаем его менее угловатым и плоским, а также улучшим вид маркированных списков.

Использование графики для маркированных списков

Стандартный маркер, используемый для списков, представляет собой черное пятнышко, что совсем не впечатляет. Но вы можете создать интересные маркеры с помощью свойства `background-image`, заменив однообразные и скучные значки собственным изображением. Первое, что необходимо сделать, — скрыть стандартные маркеры, которые предваряют элементы-пункты списка.

1. Вернитесь к файлу `styles.css` в редакторе HTML-кода. Добавьте стиль для форматирования списка пунктов левой боковой панели.

```
.announcement li {  
  list-style: none;  
}
```

Маркированный список находится внутри элемента `div` с классом `announcement`, так что этот селектор потомков воздействует только на пункты списка (элементы `li`) внутри этого контейнера `div`. Стиль удаляет маркеры. Теперь добавим наш рисунок.

ПРИМЕЧАНИЕ

Вы можете точно так же применить свойство `list-style: none`; к элементам `ul` или `ol`, чтобы удалить маркеры (или цифры) всех элементов-пунктов списка.

2. Добавьте в стиль `.announcement li` следующие два свойства:

```
background-image: url(images/bullet.png);
background-repeat: no-repeat;
```

Мы встречались с ними раньше. Одно из них добавляет фоновое изображение, а другое отменяет его повтор, чтобы рисунок отображался однократно.

При просмотре веб-страницы вы увидите, что маркеры накладываются на текст элементов списка и пункты списка отображаются очень тесно друг к другу (рис. 8.25, *вверху*). Небольшие отступы и поля исправят эту проблему.

3. Добавьте в стиль `.announcement li` еще два свойства:

```
padding-left: 25px;
margin-bottom: 10px;
```

Левый отступ добавляет пустой промежуток, смещая текст в сторону, чтобы отобразить новый значок маркера. Нижнее поле обеспечивает рассредоточение элементов списка, образуя небольшие интервалы (см. рис. 8.25, *посередине*).

Однако остался еще один недостаток. Изображение маркера чуть выступает над текстом строки, и значок выделяется над текстом пунктов списка. Это легко исправить с помощью свойства `background-position`.

4. Завершим этот стиль, добавив свойство `background-position: 0px 4px`. Законченный код стиля должен выглядеть следующим образом:

```
.announcement li {
  list-style: none;
  background-image: url(images/bullet.png);
  background-repeat: no-repeat;
  background-position: 0 4px;
  padding-left: 25px;
  margin-bottom: 10px;
}
```

Последнее изменение стиля позиционирует значок маркера в крайнюю левую позицию (это 0), отстоящую на 4 пиксела (4px) от верхнего края элемента-пункта списка, что обеспечивает совсем незначительное смещение значка, тем не менее достаточное для того, чтобы маркер выглядел безупречно.

СОВЕТ

Я рекомендую использовать именно свойство `background` вместо `list-style-image` для добавления собственных графических маркеров списков. Оно обеспечивает возможность точного позиционирования значков маркеров.

5. Сохраните файл и просмотрите веб-страницу в браузере.

Теперь пункты списка имеют трехмерные маркеры вместо скучных черных кружков (см. рис. 8.25, *внизу*).

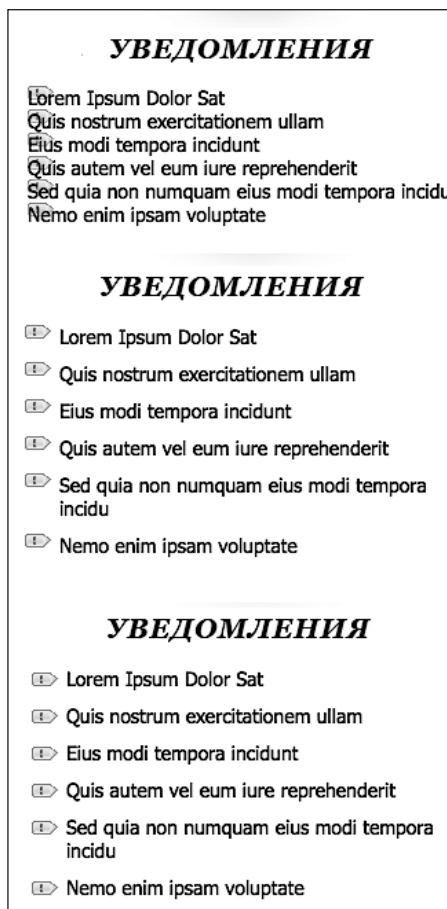


Рис. 8.25. Заменить стандартные маркеры списков своими собственными графическими значками чрезвычайно просто: всего несколько дополнительных шагов обеспечат точное и правильное размещение маркеров и текста пунктов списка

Персонализация боковой панели

На данный момент боковая панель выглядит неплохо. Текст приятно отформатирован, маркеры отлично смотрятся, но сама боковая панель теряется на белом фоне. Добавление фонового изображения позволит ее заметно выделить. Вы можете использовать отдельное изображение в виде свитка, показанное на рис. 8.23, в качестве фона элемента `div`. А чтобы убедиться, что весь текст попадает в изображение, вам следует ограничить количество содержимого, которое вы размещаете на боковой панели. Если будет слишком много текста, он не поместится в пределах изображения (верхнее левое изображение на рис. 8.12), и, наоборот, если будет слишком мало текста, у вас окажется много пустого пространства.

Более гибкий подход позволяет изображению увеличиваться по мере того, как на боковой панели увеличивается количество текста (см. рис. 8.13, *справа внизу*).

Хорошо, что эту маленькую хитрость не так сложно реализовать — вам понадобится три разных изображения и три разных стиля.

1. Вернитесь к редактору HTML-кода и файлу `styles.css`. Перейдите к стилю `.announcement` (который вы создали в шаге 6 первого раздела этого практикума) и добавьте одно свойство:

```
.announcement {
  background: url(images/scroll_top.jpg) no-repeat center top,
             url(images/scroll_bottom.jpg) no-repeat center bottom,
             url(images/scroll_middle.jpg) repeat-y center top;
  margin-top: 115px;
}
```

Да, только одно свойство (`background`), но оно содержит три изображения. Очередность отображения картинок на странице зависит от того, как они перечислены в списке. В данном случае первое изображение выводится в верхней части свитка по центру и не повторяется (благодаря параметру `no-repeat`). Второе изображение находится в нижней части свитка. Оно также повторяется один раз, но в конце элемента `div`. Наконец, средняя часть свитка (изображение `scroll_middle.jpg`) появится под двумя другими изображениями (поскольку является последним в списке) и будет повторяться по оси *Y* (по вертикали), поэтому, если элемент `div` поднимется выше, третье изображение заполнит освободившуюся область.

Если вы просмотрите страницу, то заметите несколько проблем. Во-первых, текст отображается в верхней и нижней областях свитка, которые свернуты. Небольшой отступ исправит это.

2. Обновите стиль `.announcement`, добавив верхний и нижний отступы. Внесите изменение, выделенное полужирным шрифтом:

```
.announcement {
  background: url(images/scroll_top.jpg) no-repeat center top,
             url(images/scroll_bottom.jpg) no-repeat center bottom,
             url(images/scroll_middle.jpg) repeat-y center top;
  padding: 70px 0 60px 0;
  margin-top: 115px;
}
```

Еще одна проблема проявляется в том, что маркированный список выдается как с левой, так и с правой стороны изображения боковой панели. Сделаем так, чтобы он соответствовал этому изображению, добавив правое и левое поля.

3. Перейдите к стилю `.announcement li`, созданному ранее, и добавьте два свойства в конце:

```
.announcement li {
  list-style: none;
  background-image: url(images/bullet.png);
  background-repeat: no-repeat;
  background-position: 0 4px;
  padding-left: 25px;
}
```

```
margin-bottom: 10px;  
margin-left: 30px;  
margin-right: 40px;  
}
```

Эти свойства перемещают левый и правый края каждого пункта маркированного списка на достаточное расстояние, чтобы очистить края фонового изображения.

4. Сохраните файл и просмотрите его в браузере.

Страница должна выглядеть как правое изображение на рис. 8.23. С помощью нескольких изображений и языка CSS очень легко придать жизнь скучным страницам.

9 Построение навигационной системы сайта

Можно с уверенностью сказать, что без ссылок не было бы Всемирной паутины. Возможность находиться на одной веб-странице, а затем, щелкнув кнопкой мыши, перейти на другую страницу, расположенную на компьютере на другом конце света, — вот что делает Всемирную паутину такой полезной и незаменимой. Ссылки предоставляют своеобразный способ навигации и управления содержимым сайта. Именно поэтому дизайнеры веб-страниц прилагают столько усилий для создания привлекательных ссылок, работающих должным образом.

В этой главе вы познакомитесь с форматированием ссылок, позволив им визуально выделяться из контента страниц. Вы узнаете, как обеспечить визуальные подсказки, чтобы посетители сайта могли видеть, какие ссылки они уже посещали, а какие — нет. Вы также научитесь использовать каскадные таблицы стилей для создания на веб-страницах кнопок и панелей навигации, как это делают профессиональные разработчики. И в заключение, в практической части, мы создадим полный набор средств — элементов навигации, одинаково хорошо работающих во всех разновидностях браузеров.

Выборка форматлируемых ссылок

В языке CSS, как правило, сначала нужно выбрать тот объект, для которого вы хотите определить стиль. Для этого надо сообщить CSS не только, *что* форматировать, то есть конкретный объект, но и *когда* форматировать. Браузеры отслеживают процесс взаимодействия посетителя сайта со ссылками и затем отображают их по-разному в зависимости от статуса (*состояния*). Таким образом, применяя селекторы, вам предоставляется возможность адресовать стили к ссылкам в определенном состоянии.

Состояния ссылок

Большинство браузеров распознают четыре основных состояния ссылок:

- непосещенная ссылка;
- посещенная ссылка (это означает, что по ссылке уже выполнялся переход, то есть URL-адрес сохранен в журнале браузера);

- ссылка, над которой находится указатель мыши;
- ссылка в момент нажатия.

Как описано в главе 3, каскадные таблицы стилей предоставляют четыре псевдокласса, соответствующие этим состояниям: `:link`, `:visited`, `:hover` и `:active`. Используя их, вы можете применить различное форматирование для каждого состояния ссылки, обеспечивая соответствующие подсказки и однозначно давая понять посетителю сайта, какие ссылки он уже посещал, а какие — нет.

ПРИМЕЧАНИЕ

Браузеры распознают также псевдоклассы `:focus`. Ссылка получает состояние `:focus`, когда посетители используют клавиатуру, чтобы перейти на ссылку (нажимают клавишу Tab). Этот псевдокласс также полезно применять с текстовыми полями веб-форм, что будет продемонстрировано на практике в главе 11.

Предположим, вы хотите изменить цвет текста непосещенной ссылки с синего на ярко-оранжевый. Для этого добавьте следующий стиль:

```
a:link { color: #F60; }
```

Щелкнув кнопкой мыши на этой ссылке, вы измените ее состояние на посещенное, а цвет в большинстве браузеров станет фиолетовым. Чтобы изменить его на насыщенно-красный, примените такой стиль:

```
a:visited { color: #900; }
```

СОВЕТ

Если требуется одинаково отформатировать все состояния ссылок, используйте один и тот же шрифт и размер для всех состояний, а затем отформатируйте элемент `a`, создав общий селектор `a`. Далее вы сможете использовать конкретные состояния ссылок, например `a:visited`, для изменения цвета или настройки конкретного состояния.

ВАЖНОЕ ЗАМЕЧАНИЕ

Ограничения псевдокласса `:visited`

Для обеспечения конфиденциальности браузеры накладывают ряд ограничений на свойства каскадных таблиц стилей, применяемые к псевдоклассу `:visited`. Некоторое время назад злоумышленники научились с помощью JavaScript-сценариев считывать изменения в стиле `:visited` ссылки, чтобы определить, какие сайты посещались. Например, путем загрузки нового фонового изображения для посещенных ссылок можно опреде-

лить, был ли посетитель на сайтах PayPal.com, eBay.com, BankofAmerica.com и т. д. Из-за возможности возникновения проблем наложены ограничения на изменение свойств `color`, `background-color` и `border-color` посещенных ссылок, за исключением форматирования непосещенных ссылок. То есть теперь уже нельзя рассчитывать на внесение изменений с помощью псевдокласса `:visited`.

Псевдокласс `:hover` предоставляет творческую свободу (мы изучим его в этой главе несколько позже). Он позволяет полностью изменить вид гиперссылки при наведении указателя мыши и перемещении его поверх ссылки. Если раньше для визуального представления кнопок навигации при наведении на них указателя мыши вы применяли объемный код на языке JavaScript, то вам понравится способ достижения такого же эффекта с помощью простейшей строки CSS-кода (см. раз-

дел «Форматирование ссылок» текущей главы). Для начала рассмотрим легкий пример, в котором стиль изменяет цвет ссылки при наведении и перемещении указателя мыши:

```
a:hover { color: #F33; }
```

СОВЕТ

Будьте внимательны при добавлении свойств к псевдоклассу `:hover`. Свойства, изменяющие размер элемента, поверх которого находится указатель мыши, могут повлиять на другие элементы, находящиеся рядом. Например, если вы увеличиваете размер шрифта текстовой ссылки, поверх которой находится указатель мыши, то при наведении указателя текст будет увеличиваться, выталкивая другие элементы с их позиций. Результат вам может не понравиться.

Теперь рассмотрим пример специально для «одержимых» веб-дизайнеров, которым нравится делать свои страницы непохожими на остальные. Изменим вид гиперссылки на доли миллисекунд в тот момент, когда посетитель щелкнул на ссылке. Для этого напишем следующий стиль:

```
a:active {color: #B2F511; }
```

В большинстве случаев для обеспечения максимальной гибкости требуется включить в таблицу стилей псевдоклассы `:link`, `:visited` и `:hover`. Но для того, чтобы этот метод работал, вы должны расположить ссылки в определенной последовательности: `link`, `visited`, `hover` и `active`. Ниже представлен способ добавления всех четырех стилей ссылок:

```
a:link { color: #F60; }  
a:visited { color: #900; }  
a:hover { color: #F33; }  
a:active {color: #B2F511; }
```

Если вы нарушите последовательность, то стили `hover` и `active` перестанут функционировать. Например, если поместить `a:hover` перед `a:link` и `a:visited`, то при наведении на ссылку указателя мыши ее цвет не изменится.

ПРИМЕЧАНИЕ

Почему последовательность имеет такое значение? Здесь работает механизм каскадности (см. главу 5). Все эти стили имеют одинаковый приоритет, значит, порядок их следования в программном коде определяет стиль, который будет применен в конечном счете. Поскольку ссылка одновременно может быть непосещенной и наведенной, то `a:link`, будучи указанным последним, согласно правилам каскадности имеет более высокий приоритет, и `a:hover` с иным цветом никогда не будет применен.

Выборка отдельных ссылок

Простейшие стили, которые мы создавали в предыдущем разделе, представляют собой форматирование элементов `a`. Они производили выборку в определенном состоянии, но при этом форматировали абсолютно *все* ссылки, независимо от их расположения на веб-странице. Что же делать, если вы хотите отформатировать одни ссылки одним способом, а другие — другим? Существует простое решение — применить классы к нужным элементам.

Допустим, на веб-странице имеется набор ссылок и часть из них указывает на другие сайты, которые вы хотите выделить (например, сайты ваших друзей, партнеров, спонсоров). Возможно, вы захотите их отформатировать таким образом, чтобы посетители заранее знали, что это какие-то особенные ссылки, и захотели бы перейти по ним. В данном случае можете применить такой класс:

```
<a href="http://www.hydroponiconline.com" class="sponsor">Посетите этот замечательный сайт</a>
```

Для форматирования этой ссылки другим способом создадим следующий стиль:

```
a.sponsor { font-family: Arial, sans-serif; }
a.sponsor:link { color: #F60; }
a.sponsor:visited { color: #900; }
a.sponsor:hover { color: #F33; }
a.sponsor:active {color: #B2F511; }
```

Определения только имени класса, без указания элемента `a`, будет тоже достаточно:

```
.sponsor { font-family: Arial, sans-serif; }
.sponsor:link { color: #F60; }
.sponsor:visited { color: #900; }
.sponsor:hover { color: #F33; }
.sponsor:active {color: #B2F511; }
```

Теперь такое форматирование приобретут лишь ссылки с классом `sponsor`.

ПРИМЕЧАНИЕ

В приведенных примерах в демонстративных целях мы изменили только цвет ссылок. На практике для форматирования вы можете использовать любые свойства каскадных таблиц стилей (за исключением ссылок `:visited`, как объяснялось во врезке «Важное замечание» выше). В следующем примере вы увидите, что существует множество способов креативного подхода к форматированию.

Группирование ссылок с помощью селекторов потомков

Если набор ссылок располагается в одной области веб-страницы, то вы можете сэкономить время, применив *селекторы потомков*. Предположим, у вас есть пять ссылок, которые ведут на основные разделы вашего сайта. Они представляют собой главную панель навигации, и вы хотите придать им характерный вид. Заключите эти ссылки в HTML5-элемент `nav`. Теперь появилась возможность произвести выборку и отформатировать только эти ссылки:

```
nav a { font-family: Arial, sans-serif; }
nav a:link { color: #F60; }
nav a:visited { color: #900; }
nav a:hover { color: #F33; }
nav a:active {color: #B2F511; }
```

Вместо этого можно воспользоваться элементом `div` и добавить к нему класс: `<div class="mainNav">`:

```
.mainNav a { font-family: Arial, sans-serif; }  
.mainNav a:link { color: #F60; }  
.mainNav a:visited { color: #900; }  
.mainNav a:hover { color: #F33; }  
.mainNav a:active {color: #B2F511; }
```

Использование селекторов потомков облегчает процесс форматирования ссылок, которые должны выглядеть по-разному в каждой области веб-страницы (полное описание неисчерпаемых возможностей таких селекторов вы найдете в разделе «Использование селекторов потомков» главы 18).

СОВЕТ

Применение маркированных списков для представления ссылок — очень распространенный метод (скоро вы увидите пример). В таком случае вы можете добавить идентификатор или класс к элементу `ul` списка, например `<ul class="mainNav">`, а затем создать селекторы потомков наподобие `.mainNav a:link` для их форматирования.

Форматирование ссылок

Теперь, когда вы знаете, как создавать селекторы, производящие выборку конкретных ссылок, нужно определить, как форматировать ссылки. Да как угодно! Вам предоставляется полный арсенал свойств языка CSS, а творческие способности ограничены только воображением. Единственное, что нужно учесть, — ваши ссылки должны выглядеть *как ссылки*. Они не обязательно будут синего цвета с подчеркиванием, но пусть они отличаются от остального контента веб-страниц, чтобы посетители сайта сразу могли понять, что они *кликабельны*.

Если сделать ссылку похожей на кнопку, у которой появляются границы и изменяется цвет фона при наведении на нее указателя мыши, большинство посетителей поймет, что они могут щелкнуть на ней. Можно также воспользоваться линейным градиентом для добавления к ссылке текстуры и глубины. Ссылки, расположенные в объемных фрагментах текста, должны четко выделяться. Этого можно добиться использованием полужирного начертания шрифта, сохранив традиционное подчеркивание, а также добавив стиль, изменяющий внешний вид ссылки при установке на нее указателя мыши. В качестве подсказки можно добавить рисунок (например, в виде стрелки), который визуально подскажет посетителю о том, что щелчок кнопкой мыши перенесет его в какой-либо другой раздел текущего сайта или вообще на другой сайт.

Подчеркивание ссылок

Еще с самого появления Всемирной паутины ярко-синий подчеркнутый текст сигнализировал: «Нажмите здесь, чтобы перейти туда-то». Однако стандартные атрибуты подчеркивания и цветового оформления ссылок — это то, что в первую очередь изменяет любой веб-дизайнер. Подчеркивание — слишком распространенный способ выделения, который порядком поднадоел (рис. 9.1, 1). У вас есть возможность изменить ситуацию, одновременно обеспечив хорошее оформление ссылок.

- **Полное удаление подчеркивания.** Чтобы убрать стандартное подчеркивание, используйте свойство `text-decoration` с присвоенным ему значением `none`:

```
a {text-decoration: none;}
```

Естественно, полное удаление подчеркивания может смутить посетителей сайта. Если вы не предусмотрите каких-то других визуальных подсказок, то ваши ссылки будут выглядеть точно так же, как и весь остальной текст веб-страницы (см. рис. 9.1, 2). Если вы пойдете этим путем, то обеспечьте выделение текста ссылок каким-то другим способом, например полужирным начертанием (см. рис. 9.1, 3), цветом фона, подсказкой в виде рисунка или преобразованием ссылки в имитированную кнопку.

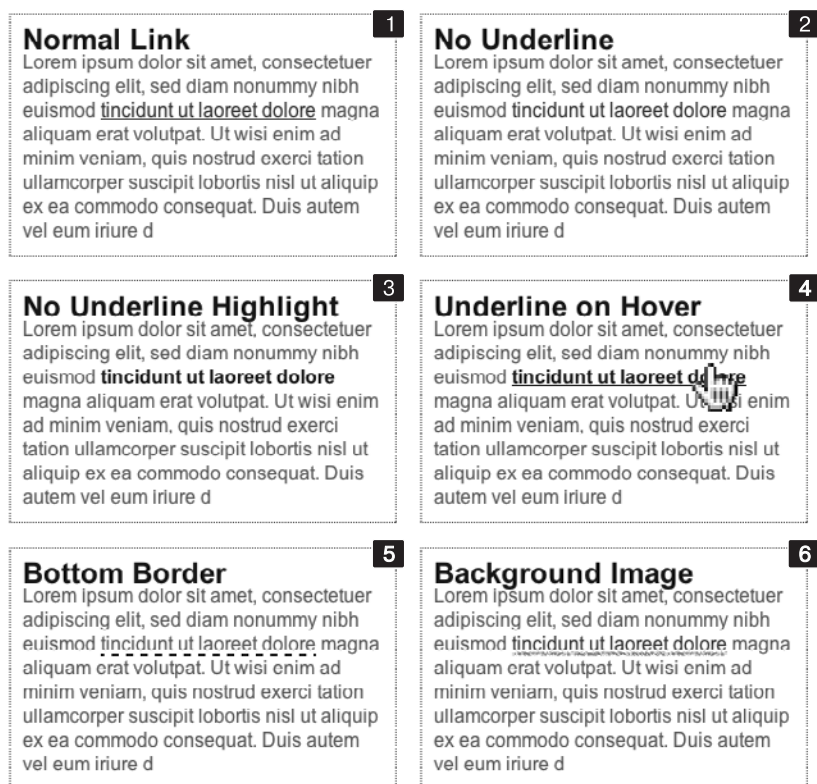


Рис. 9.1. Существует множество способов сделать стандартное подчеркивание более привлекательным

- **Добавление подчеркивания только при наведении на ссылку указателя мыши.** Некоторые веб-дизайнеры убирают подчеркивание для всех ссылок, выделяют их каким-то другим способом, а затем включают атрибут подчеркивания только при наведении указателя мыши, как показано на рис. 9.1, 4. Чтобы обеспечить такой эффект, удалите подчеркивание ссылок, а затем повторно введите его с помощью псевдокласса `:hover`:

```
a {
  text-decoration: none;
  background-color: #F00;
}
a:hover {
  background-color: transparent;
  text-decoration: underline;
}
```

- **Использование свойства border-bottom.** Так вы можете управлять цветом, толщиной и стилем стандартной линии подчеркивания ссылок, которая по умолчанию имеет толщину 1 пиксел и тот же цвет, что и текст. Большого разнообразия можно добиться путем использования вместо подчеркивания свойства border-bottom, как показано на рис. 9.1, 5. Скрыть обычное подчеркивание и добавить черту в виде пунктирной линии-границы можно следующим образом:

```
a {
  text-decoration: none;
  border-bottom: dashed 2px #9F3;
}
```

Вы можете изменить стиль, толщину линии и цвет границы. Чтобы увеличить промежуток между текстом и границей, добавьте свойство padding.

- **Использование фонового изображения.** Вы можете преобразить вид ссылок, добавив фоновый рисунок. Например, на рис. 9.1, 6 показана граница в виде рукописной линии. Подобная методика подчеркивания заголовков описана в практикуме главы 8. Напомню суть метода. Сначала создается рисунок линии подчеркивания с помощью графического редактора типа Fireworks или Photoshop, в котором имеется инструмент Кисть (Brush), имитирующий рисование мелом, фломастером и т. д. Затем создается стиль для ссылки, убирающий стандартное подчеркивание и добавляющий фоновое изображение. Оно должно быть расположено по нижнему краю ссылки и повторяться в горизонтальном направлении. Можно также добавить небольшой отступ снизу для изображения линии:

```
a {
  text-decoration: none;
  background: url(images/underline.gif) repeat-x left bottom;
  padding-bottom: 5px;
}
```

Создание кнопок

Вы можете придать ссылкам вид кнопок, присутствующих в окнах и на панелях инструментов компьютерных программ. В этом вам помогут свойства border, background-color и padding. С их помощью можно легко создавать прямоугольные кнопки разного вида (рис. 9.2).

Допустим, вы назначили класс ссылке, которую хотите отформатировать в виде кнопки: `Скидки!`. Чтобы добавить вокруг

этой ссылки простейший контур черного цвета (см. рис. 9.2, *вверху слева*), создайте следующий стиль:

```
a.button {
  border: solid 1px rgb(0,0,0);
}
```

Можете также окрасить кнопку, установив цвет фона:

```
a.button {
  border: solid 1px rgb(0,0,0);
  background-color: rgb(51,51,51);
}
```

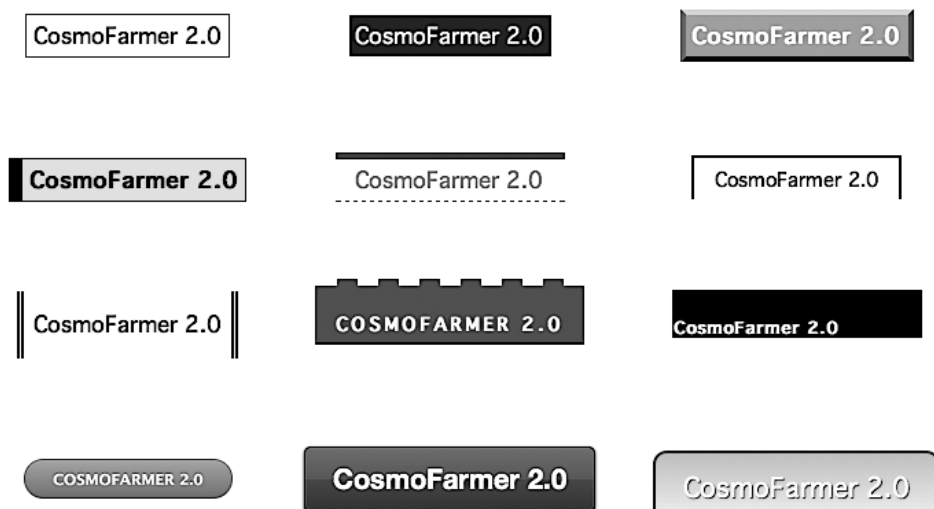


Рис. 9.2. Чтобы превратить ссылку в кнопку, на которой хочется щелкнуть, следует использовать свойства `border`, `background` и `padding`

Можно создать более впечатляющие кнопки, сочетая различные рамки или варьируя цвета, стили и ширину со свойством `background-color`. А для тех браузеров, которые поддерживают свойства `linear-gradient`, `text-shadow` и `box-shadow`, а также скругленные углы со свойством `border-radius`, можно добиться для ссылок весьма привлекательного вида (нижний ряд на рис. 9.2).

ПРИМЕЧАНИЕ

В этих примерах и `a.button`, и `.button` подойдут в качестве имени стиля. В случае `a.button` стиль применяется только к элементам `a` с классом `button`, в то время как `.button` относится к любому элементу с этим именем класса. Если вы хотите быть уверенными, что стиль применяется только к конкретному элементу, то добавьте имя элемента в начале. Добавление имени элемента также полезно при просмотре CSS-кода — оно дает понять, для форматирования какого контента предназначен стиль. Когда вы видите `a.button`, становится ясно, что стиль нацелен на определенные ссылки.

Имейте в виду: совсем не обязательно, чтобы линии границ со всех четырех сторон ссылки-кнопки были одинакового стиля. Возможно, что у ссылки даже не будет каких-либо границ. Широко применяемая дизайнерская методика превращения плоской кнопки в объемную предполагает использование четырех границ разного цвета, как показано в верхнем правом углу на рис. 9.2. Придание кнопке объемности не представляет сложности, но вы должны помнить, что освещение заставляет предмет выглядеть таким. Представьте, что свет падает на одну из четырех сторон; эта обращенная к источнику света сторона — самая светлая, а противоположная — самая темная (поскольку приподнятая кнопка в ненажатом состоянии препятствует прохождению света и вызывает появление «тени»). Остальные две стороны должны иметь цвета промежуточных оттенков между «светлыми» и «темными». Рассмотрим CSS-код для придания кнопке-ссылке, отображенной в верхнем правом углу на рис. 9.2, объемного вида:

```
a.button {
  background: #B1B1B1;
  color: #FFF;
  font-weight: bold;
  border-width: 4px;
  border-style: solid;
  border-top-color: #DFDFDF;
  border-right-color: #666;
  border-bottom-color: #333;
  border-left-color: #858585;
}
```

Вы также можете (и это рекомендуется) создать стиль для кнопки в состоянии `:hover`. Кнопки будут реагировать на наведение посетителем на ссылку указателя мыши, обеспечивая интерактивность и обратную связь. В случае с объемной кнопкой при ее нажатии очень эффективен метод инверсной смены цветов: темный фон кнопки должен стать светлым, светлая граница — темной и т. д.

По-настоящему добавить глубину кнопке можно, используя линейные градиенты и скругленные углы, блочные и текстовые тени. Например, нижняя средняя кнопка на рис. 9.2 создана с помощью следующего кода:

```
background-color: #ee432e;
background-image: -webkit-linear-gradient(top, #ee432e 0%, #c63929 50%, #b51700 50%, #891100 100%);
background-image: -moz-linear-gradient(top, #ee432e 0%, #c63929 50%, #b51700 50%, #891100 100%);
background-image: -o-linear-gradient(top, #ee432e 0%, #c63929 50%, #b51700 50%, #891100 100%);
background-image: linear-gradient(top, #ee432e 0%, #c63929 50%, #b51700 50%, #891100 100%);
border: 1px solid #951100;
border-radius: 5px;
box-shadow: inset 0px 0px 0px 1px rgba(255, 115, 100, 0.4), 0 1px 3px #333333;
padding: 12px 20px 14px 20px;
text-decoration: none;
color: #fff;
```

```
font: bold 20px/1 "helvetica neue", helvetica, arial, sans-serif;
text-align: center;
text-shadow: 0px -1px 1px rgba(0, 0, 0, 0.8);
```

В строке 1 устанавливается темно-красный цвет фона; этот цвет предоставляется для браузеров, не поддерживающих линейные градиенты (Internet Explorer 9 и более ранние версии, а также другие устаревшие браузеры). В строках 2–5 задается линейный градиент для различных браузеров. В строке 6 добавляется простая граница, а в строке 7 создаются скругленные углы, после чего в строке 8 добавляется тонкая внутренняя тень и вторая, внешняя тень ниже кнопки, то есть применяются две тени, что вполне допустимо. В строке 9 добавляется отступ, обеспечивающий появление пустого пространства вокруг текста кнопки, а в строке 10 удаляется подчеркивание, которое обычно появляется под ссылками. В последних строках устанавливаются свойства для текста в кнопке, включая цвет, шрифт, выравнивание и текстовую тень.

СОВЕТ

Создание привлекательных кнопок часто требует целого набора свойств каскадных таблиц стилей (что можно заключить из приведенного выше кода). Если в создании всего этого кода требуется помощь, обратитесь к генератору кнопок CSS по адресу bestcssbuttongenerator.com. Чтобы взглянуть на другие впечатляющие CSS-кнопки, посетите сайты tinyurl.com/nmfmrxc и tinyurl.com/7jsqkb5.

Применение изображений

Добавление к ссылкам изображений — один из самых легких и эффективных способов улучшить внешний вид элементов навигации сайта. Существует множество методик и вариантов дизайна, но при этом следует заметить, что ни в одном из грамотных методов не применяется HTML-элемент `img`. Вместо этого используется свойство `background-image`, с помощью которого можно повысить привлекательность любой ссылки. Несколько примеров приведено на рис. 9.3 (более совершенные методы использования изображений для создания графических кнопок и эффектов ролловеров¹ описаны в практике этой главы).

Чтобы вспомнить свойство `background-image` и все, что с ним связано, вернитесь к разделу «Добавление фоновых изображений» главы 8. Пока же напомним вам несколько моментов, которые нужно иметь в виду при использовании изображений со ссылками.

- **Не забывайте про значение `no-repeat`.** По умолчанию фоновые изображения выводятся на заднем плане формируемого элемента. Со многими рисунками, применяемыми для ссылок, это приводит к плохому результату (см. рис. 9.3, *вверху справа*). Если вы не используете едва заметный узор, похожий на градиентную заливку, не забудьте отключить повторное отображение фонового рисунка: `background-repeat: no-repeat;`
- **Позиционируйте с помощью свойства `background-position`.** Чтобы точно поместить фоновое изображение, используйте свойство `background-position` (см. раздел

¹ Ролловер — это элемент веб-страницы, изменяющий свой вид в зависимости от внешнего воздействия.

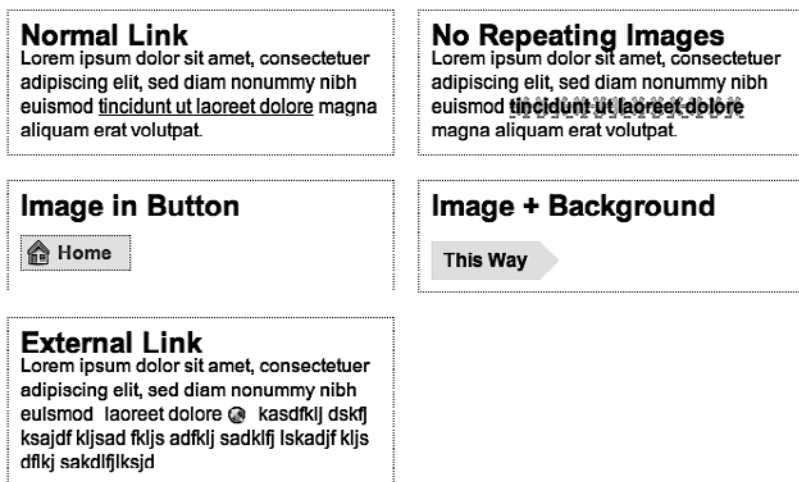


Рис. 9.3. Даже простой рисунок может оживить ссылку и сделать ее более понятной для посетителя: значок земного шара (внизу слева) — один из способов обозначения ссылок во Всемирной паутине

«Позиционирование фоновых изображений» главы 8). Если необходимо позиционировать изображение по правому краю ссылки, но при этом центрировать его вертикально на строке, то добавляйте следующий CSS-код: `background-position: right center;`.

Для еще более точного размещения изображений используйте значения параметров в пикселах или единицах `em`. Эти единицы измерения упрощают смещение рисунка на нужное расстояние от левого края ссылки. Комбинируя их с процентными значениями, вы сможете не только легко центрировать изображение вертикально по отношению к тексту ссылки, но и обеспечить его небольшое смещение на точное расстояние от левого края: `background-position: 10px 50%;`.

СОВЕТ

При позиционировании фоновых изображений первое значение свойства `background-position` определяет положение по горизонтали (по направлению слева направо); а второе — по вертикали (по направлению сверху вниз).

Увы, но не существует способа точно позиционировать изображение по отношению к правому или нижнему краю ссылки. Поэтому, если вы хотите поместить рисунок на небольшом расстоянии от правого края, доступно два варианта. Первый заключается в том, чтобы в графическом редакторе добавить с правой стороны рисунка пустое пространство. Размер этого отступа должен быть равен тому расстоянию, на которое вы хотите сместить изображение от правого края ссылки. После этого для его выравнивания по правому краю формируемого элемента используйте свойство `background-position`; например `background-position: right top;`. Кроме того, можете использовать процентные значения: `background-position: 90% 75%;` — такие параметры обеспечат позиционирование

точки изображения, находящейся на расстоянии 90 % от его левого края, на расстоянии 90 % от левого края формируемого элемента. Однако такой метод не обеспечивает точности позиционирования, так что придется немного поэкспериментировать (о том, как работает позиционирование с процентными значениями, написано в разделе «Позиционирование фоновых изображений» главы 8).

- **Добавляйте отступы с помощью свойства `padding`.** Если вы используете для выделения ссылки изображение или значок, нужно добавить отступ с той стороны, где рисунок примыкает к тексту. Так, в примере 3 на рис. 9.3 ссылка имеет отступ слева шириной 30 пикселей для предотвращения наложения текста (слово Home) ссылки на изображение дома, в то же время, как показано на нижнем изображении слева, небольшой отступ справа обеспечивает пространство для отделения текста ссылки от значков земного шара.

ПРИМЕЧАНИЕ

Поскольку элемент `a` — строчный, добавление верхнего или нижнего отступов (или полей) не окажет никакого эффекта. Причина такого поведения описана в подразделе «Отображение строчных и блочных элементов» раздела «Управление размерами полей и отступов» главы 7. Однако можно преобразовать ссылку в блочный элемент (`display: block`), чтобы к нему можно было применить верхние и нижние отступы и поля, или воспользоваться свойством `inline-block` (`display: inline-block`). Оба этих метода будут рассмотрены позже.

- **Пользуйтесь псевдоклассами.** Не забывайте о псевдоклассах `:hover` и `:visited`. Они помогут создать для ссылок замечательные динамические эффекты и обеспечить полезную обратную связь. Можно использовать *разные* фоновые рисунки для каждого из этих псевдоклассов, чтобы, например, изображение непосещенной ссылки в виде выключенной лампочки при наведении на нее указателя мыши сменялось на изображение включенной.

Если вы решите использовать изображение для ссылки в состоянии `:hover`, помните, что браузеры не загрузят рисунок, пока посетитель сайта на самом деле не наведет на него указатель мыши. Может возникнуть значительная задержка, прежде чем появится изображение. Однако уже после первой загрузки задержка исчезнет. О том, как решить данную проблему, читайте в подразделе «Использование ролловеров» далее в этой главе.

Создание панелей навигации

Каждому сайту требуется удобная панель управления с элементами навигации: во-первых, чтобы привести посетителей к нужной им информации, а во-вторых, чтобы им понравилось управление сайтом и они вернулись сюда вновь. Контент большинства сайтов организован в виде разделов. Например, продукция, контактная информация, журнал отзывов и т. д. Такая организация позволяет посетителям успешно ориентироваться в содержимом сайта, и они точно знают, что и где смогут найти. В большинстве случаев ссылки на основные разделы сайта можно найти на *панели навигации*. Каскадные таблицы стилей облегчают создание красивых и функциональных панелей, ролловеров и т. д.

Использование маркированных списков

По сути, панель навигации — не что иное, как группа ссылок, точнее, *список* разделов сайта. Как разъясняется в главе 1, задача языка HTML состоит в том, чтобы обеспечить структурирование. Следовательно, вы всегда должны использовать HTML-элементы в соответствии с их функциональным назначением, чтобы они соответствовали заключенному в них содержанию. К примеру, для списка элементов это должен быть `ul`, элемент неупорядоченного списка. Не имеет значения, будет список вообще *без* маркеров или будет располагаться горизонтально вдоль верхнего края веб-страницы: все форматирование элемента `ul` обеспечено средствами языка CSS. Пример показан на рис. 9.4.

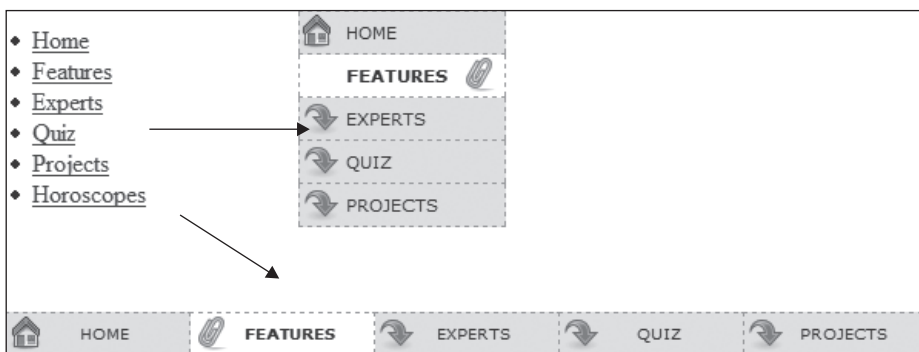


Рис. 9.4. Используя каскадные таблицы стилей, вы можете превратить обычные элементы `ul` в, например, вертикальное или горизонтальное управляющее меню (панель навигации)

Применение HTML для создания панели навигации — это и есть использование языка разметки по прямому назначению. В каждом элементе списка имеется по одной ссылке. Кроме того, нам всего лишь необходимо форматировать этот маркированный список (*не нужно*, чтобы его пункты составляли панель навигации). Правильным подходом будет применение класса или идентификатора к элементу `ul`:

```
<ul class="nav">
<li><a href="index.html">Главная</a></li>
<li><a href="news.html">Новости</a></li>
<li><a href="reviews.html">Обзоры</a></li>
</ul>
```

CSS-код немного отличается в зависимости от того, какая нужна панель навигации — горизонтальная или вертикальная. В любом случае вы должны сделать два шага.

1. **Удалить маркеры.** Чтобы панель навигации не была похожа на маркированный список, удалите маркеры, присвоив свойству `list-style-type` значение `none`:

```
ul.nav {
  list-style-type: none;
}
```

2. **Убрать отступы и поля.** Поскольку браузеры сами создают отступы для элементов списка, нам придется избавиться от них. Однако одни браузеры используют для этих целей свойство `padding`, а другие — `margin`, поэтому нам нужно присвоить обоим свойствам значение 0:

```
ul.nav {  
  list-style-type: none;  
  padding-left: 0;  
  margin-left: 0;  
}
```

По сути, эти два шага обеспечивают всем элементам списка простой вид, как у обычного блочного элемента — абзаца или заголовка (с одним исключением: браузер не добавляет полей между элементами списка). С этого момента можно начинать форматирование. Если вам нужна вертикальная панель навигации, продолжайте читать дальше; если же хотите создать горизонтальную панель управления, то пропустите следующий раздел.

Вертикальные панели навигации

Вертикальная панель навигации — это всего лишь набор ссылок, расположенных одна за другой в виде списка или стека. Удалив маркеры, а также поля и отступы слева (как описано в предыдущем разделе), вы выполните большую часть работы, но вы должны знать, что существует еще несколько дополнительных хитростей, обеспечивающих правильное отображение панели навигации.

1. Отобразите ссылку в виде блочного элемента.

Поскольку элемент `a` — строчный, ссылка будет занимать по ширине пространство, равное ширине содержимого элемента. Кнопки со ссылками в виде текста различной ширины (например, **Главная** и **Продукция**) будут иметь различную ширину. Ступенчатое вертикальное отображение кнопок друг над другом выглядит не очень хорошо (рис. 9.5, 1). Кроме того, верхние и нижние отступы и поля не оказывают никакого эффекта на строчные элементы. Чтобы обойти эти ограничения, преобразуем ссылки в блочные элементы:

```
ul.nav a {  
  display: block;  
}
```

Значение `block` не только выравнивает все кнопки по ширине, но и делает всю область ссылок активной при щелчке по ней кнопкой мыши, подобно настоящим кнопкам. Это тот случай, когда посетители щелкают на области кнопки, где нет текста (например, отступы вокруг), но ссылка по-прежнему работает.

2. Ограничьте ширину кнопок.

Преобразование пунктов списка в блочные элементы обеспечивает ссылкам ширину элементов, в которые они вложены. Поскольку ссылки просто помещены на веб-страницу, их ширина окажется равной ширине окна браузера (см. рис. 9.5, 2). Есть несколько способов ограничить ширину ссылок. Сначала можно установить

ширину элемента `a`. Например, если вы хотите, чтобы все кнопки имели ширину 8 em, добавьте в стиль свойство `width` следующим образом:

```
ul.nav a {
  display: block;
  width: 8em;
}
```



Рис. 9.5. Всего четырьмя простыми действиями можно превратить маркированный список ссылок в привлекательную панель навигации

Установка ширины любого элемента, в который заключены ссылки, например `li` или `ul`, приведет к тому же результату.

Если текст кнопки занимает всего одну строку, то можно его центрировать вертикально, чтобы над и под ним были одинаковые промежутки. Укажите высоту ссылки `a` и назначьте такое же значение свойства межстрочного интервала:

```
a {
  height: 1.25em;
  line-height: 1.25em;
}
```

ПРИМЕЧАНИЕ

Не обязательно явно назначать свойство `width`, если панель навигации вложена в элемент разметки веб-страницы, для которого уже установлена ширина. В части III вы узнаете, каким образом элементарно создать боковую панель, примыкающую к левому или правому краю веб-страницы.

Боковая панель имеет свою ширину, и помещенный в нее маркированный список со ссылками-кнопками автоматически приобретает эту ширину.

Теперь, когда мы закончили всю эту малопродуктивную работу, займемся основной задачей — форматированием кнопок. Можно добавить отступы, поля, изображения, изменить фоновый цвет. Если хотите расположить кнопки так, чтобы они не соприкасались между собой, можете добавить нижнее (или верхнее) поле к каждой из ссылок.

Горизонтальные панели навигации

Каскадные таблицы стилей позволяют преобразовать список ссылок, расположенных вертикально одна за другой, в список с горизонтальным представлением, как было показано на рис. 9.4. В этом разделе описываются два распространенных подхода к созданию из списка горизонтальной панели навигации. Первый метод основывается на использовании свойства `display: inline` — он прост, но обычно приводит к появлению маленьких расстояний между кнопками. Проблема решаема, но для этого придется возиться с HTML-кодом. Если нужна горизонтальная навигационная панель с соприкасающимися кнопками, обратитесь к методу с применением плавающих элементов `ul`.

ОБХОДНОЙ ПРИЕМ

Если границы соединяются

Если на вертикальной панели навигации кнопки расположены вплотную друг к другу, то линии границ соседних ссылок сливаются между собой и превращаются в двойные. Другими словами, граница одной кнопки касается границы следующей кнопки.

Чтобы избавиться от этого, можно добавить границу только к *верхнему* краю ссылок. Таким образом, мы получим всего одну границу, отделяющую кнопки, расположенные друг над другом.

Однако в результате этого обходного приема последняя, нижняя ссылка панели навигации получается без границы. Чтобы решить проблему, можно создать класс со стилем нижней границы и применить

его к последней ссылке, но при этом придется дописывать HTML-код. Другой вариант — добавить нижнюю границу к элементу `ul`, завершающему панель навигации. Или можете использовать селектор `:last-of-type` для добавления нижней границы к последней ссылке (этот прием будет продемонстрирован на практике этой главы).

То же самое справедливо и для горизонтальных ссылок. В этом случае границы левой и правой кнопок касаются друг друга. Удалите левую границу у ссылок и добавьте ее, к примеру, к элементу-контейнеру `ul` или используйте селектор `:first-of-type`, чтобы выделить первую ссылку на панели и добавить границу к ее левому краю.

Какой бы метод вы ни использовали, сначала удалите маркеры и пространство слева из элементов `ul`, как показано на рис. 9.6, 1.

Использование свойств `display:inline` и `display:inline-block`

Самый простой метод создания горизонтальной панели навигации заключается в смене для элементов списка значения `block` (блочный элемент) свойства `display`

на `inline` (строчный элемент). Это делается средствами каскадных таблиц стилей.

1. Создайте стиль для маркированного списка, чтобы удалить отступы, поля и маркеры.

```
ul.nav {
  margin-left: 0px;
  padding-left: 0px;
  list-style: none;
  border-bottom: 1px dashed #000;
}
```

В данном случае добавляется нижняя граница, которая появится ниже кнопок (см. рис. 9.6, 1).

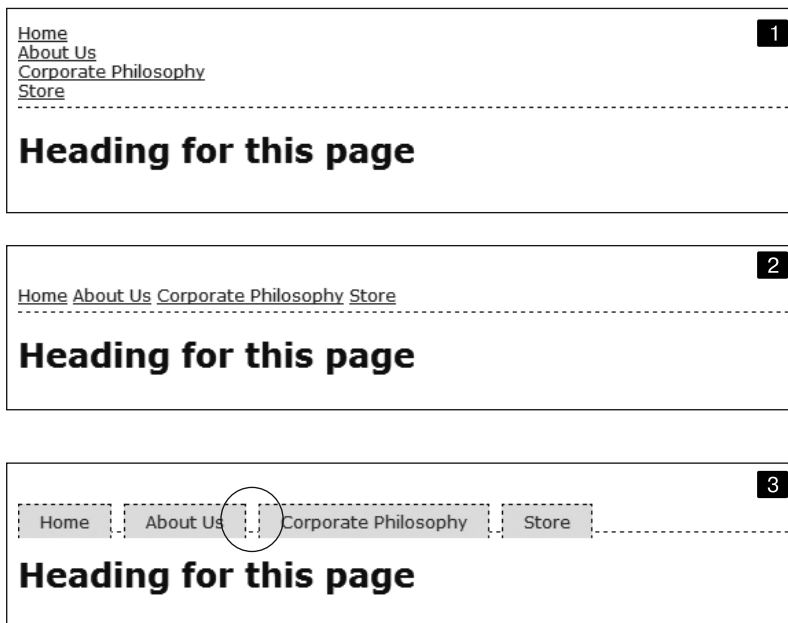


Рис. 9.6. Создание горизонтального меню из списка ссылок потребует выполнения всего нескольких действий

2. Преобразуйте пункты списка в строчные элементы.

Строчные элементы не создают разрывов строк ни перед элементом, ни после него, как это делают блочные. Присвоение значения `inline` свойству `display` элементов `li` приведет к отображению элементов списка в одну строку (см. рис. 9.6, 2).

```
.nav li { display: inline; }
```

Вы должны быть уверены в том, что панель навигации кнопок-ссылок не слишком большая. Те кнопки, которые не поместятся в одну строку, перенесутся на другую.

3. Отформатируйте ссылки.

Можно убрать подчеркивание под ссылками и вместо этого добавить вокруг них окаймляющую границу. Чтобы обеспечить визуальную глубину и реалистичность кнопок, надо изменить цвет фона или добавить рисунок. При необходимости добавьте отступы вокруг текста ссылок. Если требуется разнести кнопки друг от друга, установите для них правое поле. Следующий стиль обеспечивает ссылкам визуальное подобие кнопок, как показано на рис. 9.6, 3:

```
.nav a {
  display: inline-block;
  border: 1px dashed #000;
  border-bottom: none;
  padding: 5px 15px 5px 15px;
  background-color: #EAEAEA;
  text-decoration: none;
  color: #333;
}
```

Сначала нужно присвоить свойству `display` ссылок значение `inline-block`. После этого верхние и нижние отступы и поля будут соответствовать требованиям (как уже упоминалось, обычно строчные элементы игнорируют верхние и нижние отступы и поля, а также значения ширины и высоты). Затем можно будет указать стиль кнопок в соответствии с вашим вкусом. Здесь к ссылкам добавляется граница и удаляется нижняя граница, чтобы исключить дублирование нижней границы, применяемой к маркированному списку.

СОВЕТ

Определив ширину ссылок, все кнопки можно сделать одинакового размера.

Чтобы эта навигационная панель отображалась в центре страницы, к стилю элемента `ul` нужно добавить объявление `text-align: center;`. У такой технологии по сравнению с рассматриваемой в следующем разделе есть одно преимущество — при использовании значений `inline` и `inline-block` панель навигации можно отцентрировать, чего нельзя сделать с `float`.

Но можно заметить, что кнопки не соприкасаются (см. область, выделенную на рис. 9.6). Это связано с тем, как браузеры рассматривают пустое пространство между элементами `li`. Возьмем, например, следующий фрагмент HTML-кода:

```
<ul class="nav">
<li><a href="index.html">Главная</a></li>
<li><a href="news.html">Новости</a></li>
<li><a href="reviews.html">Обзоры</a></li>
</ul>
```

Пробельные символы (символ табуляции, возврата каретки или пробел) между закрывающим тегом `` и следующим открывающим тегом `` браузеры рассматривают как пустое пространство. Чтобы его удалить, доступны два способа.

- Разместить закрывающий тег `` и открывающий тег `` на одной и той же строке:

```
<ul class="nav">
<li><a href="index.html">Главная</a></li><li>
<a href="news.html">Новости</a></li><li>
<a href="reviews.html">Обзоры</a></li>
</ul>
```

Обычно так код не записывают, и такие HTML-редакторы, как Dreamweaver, так бы не сделали. Чтобы удалить пустое пространство, нужно внести изменения в код вручную.

- Добавить к элементам списка правое поле с отрицательным значением. Например, в стиль `li` в показанном выше шаге 2 можно внести следующее изменение:

```
.nav li {
  display: inline;
  margin-right: -5px;
}
```

Отрицательное значение поля приводит к наложению следующего элемента списка на 5 пикселей, закрывая промежуток между кнопками. Проблема при использовании этой технологии заключается в том, что конкретное используемое значение варьируется в зависимости от размера текста — 5 пикселей могут сработать, а могут — и нет, поэтому для получения нужного значения следует провести эксперимент.

Панели навигации на основе плавающих элементов

Другой, более популярной, технологией является применение плавающих элементов списка. Такой метод также позволяет поместить ссылки бок о бок. Кроме того, эта технология не страдает от проблемы пустого пространства, присущей строчному методу.

ПРИМЕЧАНИЕ

Панели навигации на основе плавающих элементов трудно центрировать горизонтально посередине веб-страницы. Если требуется такое выравнивание, то лучше пользоваться методом со строчными элементами, описанным в этой главе, или методом `flexbox`, показанным в практикуме к главе 17.

1. Преобразуем пункты списка в плавающие элементы.

Добавление к элементам `li` свойства `float` с выравниванием по левому краю исключает их из обычного нисходящего потока элементов:

```
.nav li { float: left; }
```

Плавающие элементы списка (вместе с вложенными ссылками) располагаются на одной строке, точно так же, как изображения фотогалереи из практикума главы 8 (при необходимости можно с такой же легкостью установить выравнивание кнопок по правому краю экрана (окна браузера) или боковой панели, в которую заключены кнопки).

ЧАВО

Всплывающие меню

Как создать стильные современные всплывающие меню, которые показывают подменю со ссылками при наведении указателя мыши на кнопки панели навигации?

Панели навигации в виде всплывающих многоуровневых меню чрезвычайно популярны. Они представляют собой отличный способ вместить огромное количество ссылок в компактную панель навигации. Вы можете создать их несколькими способами.

Один из методов создания всплывающих меню использует только средства каскадных таблиц стилей. Основное руководство по процессу создания можно найти по адресу tinyurl.com/o7bhpw5. Как создать изящное меню с CSS-переходами для появления и исчезновения всплывающего меню, можно узнать по адресу tinyurl.com/kv9bg6v.

Если вы не хотите делать меню самостоятельно или ограничены во времени, то можете воспользоваться бесплатным мастером, реализованным в виде веб-страницы, которая автоматически создает необходимый код HTML и CSS (purecssmenu.com).

Подход с применением исключительно средств каскадных таблиц стилей имеет один недостаток: когда посетитель убирает с кнопки-ссылки указатель мыши, подменю исчезает мгновенно, без задержки — остается надеяться, что у посетителей сайта достаточно хорошая реакция. Есть и другой подход: используйте CSS-код для форматирования кнопок-ссылок и JavaScript-код для управления работой подменю. Пример такого использования JavaScript-кода находится по адресу tinyurl.com/qhuupeb.

2. Добавьте в стиль ссылок свойство `display: block`.

Ссылки — это строчные элементы, и значения ширины (как верхние и нижние отступы и поля) к ним неприменимы. Преобразование ссылок в блочные элементы позволит установить точную ширину кнопок и добавить требуемые промежутки для ссылок:

```
.nav a { display: block; }
```

3. Отформатируйте ссылки.

Измените цвет фона, добавьте границы и т. д.

4. Укажите ширину.

Если необходимо, чтобы все кнопки панели управления имели одинаковую ширину, укажите ширину элемента `a`. Точное значение ширины зависит от максимальной длины текста на каждой из кнопок. Очевидно, что для ссылки «Миссия корпорации» нужна кнопка большей ширины. Если вы хотите, чтобы каждая кнопка имела ширину находящегося внутри нее текста, не указывайте значение ширины. Хотя можно добавить отступы слева/справа и таким образом задать для текста немного пространства, избавившись от тесноты.

СОВЕТ

Чтобы расположить текст ссылок по центру кнопок, добавьте в стиль ссылок свойство `text-align: center;`

5. Добавьте свойство `overflow: hidden` в стиль элемента `ul`.

В случае если для элемента `ul` заданы границы, цвет фона или изображение, вам придется «удержать плавающий элемент», то есть плавающие пункты

списка внутри элемента `ul` будут высываться из-под нижней части списка (и в таком случае находиться за пределами границ и фонового цвета элемента `ul`).

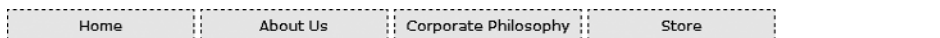
```
.nav {
  overflow: hidden;
}
```

Ниже представлен код стилей, требуемых для создания панели навигации, изображенной на рис. 9.7. Обратите внимание, что кнопки имеют одинаковую ширину и текст в них центрирован.

```
.nav {
  margin: 0px;
  padding: 0px;
  list-style: none;
  border-bottom: 3px solid rgb(204,204,204);
  overflow: hidden;
}

.nav li {
  float: left;
}

.nav a {
  width: 12em;
  display: block;
  border: 3px solid rgb(204,204,204);
  border-bottom: none;
  border-radius: 5px 5px 0 0;
  padding: 10px;
  margin-right: 5px;
  background-color: rgb(95,95,95);
  background-image: -webkit-linear-gradient(rgb(175,175,175), rgb(95,95,95));
  background-image: -moz-linear-gradient(rgb(175,175,175), rgb(95,95,95));
  background-image: -o-linear-gradient(rgb(175,175,175), rgb(95,95,95));
  background-image: linear-gradient(rgb(175,175,175), rgb(95,95,95));
  text-decoration:none;
  color: white;
  text-align: center;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: bold;
}
```



Heading for this page

Рис. 9.7. Плавающие элементы списка позволяют создавать кнопки панели навигации одинаковой ширины. Вы можете использовать CSS-код, приведенный выше

Использование ролловеров

Очень давно, когда еще не было языка CSS, для смены одной графической ссылки на другую при наведении на нее указателя мыши приходилось привлекать средства JavaScript. Теперь с помощью каскадных таблиц стилей вы можете достичь подобного эффекта посредством псевдокласса `:hover` и фонового изображения. Однако с этим методом есть одна проблема: если браузером не загружено сменное изображение, оно не появится, пока на данную ссылку не будет наведен указатель мыши, — для посетителя заметна задержка этого сменного изображения на время загрузки браузером. Задержка произойдет всего один раз, во время первого наведения указателя мыши посетителем на ссылку, но все-таки ожидание загрузки графики — это издержки XX века.

Метод с применением JavaScript-сценария поможет избежать этой проблемы благодаря возможности *предварительной* загрузки сменных изображений задолго до того, как они понадобятся. В CSS нет таких средств, поэтому придется задействовать другой хитрый трюк, называемый *CSS-спрайтом*, который использует единственное изображение для представления различных состояний одной навигационной кнопки.

ПРИМЕЧАНИЕ

Эволюционировавший метод CSS-спрайтов в настоящее время широко используется такими компаниями, как Yahoo! и Google, причем не только для ролловеров, но и для оптимизации скорости загрузки сайтов. Вы можете прочитать об этом больше на странице tinyurl.com/neбу3km. По адресу tinyurl.com/оуmwd8 находится список инструментов, примеров использования и полезных приложений для работы со спрайтами.

Рассмотрим, как реализуется метод.

1. В программе редактирования изображений создайте один рисунок с различными вариантами отображения кнопки.

Нужно создать изображения, представляющие собой различные состояния кнопки: обычное состояние, состояние при наведении указателя мыши и при необходимости состояние нажатой кнопки. Далее поместите изображения вертикально одно над другим. У нас будет два состояния кнопки: изображение в обычном состоянии должно располагаться сверху, а изображение-ролlover при наведении указателя мыши — снизу.

ПРИМЕЧАНИЕ

Существует множество онлайн-инструментов для создания CSS-спрайтов, например SpritePad (tinyurl.com/6ut2p7j) или Sprite Cow (spritecow.com).

2. Измерьте расстояние от верхнего края получившегося комбинированного изображения до верхнего края каждого следующего изображения.

На рис. 9.8, *вверху*, верхняя граница сменного изображения-роллера имеет смещение от верхнего края общего изображения 39 пикселей.

3. Создайте стиль для ссылки в обычном ненажатом состоянии. Поместите изображение в качестве фонового в левый верхний угол формируемой ссылки (см. рис. 9.8, *посередине*).

Стиль может выглядеть следующим образом:

```
a { background: #E7E7E7 url(images/pixy.png) no-repeat left top; }
```

4. Создайте стиль :hover.

Примените свойство `background-position`, чтобы сместить изображение *вверх*. Таким образом, первое изображение исчезнет, а второе, нижнее изображение-ролlover окажется видимым (см. рис. 9.8, *внизу*).

```
a:hover { background-position: 0 -39px; }
```

Такая методика, помимо предотвращения длительной задержки в загрузке дополнительного изображения-роллера, обеспечит хранение всех изображений кнопки, отражающих ее состояния, в единственном файле.

ПРИМЕЧАНИЕ

Некоторые сайты довели использование этого метода до крайности. Ресурсы Yahoo!, Amazon и Google (и многие другие) часто собирают вместе в одном файле множество маленьких изображений и показывают лишь часть файла, содержащую необходимую кнопку. Вы можете посмотреть пример спрайта с сайта Amazon по адресу tinyurl.com/o9p9oob. Сайты с миллионами посещений в день активно используют сложные спрайты, поскольку веб-серверу проще и быстрее отправить один (хотя и больший по размеру) файл изображения, чем десяток меньших по размеру файлов.

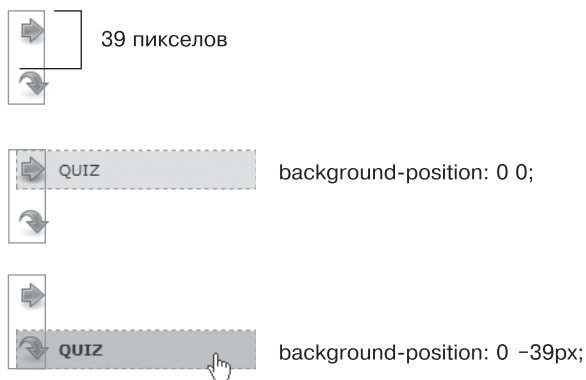


Рис. 9.8. Используя метод CSS-спрайтов, можно избежать задержки во время загрузки браузером рисунка-роллера в первый раз, поместив все изображения различных состояний кнопки-ссылки в единственный графический файл

Форматирование ссылок определенного типа

Веб-дизайнеры используют ссылки на все что угодно: на другие веб-страницы своих сайтов, веб-страницы посторонних сайтов, PDF-файлы, документы Word, ZIP-архивы и многое другое. Чтобы помочь ориентироваться посетителям сайта, вы можете подсказать им, к какому объекту ведет ссылка, прежде чем они щелкнут на ней. Расширенные селекторы представляют отличный способ решения данной проблемы.

Ссылки на другие сайты

Вы можете легко создать стиль, определяющий ссылки на другие сайты, используя селектор атрибута. Как рассказывалось в главе 3, селекторы атрибутов позволяют форматировать HTML-элементы, у которых есть определенный атрибут, например изображение `img` с атрибутом `alt` со значением `Наша компания`. Вы также можете форматировать элементы, атрибуты которых *содержат* определенные значения. Любая ссылка, которая ведет за пределы вашего сайта, должна быть абсолютным URL-адресом, то есть должна начинаться с `http://` или `https://`, например `https://css-tricks.com`.

ПРИМЕЧАНИЕ

В настоящее время все крупные сайты используют протокол SSL (secure sockets layer — уровень защищенных сокетов), поэтому ссылки на такие сайты всегда начинаются со значения `https://`. На самом деле протокол SSL применяется практически повсеместно, поэтому вы, вероятно, чаще встречаете ссылки `https://`, чем `http://`.

Поскольку вам необходимо учесть ссылки как `http://`, так и `https://`, существует два выхода, каждый из которых обладает недостатком. Сначала вы могли бы заполнить поиск начала свойства `href` для `http` следующим образом:

```
a[href^='http']
```

Эта запись соответствует URL-адресу, который начинается как с `http://`, так и с `https://`. Тем не менее данный способ также выбирает относительные ссылки, которые указывают на адрес `http.html` или `http-information.html`. Если вы не планируете ссылаться на любые страницы своего сайта, которые начинаются с `http`, все будет работать прекрасно.

Другой способ заключается в применении селектора для поиска значения `://` среди URL-адресов, например:

```
a[href^='://']
```

Эта запись соответствует ссылкам `http://` и `https://`, а поскольку значение `://` не встречается больше нигде в свойстве `href`, данный способ должен работать для всех внешних ссылок.

Вы могли бы форматировать их любым образом, но распространенный способ — добавить рядом со ссылкой маленькое изображение — значок, который указывает на то, что это внешняя ссылка. Этот метод представлен в практикуме в конце текущей главы.

Если вы решили использовать абсолютные ссылки на другие страницы *вашего* сайта, то, вероятно, не захотите применять тот же стиль для них. На этот случай *отрицающий* псевдокласс `:not()` легко выполняет эту функцию. С помощью приведенного ниже кода вы легко можете выделить абсолютные ссылки, которые *не* ведут на страницы вашего сайта:

```
a[href*='://']:not(a[href*='www.mysite.com']) {
  background: url(images/globe.png) no-repeat center right;
  padding-right: 15px;
}
```


Ссылки на адреса электронной почты

Ссылки на адреса электронной почты — еще один особый вид ссылок. Обычно они выглядят как любые другие ссылки: имеют синий цвет и подчеркивание. Тем не менее они действуют не так, как другие ссылки. Переход по ссылке на адрес электронной почты запускает ассоциированный почтовый клиент на компьютере посетителей, что некоторых из них действительно отвлекает. Поэтому желательно как-то подсказать пользователю, что эта ссылка ведет на адрес электронной почты.

Для этого используется тот же базовый метод, который был описан применительно к внешним ссылкам выше. Поскольку все ссылки на почтовые адреса начинаются со значения `mailto:`, вы можете создать селектор наподобие следующего для стиля, форматирующего только определенный тип ссылок:

```
a[href^='mailto:']
```

Скоро вы увидите такой пример в действии.

Ссылки на определенные типы файлов

Некоторые ссылки указывают на файлы, а не на другие веб-страницы. Вы могли видеть ежегодные отчеты различных компаний в виде загружаемого PDF-файла или ZIP-архива файлов (как примеры для этой книги) на сайте. Ссылки на такие типы файлов, как правило, вынуждают браузер приступить к загрузке соответствующего файла на компьютер посетителя или, в случае PDF-файлов, запустить плагин (подключаемый модуль), который позволит просматривать документ прямо в браузере. Выбрав ссылку, посетитель может оказаться в шоке от того, что на самом деле началась загрузка файла размером 100 Мбайт!

Вы можете идентифицировать определенные типы файлов наподобие внешних ссылок или ссылок на адреса электронной почты. Но вместо того, чтобы пытаться найти какую-то конкретную информацию в начале URL-адреса ссылки, поищите ее в конце. Например, ссылка на PDF-документ может выглядеть так: ``, а ссылка на ZIP-архив так: ``. В каждом случае конкретный тип файла определяется расширением в конце URL-адреса: `.pdf` или `.zip`.

Каскадные таблицы стилей предоставляют селектор, который позволяет искать атрибуты, заканчивающиеся какой-либо конкретной информацией. Так, чтобы создать стиль ссылки на PDF-файлы, используйте следующий селектор:

```
a[href$='.pdf']
```

Сочетание `$=` означает «заканчивается на», и, соответственно, данный селектор позволяет выбрать все ссылки, значение атрибута `href` которых заканчивается на `.pdf`. Вы можете создать аналогичные стили и для других типов файлов:

```
a[href$='.zip'] /* zip-архив */  
a[href$='.doc'] /* документ Word */
```

Далее вы увидите примеры этого метода.

Практикум: форматирование ссылок

В этом практикуме мы потренируемся в форматировании ссылок разнообразными способами, в том числе путем добавления изображений-ролловеров и фоновых рисунков.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке с именем **09**.

Простейшее форматирование ссылок

Начнем с простого форматирования ссылок.

1. Запустите браузер и откройте файл веб-страницы `links.html` из папки `09\links`. Эта страница содержит множество ссылок (рис. 9.9), которые указывают на другие веб-страницы текущего или иных сайтов, а также ведут на адреса электронной почты. Сначала изменим цвет ссылок данной страницы.
2. Откройте файл `links.html` в редакторе HTML-кода и поместите указатель между открывающим и закрывающим тегами элемента `<style>`. Эта веб-страница уже имеет внешнюю таблицу стилей, придающую ей какое-то базовое форматирование и содержащую элемент `style` внутренней таблицы.

ПРИМЕЧАНИЕ

Вы разместите стили для этого упражнения во внутренней таблице, чтобы было легче писать код и просматривать страницу. Когда все будет готово, рекомендуется переместить стили во внешнюю таблицу стилей.

3. Добавьте во внутреннюю таблицу новый стиль:

```
<style>
a {
  color: #207EBF;
}
</style>
```

Он будет применяться ко всем элементам `a` на странице. С него хорошо начинать, поскольку он устанавливает общий внешний вид для ссылок на странице. Вы добавите больше стилей, которые позволят вам форматировать ссылки в определенных областях страницы. Теперь пришло время удалить это надоевшее подчеркивание под ссылкой.

4. Добавьте в только что созданный стиль свойство `text-decoration: none;`. Он убирает подчеркивание, но в то же время ссылка на веб-странице становится менее заметной. Не забывайте, что ссылки всегда должны визуально выделяться на общем фоне, чтобы у посетителей сайта не возникало сомнений в том, что это именно ссылки и на них можно щелкнуть кнопкой мыши.

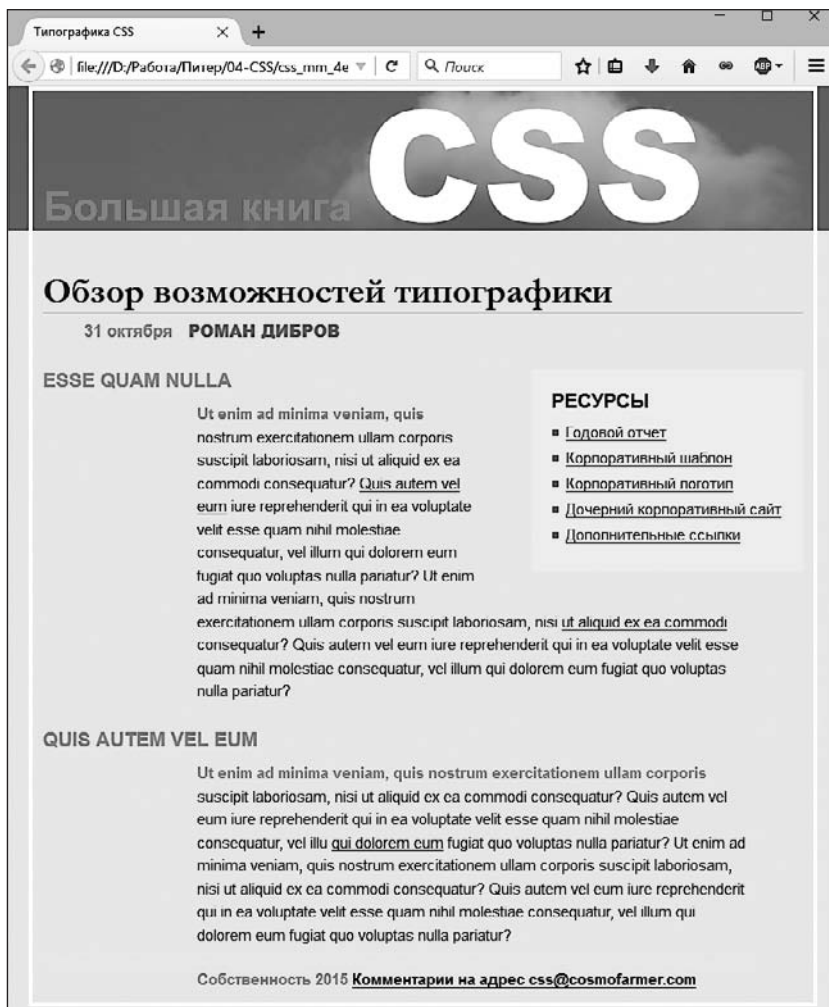


Рис. 9.9. Простейшая веб-страница со стандартным для браузеров форматированием — ссылки выделены синим цветом (или фиолетовым, если посетители ранее посещали страницы, на которые они ссылаются) и подчеркнуты

- Добавьте в стиль `a` свойство `font-weight: bold;`

Теперь ссылки отображаются полужирным шрифтом (для остального текста он также может быть установлен). Далее заменим подчеркивание на выделение другим способом, но сделаем это творчески, используя вместо свойства `text-decoration` границы.

- Добавьте свойство `border`, чтобы стиль принял следующий вид:

```
a {
  color: #207EBF;
  text-decoration: none;
```

```
font-weight: bold;
border-bottom: 2px solid #F60;
}
```

Теперь ссылки выделены, а использование границ вместо обычного подчеркивания позволило изменить стиль, цвет и толщину линий. А сейчас вы измените вид посещенных ссылок.

7. Добавьте псевдокласс `:visited` для посещенных ссылок следующим образом:

```
a:visited {
  color: #6E97BF;
}
```

Стиль изменяет внешний вид посещенных ссылок на более светлый с серым оттенком — это искусный способ отвлечь внимание от уже посещенной страницы. Если вы просматриваете страницу, щелкните на одной из ссылок (попробуйте, например, одну из тех, что находятся в центре страницы), а затем вернитесь к странице `links.html`. Вы должны увидеть, что ссылка стала светлее. Но вы также заметите, что остается выделение полужирным и все еще присутствует оранжевое подчеркивание, которое вы назначили стилем в шаге 6. Это пример каскадности в действии (см. главу 5): стиль `a:visited` более специфичен, чем простой селектор, поэтому его свойство цвета переопределяет тот цвет, который был назначен стилем.

Сейчас добавим эффект ролловера, чтобы фоновый цвет ссылки изменялся при наведении на нее указателя мыши.

8. Добавьте в таблицу стилей псевдокласс `:hover`:

```
a:hover {
  color: #FFF;
  background-color: #6E97BF;
  border-bottom-color: #6E97BF;
}
```

Этот псевдокласс применяется только на время, пока указатель мыши находится поверх ссылки. Свойство интерактивности эффекта ролловера позволяет посетителям узнать, что выполняется какое-то действие (рис. 9.10).

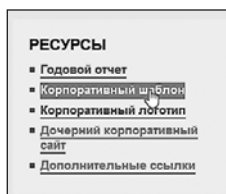


Рис. 9.10. С помощью нескольких стилей вы можете изменить внешний вид любой ссылки, а используя псевдокласс `:hover`, можно установить отдельный стиль, активируемый при нахождении указателя мыши над ссылкой

Добавление фонового изображения

Ссылка на адрес электронной почты в нижней части веб-страницы ничем не отличается от других ссылок на странице (рис. 9.11, *вверху*). Поскольку значение `mailto:`

указывает на адрес электронной почты, при щелчке на ней посетитель не перейдет на другую веб-страницу, а запустит почтовую программу. Чтобы обеспечить визуальное выделение этой ссылки, добавим небольшой значок почтового конверта.

1. Добавьте во внутреннюю таблицу стилей файла `links.html` селектор потомков:

```
a[href^="mailto:"] {
  color: #666666;
  border: none;
  background: url(images/email.gif) no-repeat left center;
}
```

Здесь используется расширенный селектор атрибута, выбирающий любые ссылки, начинающиеся со значения `mailto:` (то есть он выбирает ссылки на адреса электронной почты). Ссылке назначен серый цвет, а код `border: none` убирает подчеркивание, определенное в шаге 6. Свойство `background` добавляет фоновое изображение с левой стороны, а параметр `no-repeat` обеспечивает однократное отображение рисунка. Здесь трудность состоит в том, что фоновое изображение (значок конверта) располагается на заднем плане позади текста ссылки, и он становится трудночитаемым (см. рис. 9.11, *посередине*).

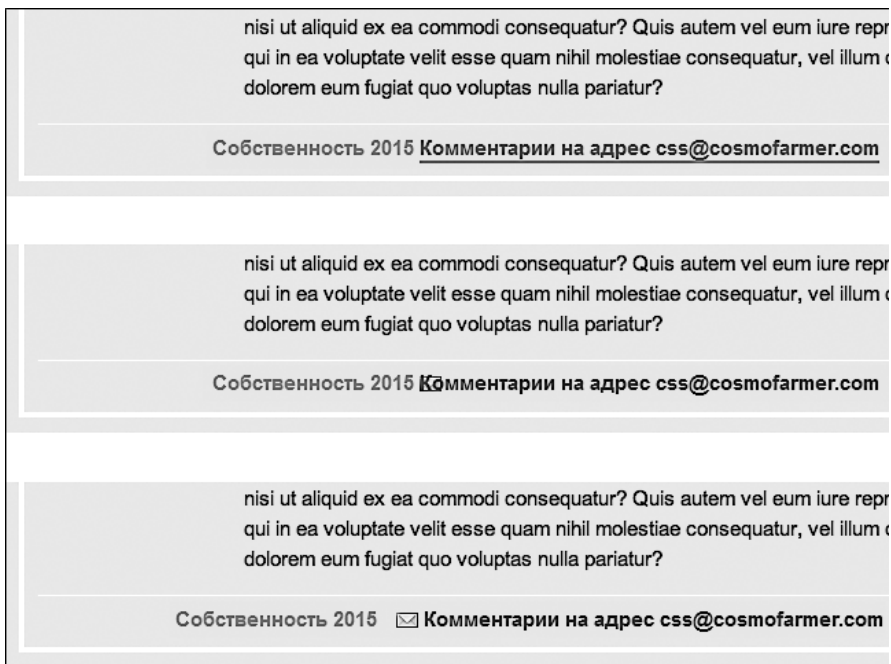


Рис. 9.11. Несколько тонких изменений сделают назначение ссылки понятным: простая ссылка (*вверху*) превращается в явную, узнаваемую владельцами электронной почты (*внизу*)

2. Добавьте в только что созданный стиль атрибута левый отступ размером 20 пикселей:

```
padding-left: 20px;
```

Помните, что отступ добавляет промежуток между содержимым и границей элемента, поэтому установка небольшого отступа слева смещает текст ссылки на 20 пикселей, оставляя фоновое изображение на месте. И последний штрих: нужно немного отодвинуть всю ссылку от упоминания об авторском праве.

3. Добавьте в стиль поле слева размером 10 пикселей. Должен получиться следующий стиль:

```
a[href^="mailto:"] {
  color: #666666;
  border: none;
  background: url(images/email.gif) no-repeat left center;
  padding-left: 20px;
  margin-left: 10px;
}
```

Эта небольшая поправка обеспечивает визуальное отделение изображения-значка почтовой ссылки от упоминания об авторском праве (см. рис. 9.11, *внизу*). Таким образом, ссылка воспринимается посетителем как сочетание значка с текстом.

Выделение внешних ссылок

Иногда требуется визуально показать, что ссылка ведет на внешний сайт. Этим можно сказать посетителям, что дополнительная информация по теме находится где-то во Всемирной паутине, на другом сайте, или предупредить, что выбор этой ссылки приведет к переходу на другой сайт. Кроме того, вы, возможно, захотите идентифицировать ссылки, указывающие на загружаемые файлы или другие документы, не являющиеся веб-страницами.

На веб-странице, над которой вы работаете, боковая панель, размещенная справа, содержит различные типы ссылок. Вы их выделите значками, используя отдельный значок для каждого типа. Для начала вы настроите базовый стиль, который применяется ко всем ссылкам.

1. Добавьте во внутреннюю таблицу стилей веб-страницы `links.html` следующий код:

```
.resources a {
  border-bottom: none;
}
```

Поскольку все ссылки, которые нужно отформатировать, находятся внутри элемента `div` с классом `resources`, селектор потомков `.resources a` воздействует только на эти ссылки. Стиль избавляет от подчеркивания, которое было добавлено в общем стиле ранее.

Далее вы добавите значок, размещаемый рядом с внешними ссылками.

2. Добавьте еще один стиль в конец внутренней таблицы стилей веб-страницы `links.html`:

```
.resources a[href*='://'] {
  background: url(images/globe.png) no-repeat right top;
}
```

В этом стиле селектора потомков используется расширенный селектор атрибута. В основном он воздействует на любую ссылку, которая содержит символы `://` (но только на ту, которая находится внутри элемента с классом `resources`). Как и стиль ссылки на адрес электронной почты, который вы создали ранее, этот стиль добавляет фоновое изображение. Он помещает рисунок в правую часть ссылки.

Тем не менее у этого стиля есть проблема, аналогичная проблеме стиля ссылки на адрес электронной почты, — изображение появляется позади текста ссылки. Решение такое же — нужно добавить отступы, чтобы переместить изображение в нужное место. В этом случае, однако, вместо левого отступа вы добавите правый (поскольку значок появляется в правой части ссылки). Кроме того, так как все ссылки в разделе ресурсов будут иметь значки схожих размеров, вы можете уменьшить объем кода, добавив отступы с помощью стиля `.resources a`, созданного на шаге 1.

3. Отредактируйте стиль `.resources a` так, чтобы он выглядел следующим образом:

```
.resources a {  
  border-bottom: none;  
  padding-right: 22px;  
}
```

Если вы сохраните страницу и просмотрите ее в браузере, то увидите маленькие значки с земным шаром в правой части двух нижних ссылок боковой панели. Теперь отформатируем другие ссылки.

4. Добавьте еще три стиля во внутреннюю таблицу стилей:

```
.resources a[href$='.pdf'] {  
  background: url(images/acrobat.png) no-repeat right top;  
}  
.resources a[href$='.zip'] {  
  background: url(images/zip.png) no-repeat right top;  
}  
.resources a[href$='.doc'] {  
  background: url(images/word.png) no-repeat right top;  
}
```

Эти три стиля проверяют, каким значением заканчивается атрибут `href`, идентифицируют ссылки как файлы Adobe Reader (`.pdf`), ZIP-архивы (`.zip`) или документы Word (`.doc`) и назначают различные значки в каждом конкретном случае.

5. Наконец, добавьте состояние наведения указателя мыши для ссылок на ресурсы:

```
.resources a:hover {  
  color: #000;  
  background-color: rgba(255,255,255,.8);  
}
```

Этот стиль изменяет цвет текста и добавляет цвет фона (рис. 9.12).

Окончательную версию этого урока вы найдете в файле `09_finished/links/links.html`.

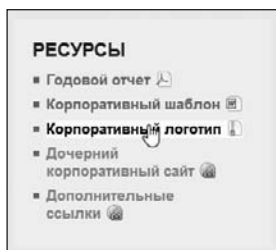


Рис. 9.12. Используя расширенные селекторы атрибутов, вы можете легко идентифицировать и форматировать различные типы ссылок — внешние, ссылки на PDF-файлы, текстовые документы Word и архивы

Практикум: создание панели навигации

На этом практическом занятии мы превратим простой список ссылок во впечатляющую панель навигации с эффектом ролловера и выделением нажатой кнопки текущего раздела сайта.

1. Откройте файл веб-страницы `nav_bar.html` из папки `09\nav_bar` в редакторе HTML-кода.

Как видите, в этом файле содержится совсем короткий исходный код. Здесь присутствует внутренняя таблица со сбросом стандартных стилей CSS и одним правилом, определяющим простейшее форматирование тела веб-страницы. HTML-код создает маркированный список, состоящий из шести ссылок. Веб-страница имеет вид, показанный на рис. 9.13, 1. Первым шагом будет добавление HTML-кода, обеспечивающего целенаправленное форматирование ссылок списка средствами каскадных таблиц стилей.

2. Найдите открывающий тег `` и добавьте в него код `class="mainNav"` следующим образом:

```
<ul class="mainNav">
```

Атрибут `class` назначает этот список основной областью навигации. Мы будем использовать его для создания селекторов потомков, избирательно форматизирующих только ссылки этого списка (а не все ссылки веб-страницы).

3. Добавьте во внутреннюю таблицу после стиля `body` следующий код:

```
.mainNav {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
}
```

Этот стиль будет применен лишь к элементу с классом `mainNav`, в данном случае элементу `ul`. Он удаляет отступ и маркеры, которые браузер добавляет в маркированные списки, как показано на рис. 9.13, 2. Теперь приступим к форматированию ссылок.



Рис. 9.13. Вам может показаться, что преобразование простого списка в сложную панель навигации требует выполнения объемной работы, но на самом деле нужно лишь создать несколько стилей

4. Добавьте селектор потомков для форматирования ссылок списка:

```
.mainNav a {
  color: #000;
  font-size: 11px;
  text-transform: uppercase;
  text-decoration: none;
}
```

Этот стиль определяет базовое форматирование текста ссылок: устанавливает цвет и размер шрифта, определяет все буквы как прописные и удаляет линию подчеркивания (см. рис. 9.13, 3). Сейчас придадим ссылкам вид кнопок.

5. Добавьте в стиль `.mainNav a` свойства `border` и `padding`:

```
border: 1px dashed #999;
padding: 7px 5px;
```

Теперь при просмотре веб-страницы в браузере вы заметите пару проблем (см. рис. 9.13, 4): границы накладываются друг на друга, а ссылки имеют различную ширину. Это происходит потому, что элемент `a` является строчным и ширина ссылки равна длине ее текста. Кроме того, верхний и нижний отступы не обеспечивают увеличения высоты строчным элементам, поэтому границы накладываются друг на друга (о строчных элементах читайте в разделе «Управление размерами полей и отступов» главы 7). Решить эти проблемы со ссылками можно путем изменения их способа отображения браузером.

6. Добавьте свойство `display: block`; в стиль `.mainNav a`.

Мы изменяем параметры отображения элемента `a` таким образом, чтобы он обрабатывался браузером как блочный элемент (например, абзац текста), с расположением ссылок в виде списка, в точности одна над другой. Осталась

единственная проблема — ссылки растянуты по всей ширине окна браузера — слишком много для кнопок. Исправим ситуацию, ограничив ширину элемента `ul` соответствующим стилем.

7. Во внутренней таблице найдите стиль `.mainNav` и добавьте в него свойство `width: 175px;`.

```
.mainNav {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
  width: 175px;  
}
```

При ширине списка 175 пикселей ссылки по-прежнему слишком широки, хотя и ограничиваются шириной элемента-контейнера (`ul`). В большинстве случаев список ссылок заключен в какой-либо элемент разметки (например, боковую панель), для которого уже определена ширина, поэтому можете пропустить этот шаг (о том, как создавать боковые панели, читайте в части III).

Сейчас перейдем к самому интересному.

8. Добавьте в стиль `.mainNav a` свойства фона следующим образом:

```
.mainNav a {  
  color: #000;  
  font-size: 11px;  
  text-transform: uppercase;  
  text-decoration: none;  
  border: 1px dashed #999;  
  padding: 7px 5px;  
  display: block;  
  background-color: #E7E7E7;  
  background-image: url(images/nav.png);  
  background-repeat: no-repeat;  
  background-position: 0 2px;  
}
```

Эти строки кода изменяют цвет фона ссылок на серый и устанавливают одиночное изображение с левой стороны каждой кнопки (см. рис. 9.13, 5). Здесь тоже нужно кое-что исправить: текст ссылки накладывается на значок, а линии границ между кнопками имеют толщину, равную 2 пикселям (теоретически границы имеют толщину 1 пиксел, но сливающиеся воедино линии соседних ссылок образуют линию толщиной 2 пикселя).

ПРИМЕЧАНИЕ

Используя сокращенную запись свойства `background`, можно изменить код на шаге 8 на следующий: `background: #E7E7E7 url(images/nav.png) no-repeat 0 2px;`

9. Удалите верхнюю границу и измените отступ в стиле `.mainNav a`, чтобы его код выглядел следующим образом:

```
.mainNav a {  
  color: #000;
```

```

font-size: 11px;
text-transform: uppercase;
text-decoration: none;
border: 1px dashed #999;
border-bottom: none;
padding: 7px 5px 7px 30px;
display: block;
background-color: #E7E7E7;
background-image: url(images/nav.png);
background-repeat: no-repeat;
background-position: 0 2px;
}

```

Выглядит неплохо: текст каждой ссылки отделен от значка, границы выделяются. Однако нижняя граница последней ссылки исчезла. Существует несколько способов разобраться с этим. Один состоит в том, чтобы применить нижнюю границу к элементу `ul`, содержащему список ссылок (поскольку этот элемент не имеет отступов, нет промежутка, отделяющего верх элемента `ul` от верхней стороны этой первой ссылки). Второй способ заключается в использовании псевдокласса `:last-of-type`. Вам нужно выделить ссылку, которая находится внутри последнего элемента списка на панели навигации, и добавить к ней нижнюю границу.

10. Между стилями `.mainNav` и `.mainNav a` добавьте следующий стиль:

```

.mainNav li:last-of-type a {
border-bottom: 1px dashed #999;
}

```

Эти стили селектора потомка форматируют ссылку (`a`), расположенную внутри последнего элемента списка (`li:last-of-type`) или списка навигации (`.mainNav`).

Вот и все, мы создали простейшую панель навигации с применением границ, отступов, фонового цвета и изображений (см. рис. 9.13, б).

Добавление ролловеров и выделение текущего раздела сайта

Пришло время усовершенствовать панель навигации и придать ей некоторую интерактивность. Во-первых, обеспечим кнопкам панели навигации эффект ролловера. Они должны изменять свой вид, акцентируя внимание посетителя на том, какую кнопку он собирается нажать.

Неплохо было бы оповещать посетителя, в каком разделе (на какой странице) сайта он находится. Мы можем обеспечить созданной панели навигации автоматическую интерактивность. Она будет изменять свой стиль в соответствии с выбранным разделом страницы. Звучит совсем просто, но в действительности это потребует некоторой разметки и настроек, как вы увидите в последующих шагах.

Эффект ролловера реализовать легко, но начнем по порядку.

1. Добавьте в конце таблицы стилей файла `nav_bar.html` следующий стиль:

```

.mainNav a:hover {
font-weight: bold;
}

```

```
background-color: #B2F511;
background-position: 3px 50%;
}
```

Он определяет внешний вид ссылки-кнопки в состоянии наведения указателя мыши. Стиль изменяет шрифт текста кнопки с обычного на полужирный, а также цвет фона на ярко-зеленый. Кроме того, он использует метод CSS-спрайтов, упоминавшийся ранее. То же самое изображение применяется на шаге 8 в прошлом разделе — оно в действительности содержит три различных значка (рис. 9.14). В этом случае изображение центрируется внутри кнопки, отображая средний значок файла.

Теперь при появлении указателя мыши поверх любой кнопки она моментально изменяет свой вид (откройте веб-страницу в своем браузере).

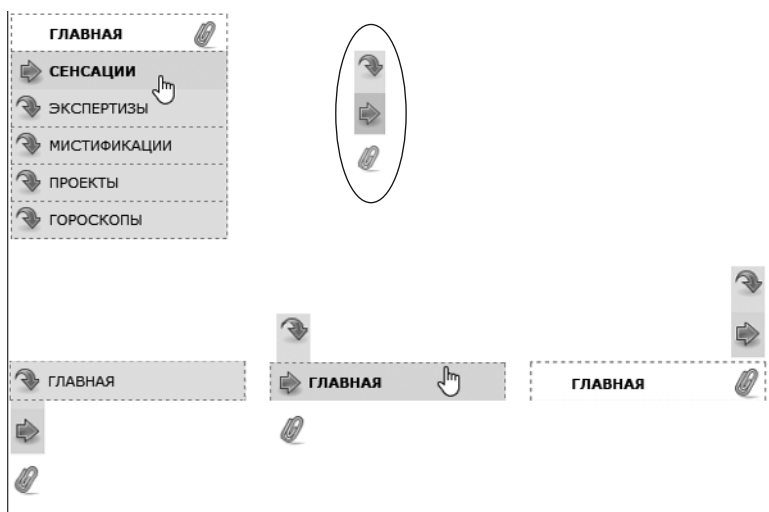


Рис. 9.14. С помощью несложного CSS-кода можно создать интерактивный эффект ролловера для кнопок панели навигации, а также автоматически подсветить раздел сайта текущей страницы

Теперь сделаем панель навигации более информативной, выделив кнопку, соответствующую текущему разделу, страницу которого открыл посетитель сайта. Чтобы сделать это, нам потребуется идентифицировать в HTML-коде панели навигации две вещи: раздел, к которому принадлежит страница и разделы, на которые указывает каждая ссылка. В нашем примере предположим, что открыта домашняя страница.

ПРИМЕЧАНИЕ

Другим вариантом будет создание класса, который изменяет внешний вид ссылки, представляющей раздел страницы. Для веб-страницы гороскопа ссылка панели навигации может выглядеть следующим образом: `Гороскопы`.

- Найдите открывающий тег `<body>` и добавьте в него класс `class="home"`:
`<body class="home">`

Теперь, когда мы знаем, какому разделу сайта принадлежит текущая открытая страница, можно использовать селектор потомков для создания конкретного стиля, который будет применен к веб-странице раздела «Сенсации». Далее мы должны маркировать разделы, на которые указывает каждая ссылка-кнопка, что достигается путем добавления классов.

3. Найдите в HTML-коде панели навигации ссылку «Главная» и добавьте в нее класс `class="homeLink"`, чтобы код элемента выглядел следующим образом:

```
<a href="/index.html" class="homeLink">Главная</a>
```

Этот класс определяет ссылку, предоставляя возможность создания стиля, который будет применен только к ней.

Естественно, нам потребуется добавить классы для всех ссылок панели навигации.

4. Повторите шаг 3 для каждой из ссылок, используя следующие классы: `featureLink`, `expertLink`, `quizLink`, `projectLink` и `horoscopeLink`.

С HTML-кодом закончили. Настало время создать CSS-код. Теперь у нас есть маркированные страница и ссылка, и мы можем создать селектор потомков, выделяющий кнопку-ссылку «Главная».

5. Добавьте в таблицу стилей веб-страницы следующий код:

```
.home .homeLink {  
  background-color: #FFFFFF;  
  background-position: 97% 100%;  
  padding-right: 15px;  
  padding-left: 30px;  
  font-weight: bold;  
}
```

Мы уже рассматривали эти свойства ранее. Снова вы используете метод CSS-спрайтов для управления позицией фонового изображения. На этот раз изображение смещается на 97 % вправо, а его нижняя часть помещается в нижнюю часть кнопки. Другими словами, значок будет отображаться внизу изображения (см. рис. 9.14). О том, как использовать процентные значения при работе с фоновыми изображениями, мы говорили в главе 8.

В данном случае наибольший интерес представляет селектор `.home .homeLink`. Это очень специфичный селектор, применяемый только к ссылке с классом `homeLink`, которая *также* заключена внутрь элемента `body` с классом `home`. Если вы измените класс страницы, например, на `quiz`, с кнопки-ссылки раздела «Главная» исчезнет выделение.

Посмотрите веб-страницу в браузере, чтобы увидеть результат: теперь ссылка «Главная» имеет белый фон и значок скрепки. Чтобы проделать все то же самое с остальными ссылками, вы должны немного *расширить* селектор.

6. Отредактируйте селектор только что созданного стиля:

```
.home .homeLink,  
.feature .featureLink,  
.expert .expertLink,
```

```
.quiz .quizLink,  
.project .projectLink,  
.horoscope .horoscopeLink{  
  background-color: #FFFFFF;  
  background-position: 97% 100%;  
  padding-right: 15px;  
  padding-left: 30px;  
  font-weight: bold;  
}
```

Конечно, получился довольно объемный CSS-код. Этот стиль теперь адресует-ся ко всем ссылкам панели навигации, но только при выполнении определенных условий. Если вы в дальнейшем измените класс элемента `body`, например, на `quiz`, то ссылка раздела «Мистификации» приобретет такое же форматирование, каким была выделена ссылка раздела «Сенсации».

Теперь пришло время проверить результаты работы.

ПРИМЕЧАНИЕ

Этот длинный селектор является примером группового селектора, который мы обсуждали в главе 3.

7. Измените атрибут `class` элемента `body`, присвоив ему значение `feature`, следующим образом:

```
<body class="feature">
```

Просмотрев веб-страницу, вы увидите, что ссылка «Сенсации» теперь выделена белым цветом фона и значком скрепки. Весь секрет в том, что нужно изменить атрибут `class` элемента `body`, чтобы указать, к какому разделу сайта принадлежит страница. Например, для страницы гороскопа измените атрибут класса `body` на `class="horoscope"`, а для главной страницы используйте атрибут `class="home"` (см. рис. 9.14).

ПРИМЕЧАНИЕ

Готовы продолжить форматирование? Попытайтесь добавить эффект ролловера, чтобы закончить стиль, созданный на шаге 6 (используйте псевдокласс `:hover` в качестве компонента селектора: `.quiz .quizLink:hover`). Попробуйте также добавить другое изображение для ссылки главной страницы (можете использовать файл `home.png` из папки `images`).

Файл с финальным вариантом панели навигации `nav_bar_vertical.html` можно найти в папке `09_finished\nav_bar`.

От вертикальной к горизонтальной панели навигации

Предположим, вы хотите создать горизонтальную панель навигации в верхней части веб-страницы. Никаких проблем — большую часть работы мы уже выполнили. Чтобы расположить кнопки горизонтально в одну строку, необходимо немного изменить уже созданную веб-страницу (мы будем дорабатывать файл `nav_bar.html`, поэтому, если вы хотите сохранить вертикальную панель навигации, прежде чем продолжить, сделайте копию файла).

1. Убедитесь в том, что вы выполнили все шаги по созданию вертикальной панели навигации и откройте файл `nav_bar.html` в редакторе HTML-кода.

Сейчас вы увидите, как легко можно изменить ориентацию панели. Сначала подчистим кое-что из того, что мы уже сделали. Вам нужно удалить свойство `width`, которое вы установили для элемента `ul` ранее. Оно препятствует тому, чтобы навигационные кнопки охватывали всю длину страницы.

2. Найдите стиль `.mainNav` и удалите из него свойство `width: 175px;`.

Теперь приведем секрет преобразования вертикальной панели навигации в горизонтальную.

3. Добавьте в таблицу новый стиль (сразу после `.mainNav`):

```
.mainNav li {
  float: left;
  width: 14em;
}
```

Стиль применяется к элементу `li` (представляющему собой элементы списка, каждый из которых содержит свою ссылку). Первая команда устанавливает для элемента выравнивание по левому краю. Таким образом, каждый последующий элемент `li` располагается с правой стороны предыдущего (такой же эффект вы могли наблюдать в практикуме по созданию фотогалереи в главе 8). Кроме того, устанавливая ширину элемента `li`, мы одновременно определяем ширину для всех кнопок панели навигации. В данном случае значение `14em` обеспечивает достаточный размер, чтобы вместить самый длинный текст названия ссылки. Если будет необходимо увеличить длину, вы должны будете увеличить это значение.

При просмотре веб-страницы в браузере вы увидите, что с панелью навигации в основном все в порядке. Требуется только небольшие корректировки (рис. 9.15, 1). Нижняя граница под кнопками отсутствует. Кроме того, поскольку левая граница кнопки расположена рядом с границей ссылки, их левые и правые границы сливаются между собой, образуя линии толщиной 2 пикселя. Сейчас мы решим эти проблемы.



Рис. 9.15. Преобразование вертикальной панели в более компактную с горизонтальным расположением кнопок требует выполнения всего нескольких действий; гораздо больше усилий нужно приложить для настройки стилей, определяющих параметры границ и позиционирование фонового изображения

4. В стиле `.mainNav` а измените свойство `border-bottom: none;` на `border-right: none;`:
Это действие приводит к удалению левых границ кнопок, чтобы они не удваивались, и в то же время к нижнему краю добавляется граница. Однако теперь граница с правой стороны последней кнопки на панели навигации исчезла (см. рис. 9.15, 2). Чтобы исправить эту проблему, вы можете сделать то же, что делали в предыдущей части практикума, — использовать селектор `:last-of-type`.
5. Измените свойство `border-bottom` стиля `.mainNav li:last-of-type` а на `border-right`:

```
.mainNav li:last-of-type a {  
    border-right: 1px dashed #999;  
}
```

Это изменение добавит правую границу, но только к последней ссылке на панели навигации (см. рис. 9.15, 3). Наконец, скрепки, выровненные по правому краю выбранной кнопки, выглядят странно. Сейчас вы измените их положение, разместив у левого края кнопки.

6. Найдите стиль выбранной кнопки, созданный в шаге 6 (стиль с длинным селектором). Измените координаты его свойства `background` с `97% 100%` на `3px 100%`. Теперь стиль должен выглядеть следующим образом:

```
.home .homeLink,  
.feature .featureLink,  
.expert .expertLink,  
.quiz .quizLink,  
.project .projectLink,  
.horoscope .horoscopeLink {  
    background-color: #FFFFFF;  
    background-position: 3px 100%;  
    padding-right: 15px;  
    padding-left: 30px;  
    font-weight: bold;  
}
```

Просмотрите веб-страницу в браузере, и вы обнаружите, что горизонтальная панель навигации полностью работоспособна и замечательно выглядит (см. рис. 9.15, 4).

Окончательную версию этого урока вы найдете в файле `09_finished/links/links.html`.

ПРИМЕЧАНИЕ

Возможно, вы захотите центрировать текст кнопок-ссылок панели навигации. Для этого нужно сделать две вещи: добавить в стиль `.mainNav` а свойство `text-align: center;` и изменить свойство `left-padding` этого же стиля, чтобы текст был расположен точно по центру кнопок.

10 Преобразования, переходы и анимация с помощью CSS

За короткую историю Всемирной паутины у разработчиков было несколько вариантов добавления анимации к своим сайтам. Самая простая и примитивная анимация в рамках одного изображения предоставлялась в GIF-формате. Технология Adobe Flash стала инструментом номер один и позволяла создавать сложную анимацию и даже игры и веб-приложения, но обладала рядом недостатков. Изучить ее весьма непросто, и она не позволяет взаимодействовать с другим HTML-кодом на странице, представляющим изображения, заголовки и абзацы, из которых составлен веб-контент, а также недоступна для использования на мобильных устройствах. Эти минусы определили эру заката технологии Flash. JavaScript-сценарии позволяют анимировать все, что имеется на веб-странице, но ценой изучения всех тонкостей этого языка программирования. К счастью, CSS предоставляет способ перемещения, преобразования и анимации любого HTML-элемента, имеющегося на странице без обращения к любой из упомянутых здесь технологий.

Преобразования

Каскадные таблицы стилей включают несколько свойств, связанных с преобразованиями элемента веб-страницы, будь то вращение, масштабирование, перемещение этого элемента или его искажение вдоль горизонтальной или вертикальной оси (которое называется *скашиванием*). Преобразование можно использовать, например, для небольшого наклона (вращения) панели навигации или для увеличения изображения при нахождении поверх него указателя мыши. Можно даже сочетать несколько преобразований для получения весьма впечатляющих визуальных эффектов.

Основным свойством CSS для получения любого из этих изменений является `transform`. Оно используется с предоставлением *типа* желаемого преобразования и добавлением значения, указывающего на *степень* преобразования элемента. Например, для вращения элемента предоставляется ключевое слово `rotate`, за которым следует количество градусов поворота:

```
transform: rotate(10deg);
```

Показанное выше объявление приведет к вращению элемента на 10° по часовой стрелке.

Все популярные браузеры поддерживают преобразования: Internet Explorer 9, Safari, Chrome, Firefox и Opera. Тем не менее, чтобы преобразования работали в браузерах Safari и Internet Explorer 9, вам необходимо использовать вендорные префиксы, как это показано в первых двух строках следующего кода (см. также врезку далее):

```
-webkit-transform: rotate(10deg);  
-ms-transform: rotate(10deg);  
transform: rotate(10deg);
```

У преобразований в CSS есть одна странность: они не касаются окружающих элементов. То есть, если повернуть элемент на 45° , он может наложиться на те элементы, которые находятся выше, ниже его или по сторонам. Сначала браузеры выделяют элементу то пространство, которое он занимал бы при обычных обстоятельствах (до преобразования), а затем они выполняют преобразование элемента (его вращение, увеличение или сжатие). Этот процесс становится очевидным при увеличении размеров элемента путем использования функции `transform: scale` (см. далее подраздел «Масштабирование»). При увеличении масштаба элемента в два раза браузер увеличивает преобразуемый элемент, но при этом не смещает окружающий его контент, что обычно приводит к частичному перекрытию содержимого страницы (рис. 10.1). Иначе говоря, браузер сохраняет все остальные части страницы так, словно элемент не был масштабирован.

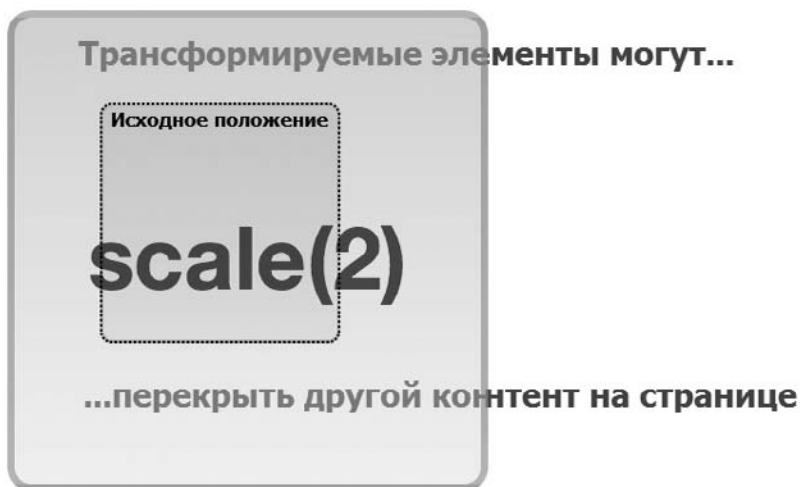


Рис. 10.1. Трансформируемые элементы игнорируются другими, окружающими их, элементами. Показанное здесь увеличение элемента `div` (большой квадрат) приводит к перекрытию им текста выше и ниже этого элемента. Квадрат посередине, выделенный пунктиром, представляет собой элемент `div` до его масштабирования

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Вендорные префиксы

Каскадные таблицы стилей — это всегда развивающийся набор правил. Даже сейчас, когда вы создаете страницу, умные люди в Консорциуме W3C работают над новыми свойствами CSS, а другие умные люди добавляют поддержку новых правил в браузеры. На самом деле иногда производители браузеров сами придумывают новые правила CSS, которые, по их мнению, было бы приятно использовать. Периодически члены Консорциума W3C придумывают новые свойства CSS, которые производители браузеров постепенно принимают.

Во время развития новых свойств и стандартов языка CSS производители браузеров настороже. Они не хотят окончательно принимать черновые свойства CSS, которые еще могут измениться. Точно так же в процессе экспериментов со свойствами CSS собственных разработок они не спешат утверждать, что пришли к согласованному стандарту. Чтобы пометить свойство CSS, как экспериментальное или пока еще не полностью утвержденное, производители (вендоры) браузеров используют префиксы перед его именем. Каждый из основных производителей браузеров имеет собственный *вендорный префикс*:

- `-webkit-` — используется в Chrome, Safari, Opera и других браузерах, работа которых основана на технологии WebKit;
- `-moz-` — применяется браузером Mozilla Firefox;
- `-ms-` — используется в Internet Explorer.

Пока свойство CSS находится в разработке, браузер может поддерживать версию с вендорным префиксом. Например, когда было впервые предложено свойство `border-radius`, браузер Firefox поддерживал вариант `-moz-border-radius`, в то время как Safari поддерживал `-webkit-border-radius`.

Если какое-либо свойство требует указать вендорный префикс, вам, как правило, необходимо написать несколько строк кода, чтобы создать такой же эффект: по одной для каждого производителя и, наконец, одну с версией свойства без префикса:

- `::-webkit-input-placeholder { color: red; }`
- `::-moz-placeholder { color: red; }`
- `:-ms-input-placeholder { color: red; }`
- `:placeholder-shown { color: red; }`

Обычно, если рабочая группа Консорциума W3C принимает свойство и утверждает его как окончательное, разработчики браузеров выпускают обновление своего продукта, поддерживающее изменения. Например, все основные браузеры теперь поддерживают свойство `border-radius` без необходимости использования вендорного префикса. На момент чтения этой книги некоторые свойства CSS по-прежнему будут в разработке, поэтому для них вам может понадобиться использовать вендорные префиксы. Эта книга подскажет вам, когда префиксы необходимы и как применять их с определенными свойствами.

Вращение

Разобраться в работе функции `rotate` свойства `transform` довольно просто: ей предоставляется значение угла в диапазоне от 0 до 360, а браузер вращает указанный элемент по кругу на заданное количество градусов (рис. 10.2). Чтобы указать значение угла, используется число, за которым следует сокращение `deg`. Например, для вращения элемента на 180° добавьте следующее объявление:

```
transform: rotate(180deg);
```

ПРИМЕЧАНИЕ

В приводимые здесь примеры не включаются вендорные префиксы, но при помещении кода в таблицу стилей нужно добавлять к нему свойства `-webkit-transform`, `-moz-transform`, `-o-transform` и `-ms-transform`.

Для вращения элемента против часовой стрелки можно применять отрицательные числовые значения. Например, средний элемент в верхней части рис. 10.2 повернут на 45° против часовой стрелки благодаря следующему объявлению:

```
transform: rotate(-45deg);
```

Значение 0deg не выполняет никакого вращения, значение 360deg выполняет одно вращение на полный оборот, а значение 720deg — два полных вращения. Разумеется, внешний вид элемента, которому было назначено одинарное, двойное или тройное вращение, останется таким же, как и у элемента, не подвергавшегося никакому вращению (верхнее левое и нижнее правое изображения на рис. 10.2), поэтому использование значения с числом 360 и кратным ему может вызвать удивление. В CSS предоставляется механизм анимации изменений свойств CSS. Так, например, можно заставить кнопку прокрутиться четыре раза, когда указатель мыши пользователя находится поверх нее, путем установки начального значения вращения равным 0deg и добавления стиля :hover для этой кнопки с поворотом на 1440deg. Как это делается, будет вскоре показано.

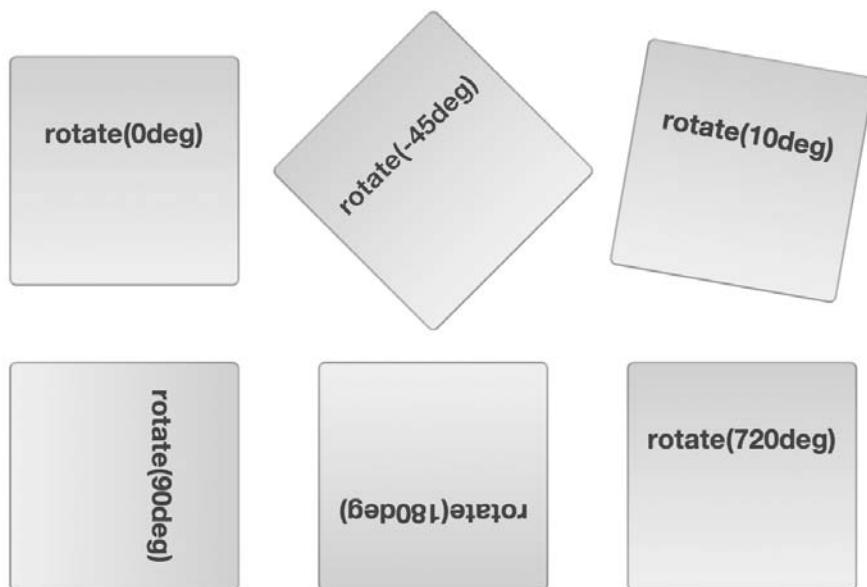


Рис. 10.2. Функция rotate позволяет любой элемент страницы — div, button, banner, photo или footer — повернуть как на небольшой, так и на совершенно немислимый градус

Масштабирование

Элемент можно увеличить или уменьшить в размерах, воспользовавшись для этого функцией scale (рис. 10.3). Например, чтобы увеличить элемент вдвое, нужно добавить следующее объявление:

```
transform: scale(2);
```

Число в скобках является коэффициентом масштабирования — значением, на которое умножаются текущие размеры элемента. Например, 1 указывает на отсутствие масштабирования, .5 указывает на половину текущего размера, а 4 увеличивает элемент в четыре раза. То есть числа в диапазоне между 0 и 1 приводят к уменьшению, а числа больше 1 — к увеличению элемента (значение 0 фактически делает элемент невидимым на странице).

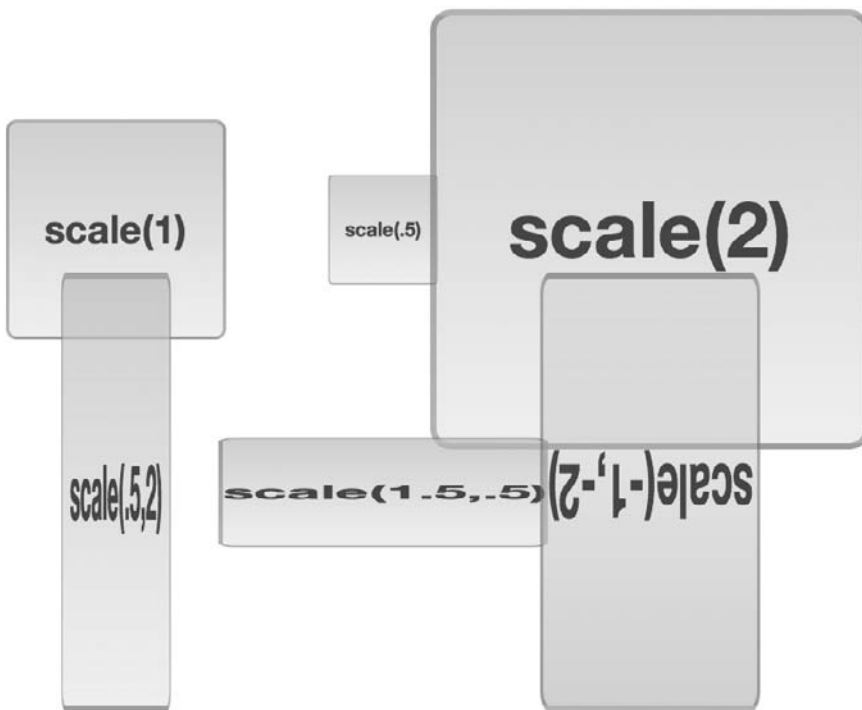


Рис. 10.3. Для увеличения размера любого элемента на странице используется функция `scale`. Но помните, что при увеличении элемента браузер не убирает другие окружающие элементы

На это число происходит масштабирование элемента и всего, что в нем находится. Например, если масштабировать контейнер `div` с коэффициентом 2, то вдвое шире и выше станет не только сам контейнер, но и текст внутри него, это же касается и находящихся внутри изображений.

Конечно, можно удивиться, а зачем вообще все это нужно. В конце концов, можно ведь увеличить или уменьшить ширину и высоту элемента, используя свойства `width` и `height`, а изменить размер шрифта текста — с помощью свойства `font-size`. Чаще всего масштабирование используется для визуальных изменений элемента на странице в динамическом режиме. Например, помещение указателя мыши поверх кнопки может тут же привести к увеличению ее размера. Этого эффекта можно добиться в помощью состояния `:hover`.

Предположим, что на странице есть ссылка, к которой применен класс `.button`. Для форматирования этой ссылки в виде кнопки можно создать следующий стиль:

```
.button {  
  font: .9em Arial, Helvetica, sans-serif;  
  border-radius: .5em;  
  background-color: rgb(34, 255, 23);  
  border: 1px solid rgba(0,0,0,.5);  
  padding: .5em;  
}
```

Чтобы выделить эту кнопку, можно сделать ее немного больше при помещении на нее указателя мыши:

```
.button:hover {  
  -webkit-transform: scale(1.2);  
  -ms-transform: scale(1.2);  
  transform: scale(1.2);  
}
```

Когда указатель мыши покидает пределы кнопки, преобразование удаляется и кнопка возвращается к своему обычному размеру. Способ анимации этого изменения размеров с помощью преобразований с применением каскадных таблиц стилей будет рассмотрен в разделе «Переходы».

СОВЕТ

Подобный метод можно применить для изображений. Показать галерею из небольших изображений, а затем добавить к ним стиль `:hover`, чтобы при помещении на них указателя мыши они становились больше. Чтобы все это выглядело вполне достойно, в HTML-коде указывается финальная, увеличенная версия изображения, но его размер уменьшается либо с помощью каскадных таблиц стилей, либо с помощью свойств `width` и `height` элемента `img`.

Можно даже проводить горизонтальное и вертикальное масштабирование по отдельности. Для этого внутри скобок нужно указать два значения, разделенные запятой: первое число будет относиться к горизонтальному, а второе — к вертикальному масштабированию. Например, чтобы сделать элемент вдвое меньше по ширине, но вдвое выше, используется следующее объявление:

```
transform: scale(.5,2);
```

Результат можно увидеть на левом нижнем изображении на рис. 10.3.

В языке CSS также предоставлены отдельные функции для горизонтального и вертикального масштабирования: `scaleX` проводит масштабирование по горизонтальной оси, а `scaleY` — по вертикальной. Например, чтобы элемент стал вдвое выше без изменения его ширины, можно воспользоваться следующим объявлением:

```
transform: scaleY(2);
```

ПРИМЕЧАНИЕ

Для экономии бумаги в этом примере опять показана только лишь версия свойства `transform`, в которой вендорные префиксы не используются. Чтобы объявление работало в браузерах Safari и Internet Explorer 9, нужно добавить соответствующие префиксы `-webkit-` и `-moz-`.

Однако чтобы элемент стал в три с половиной раза шире, но не выше или ниже, следует воспользоваться таким объявлением:

```
transform: scaleX(3.5);
```

Есть еще один визуальный трюк, предлагаемый функцией масштабирования: возможность перевернуть элемент вокруг его вертикальной или горизонтальной центральной оси. Никто не знает достоверно, какой из разделов математики был использован Консорциумом W3C, чтобы придумать эту систему, но если для масштабирования применить отрицательное значение, то элемент будет перевернут. Например, чтобы перевернуть элемент вокруг его обеих центральных осей, нужно применить следующее объявление:

```
transform: scale(-1);
```

Получится изображение, показанное на рис. 10.4, *слева*. Можно также перевернуть элемент вокруг одной центральной оси. На правом изображении на рис. 10.4 изображение перевернуто только вокруг его горизонтальной центральной оси. Среднее изображение получено за счет поворота изображения вокруг его вертикальной оси:

```
transform: scale(-1,1);
```

Похоже на эффект отражения элемента. Весьма забавно!

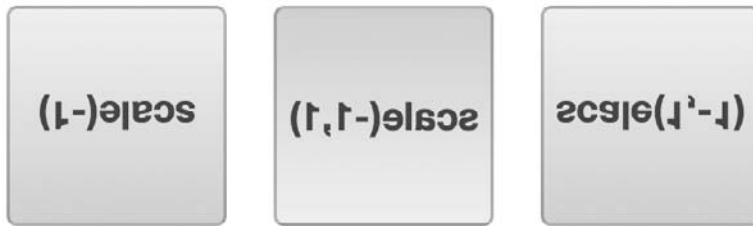


Рис. 10.4. Чтобы запутать или обескуражить посетителей вашего сайта, в языке CSS есть масса разных способов. Левый квадрат перевернут вокруг горизонтальной центральной оси, средний — вокруг вертикальной, а правый — вокруг обеих осей

Перемещение

Функция `translate` свойства `transform` перемещает элемент из его текущей позиции на некоторое расстояние вправо/влево и вверх/вниз. Пользы от этой функции как таковой немного. Как выяснится далее, когда браузер перемещает элемент, он не перестраивает контент страницы, а отображает его так, как будто элемент никуда не перемещался. Соответственно, если с использованием функции `translate` перемещается контейнер `div` или другой элемент, браузер оставляет пустое пространство там, где этот элемент отобразился без преобразования, а затем визуализирует элемент в его новой позиции (рис. 10.5). Если нужно лишь разместить элемент на странице, можно воспользоваться абсолютным или относительным позиционированием (см. главу 14).

Трансформируемые элементы могут...

... могут перекрыть другой контент на странице

`translate(200px,150px)`

Рис. 10.5. Функция `translate` перемещает элемент на указанное количество пикселей, единиц `em` или процентов от его обычного положения на странице

Но перемещение пригодится, когда нужно изобразить небольшое движение в ответ на помещение поверх указателя мыши или на щелчок кнопкой мыши. Например, во многих конструкциях пользовательского интерфейса при нажатии кнопки она смещается немного вниз и влево, имитируя вид реальной объемной кнопки. Применительно к ссылке этот эффект можно симитировать с помощью функции `translate` и состояния `:active`:

```
.button:active {  
  -webkit-translate(1px,2px);  
  -ms-translate(1px,2px);  
  translate(1px,2px);  
}
```

Функции `translate` передаются два значения: первое определяет величину горизонтального, а второе — вертикального перемещения. В данном примере щелчок на элементе с классом `.button` вызывает перемещение этого элемента на один пиксел вправо и на два пиксела вниз. Чтобы элемент переместился влево, нужно для первого значения использовать отрицательное число, применение отрицательного числа в качестве второго значения приведет к перемещению элемента вверх.

Но можно применять не только значения в пикселах. Допустимы любые значения, используемые в CSS для указания длины — `px`, `em`, `%` и т. д.

В CSS предоставляются также две дополнительные функции для перемещения элемента только влево или вправо — `translateX` и только вверх или вниз — `translateY`. Например, для перемещения элемента вверх на `.5em` используется функция `translateY`:

```
transform: translateY(-.5em);
```

Очень интересный эффект от применения функции `translate` можно получить при ее совместном использовании с переходами CSS. Это сочетание позволяет

анимировать перемещение элемента, продемонстрировав его путешествие по экрану. Вскоре вы увидите, как это делается.

Скашивание

Скашивание элемента можно осуществить по его горизонтальной и вертикальной осям: это придает элементу трехмерное представление (рис. 10.6). Например, для скашивания всех вертикальных линий влево на 45° (как на первом изображении на рис. 10.6) нужно написать следующий код:

```
transform: skew(45deg, 0);
```

Чтобы придать элементу такое же скашивание, но по оси Y (среднее изображение на рис. 10.6), нужно написать следующий код:

```
transform: skew(0,45deg);
```

Можно скосить элемент по двум осям одновременно. Например, следующий код приводит к результату, демонстрируемому третьим изображением, показанным на рис. 10.6:

```
transform: skew(25deg,10deg);
```

Первое значение задает значение угла в диапазоне от 0deg до 360deg , действующее в направлении против часовой стрелки от верхней части элемента. Например, на первом изображении (см. рис. 10.6) 45° представлены линией, нарисованной из центра элемента и повернутой на 45° против часовой стрелки (вниз и влево).

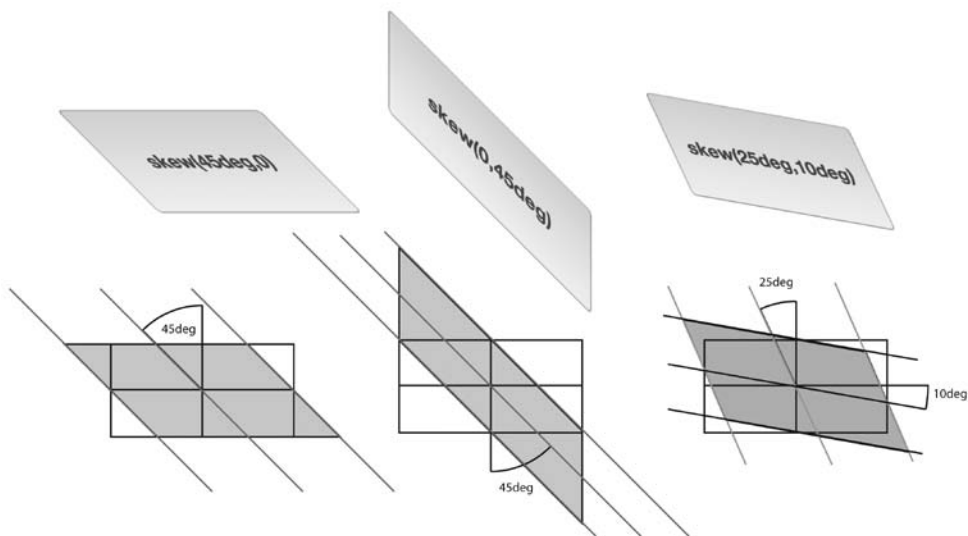


Рис. 10.6. Функция `skew` — один из способов имитации трехмерного отображения

ПРИМЕЧАНИЕ

Не забудьте в окончательном варианте CSS-кода добавить вендорные префиксы для различных браузеров: `-webkit-transform` и `-ms-transform`.

Вторым значением также является значение угла в диапазоне от 0deg до 360deg. Но этот угол действует в направлении по часовой стрелке с правой стороны элемента. Среднее изображение на рис. 10.6 является примером 45-градусного скашивания по всем горизонтальным линиям.

ПРИМЕЧАНИЕ

Чтобы получить визуальное представление о преобразованиях с помощью каскадных таблиц стилей, зайдите на сайт с интерактивным средством визуализации по адресу tinyurl.com/ydvlcc2.

Как и в случае с `translate` и `scale`, в CSS предлагаются отдельные функции для осей *X* и *Y*: `skewX` и `skewY`.

ПРИМЕЧАНИЕ

В CSS предлагается еще один метод преобразования под названием `matrix`. Он представляет собой числовой массив, похожий на тот, что используется в углубленном курсе алгебры. Этот метод не всегда понятен интуитивно и над ним придется поломать голову. Но если изучать принципы его работы не хочется, то наиболее понятное объяснение матриц преобразований можно найти на сайте tinyurl.com/qcj8jzj.

Множественные преобразования

Но вы не ограничены только одним преобразованием. Изображение можно одновременно масштабировать и скашивать, вращать и перемещать или использовать любые из четырех различных преобразований. Для этого нужно добавить через запятую дополнительные функции к свойству `transform`. Например, повернуть элемент на 45° и увеличить его размер вдвое можно с помощью такого объявления:

```
transform: rotate(45deg) scale(2);
```

А вот пример всех четырех преобразований, применяемых одновременно:

```
transform: skew(45deg,0deg) scale(.5) translate(400px,500px) rotate(90deg);
```

Браузер будет применять все эффекты в порядке следования функций. Например, во втором примере элемент сначала будет скошен, затем подвергнется масштабированию, после чего будет перемещен и наконец подвергнется вращению. Порядок не играет роли, если только не используется перемещение. Поскольку при перемещении изменяется местоположение элемента, то, если, к примеру, поместить перемещение перед вращением, браузер сначала переместит элемент, а затем применит к нему вращение. Поскольку сначала элемент перемещается, точка, вокруг которой он будет вращаться, изменится. С другой стороны, если сначала будет применено вращение, то затем элемент, уже подвергшийся вращению, будет перемещен на определенное расстояние относительно его центра (который теперь будет находиться в новой позиции).

Точка преобразования

Обычно, когда к элементу применяется преобразование, в качестве точки начала преобразования браузер использует центр элемента. Например, при вращении элемента браузер поворачивает его вокруг центральной точки (рис. 10.7, *слева*).

Но в CSS разрешается изменять точку преобразования, используя свойство `transform-origin`. Это свойство работает точно так же, как и ранее рассмотренное свойство `background-position`; в качестве значения можно указывать ключевые слова, абсолютные и относительные значения в единицах `em` и процентах.

Например, чтобы повернуть контейнер `div` вокруг его верхней левой точки (см. рис. 10.7, *посередине*), можно воспользоваться ключевыми словами `left` и `top`:
`transform-origin: left top;`

Можно также использовать пиксельные значения:

```
transform-origin: 0 0;
```

или проценты:

```
transform-origin: 0% 0%;
```

Точно так же для вращения элемента вокруг его нижнего правого угла (см. рис. 10.7, *справа*) используются ключевые слова `right` и `bottom`:

```
transform-origin: right bottom;
```

что также эквивалентно следующему объявлению:

```
transform-origin: 100% 100%;
```

При использовании пикселей, единиц `em` или процентных значений первое число означает горизонтальную, а второе — вертикальную позицию.

ПРИМЕЧАНИЕ

Свойство `transform-origin` не влияет на элементы, которые подвергаются только перемещению с помощью функции `translate`.

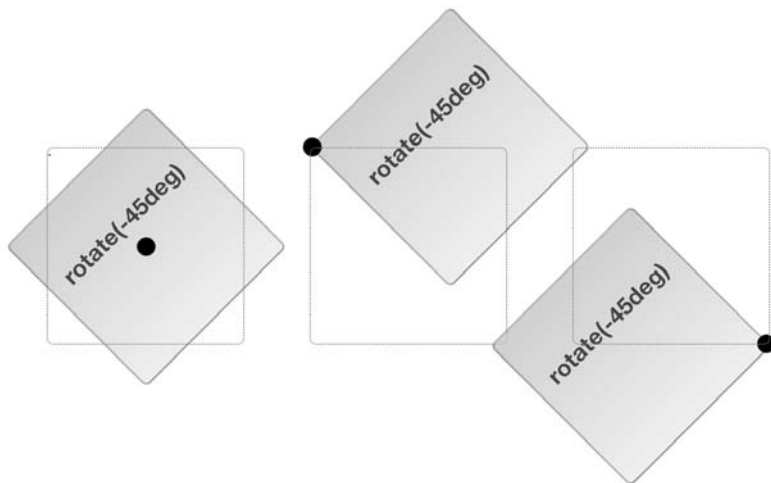


Рис. 10.7. Присваивание значения свойству `transform-origin` изменяет точку элемента, в которой будет применено преобразование. Пунктирными квадратами обозначен элемент, каким бы он был без вращения (его обычная позиция на странице)

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Трехмерные преобразования

В языке CSS предлагается также более сложный тип преобразования. Трехмерные преобразования позволяют имитировать трехмерное пространство на плоском экране монитора, планшетного компьютера или смартфона.

Краткое введение в трехмерные преобразования дано по адресу tinyurl.com/oqyfcprz, а более детальное описание и несколько примеров — по адресу tinyurl.com/d68z7ud. Посмотреть удачные примеры трехмерных преобразований в действии можно на следующих сайтах:

- один из первых примеров демонстрации возможностей трехмерных преобразований — страница [Morphing Power Cubes \(tinyurl.com/3n9qr64\)](http://Morphing Power Cubes (tinyurl.com/3n9qr64)) — здесь вы увидите вращающийся куб, который может превращаться во вращающийся набор плиток;
- еще один пример эффекта трехмерного транспонирования (эффект выглядит так, будто вы переворачиваете игральную карту, чтобы увидеть ее рубашку) находится по адресу tinyurl.com/oajfynt.

Переходы

Преобразованиями, конечно, можно позабавить посетителей (особенно функцией `rotate`), но по-настоящему они оживят вашу страницу в сочетании с переходами CSS. *Переход* представляет собой анимацию смены одного набора свойств CSS другим за определенный промежуток времени. Например, можно задать вращение баннера на 360° в течение двух секунд. Чтобы выполнить переход, потребуется следующее.

- **Два стиля.** Один стиль должен представлять начальный вид элемента, например красную кнопку, а второй — его конечный вид — синюю кнопку. О процессе анимации изменения между двумя состояниями позаботится браузер (например, об изменении цвета кнопки с красного на синий).
- **Свойство `transition`.** С помощью CSS добавляется свойство `transition` — секретная приправа, позволяющая осуществить анимацию. Как правило, свойство `transition` применяется к исходному стилю, который определяет внешний вид элемента до начала анимации.
- **Инициатор.** Инициатор представляет собой действие, вызывающее переход от одного стиля к другому. В CSS для запуска анимации можно применять несколько псевдоклассов. Наиболее часто используется псевдокласс `:hover`. С его помощью можно анимировать изменение обычного вида элемента на тот вид, когда посетитель устанавливает поверх элемента указатель мыши. Как это делается, вы увидите совсем скоро. Кроме того, можно использовать такие псевдоклассы, как `:active` (связанный с щелчком кнопкой мыши на элементе), `:target` (связанный с элементом, ставшим целью перехода по ссылке) и `:focus` (связанный с переходом по ссылке с помощью клавиши `Tab` или щелчком на элементе формы или с переходом на этот элемент с помощью клавиши `Tab`). Кроме этого, для динамической смены стиля любого элемента можно воспользоваться JavaScript-кодом (см. врезку «Использование JavaScript-сценариев для запуска переходов» далее).

Когда инициатор больше не применяется, то есть когда, к примеру, посетитель убирает указатель мыши с кнопки навигации, браузер возвращает элементу его прежний стиль и анимирует весь этот процесс. Иными словами, вам нужно лишь установить переход для элемента один раз, а анимацию перехода от одного стиля к другому и обратно к исходному стилю браузер возьмет на себя.

Браузер не может анимировать каждое свойство CSS, но у вас в распоряжении остается все же весьма длинный список свойств, на которых можно остановить свой выбор. Кроме таких преобразований, как `rotate`, `scale`, `translate` и `skew`, о которых только что шла речь, можно анимировать такие свойства, как `color`, `background-color`, `border-color`, `border-width`, `font-size`, `height`, `width`, `letter-spacing`, `line-height`, `margin`, `opacity`, `padding`, `word-spacing`, свойства позиционирования — `top`, `left`, `right` и `bottom`, речь о которых пойдет в главе 15, а также многие другие свойства. Их полный список можно найти по адресу tinyurl.com/dxjbdhd.

ПРИМЕЧАНИЕ

Переходы CSS работают в большинстве браузеров. Но когда речь заходит об Internet Explorer, приходится заметить, что переходы этот браузер поддерживает, только начиная с версии 10. Если вы все еще хотите поддерживать более ранние версии браузера Internet Explorer, используйте переходы только как средство придания выразительности. При этом в большинстве случаев все будет выглядеть вполне приемлемо: Internet Explorer 9 и более ранние версии смогут обеспечить переключение между двумя стилями (например, показать стиль `:hover`) без анимации данного изменения.

Добавление перехода

В основе переходов CSS лежат четыре свойства, который управляют тем, какие свойства анимировать, сколько времени займет анимация, какой тип анимации будет использован и какой будет необязательная отсрочка перед началом анимации. Рассмотрим простой пример. Предположим, что нужно анимировать изменение фонового цвета кнопки навигации с оранжевого на синий при помещении поверх кнопки указателя мыши. Начать нужно с двух стилей, необходимых для переключения между этими двумя цветами. Можно, к примеру, применить к ссылке класс `.navButton`, а затем создать следующие два стиля:

```
.navButton {
  background-color: orange;
}

.navButton:hover {
  background-color: blue;
}
```

Эти стили будут работать в любом браузере. При помещении указателя мыши поверх кнопки навигации фоновый цвет этой кнопки изменится с оранжевого на синий. Но это изменение произойдет мгновенно. Чтобы цвет изменился в режиме анимации за одну секунду, нужно к стилю `.navButton` добавить два новых свойства:

```
.navButton {  
  background-color: orange;  
  transition-property: background-color;  
  transition-duration: 1s;  
}  
  
.navButton:hover {  
  background-color: blue;  
}
```

Свойство `transition-property` указывает на анимируемое свойство. Можно указать одно свойство (как в показанном выше примере), воспользоваться ключевым словом `all` для анимации всех изменяемых CSS-свойств или воспользоваться списком с запятой в качестве разделителя для указания более чем одного свойства (но не всех свойств). Предположим, что вы создаете стиль `:hover`, чтобы изменялись сразу цвет текста, фоновый цвет и цвет границы. Все эти три свойства указываются в виде следующего списка:

```
transition-property: color, background-color, border-color;
```

или, чтобы упростить код, указывается ключевое слово `all`:

```
transition-property: all;
```

Во многих случаях при анимации каждого изменения средствами каскадных таблиц стилей лучше использовать ключевое слово `all`, в результате чего создается привлекательный визуальный эффект.

Чтобы указать длительность анимации, применяется свойство `transition-duration`. Ему передается значение либо в секундах, либо в миллисекундах (тысячных долях секунды). Например, чтобы переход занимал полсекунды, можно задать либо код:

```
transition-duration: .5s;
```

либо код:

```
transition-duration: 500ms;
```

Можно даже указать длительность отдельно для каждого анимируемого свойства. Например, когда посетитель устанавливает указатель мыши поверх кнопки, может потребоваться, чтобы цвет текста изменялся быстрее, цвет фона изменялся немного медленнее, а цвет границы изменялся заметно медленнее. Для этого нужно перечислить анимируемые свойства, используя `transition`, а затем указать значения длительности в свойстве `transition-duration`:

```
transition-property: color, background-color, border-color;  
transition-duration: .25s, .75s, 2s;
```

Порядок перечисления значений длительности должен соответствовать порядку перечисления свойств. Следовательно, в показанном выше примере `.25s` относится к свойству `color`, `.75s` — к свойству `background-color`, а `2s` — к свойству `border-color`. Если поменять свойства местами, их значения длительности переходов изменятся.

Тайминг перехода

Чтобы анимированный переход заработал, нужно только установить значения для свойств `transition-property` и `transition-duration`. Но с помощью свойства `transition-timing-function` можно также контролировать и скорость анимации. В предназначении этого свойства нетрудно и запутаться: оно управляет не длительностью анимации (для этого есть свойство `transition-duration`), а ее *скоростью*.

Свойство `transition-timing-function` может использовать одно из пяти ключевых слов: `linear`, `ease`, `ease-in`, `ease-out` и `ease-in-out`. Если функцию управления скоростью не задавать, браузер будет применять метод `ease`, при котором анимация начинается медленно, ускоряется к середине и замедляется к концу, предоставляя более естественное изменение. Вариант `linear` предоставляет равномерное изменение на протяжении всего периода анимации. Ни один из вариантов не имеет заметных преимуществ перед другими, они лишь предлагают различную общую картину, и для определения своих предпочтений нужно просмотреть все варианты.

Чтобы воспользоваться этим свойством, нужно добавить свойство `transition-timing-function` и требуемый метод:

```
transition-timing-function: ease-in-out;
```

ПРИМЕЧАНИЕ

В случае с различными функциями регулирования скорости перехода лучше один раз увидеть, чем сто раз услышать. Посетите сайт tinyurl.com/36edxvd, чтобы увидеть наглядное сравнение пяти методов тайминга.

Для свойства `transition-timing-function` можно также использовать значение так называемой кубической кривой Безье (`cubic-bezier`). Она определяет график воспроизведения анимации по времени (рис. 10.8). Если вам приходилось работать с такими графическими редакторами, как Adobe Illustrator, то вы, наверно, знакомы с кривыми Безье. Кривизной линии можно управлять путем настройки двух контрольных точек: чем круче линия, тем быстрее анимация, а более пологая линия означает более медленное воспроизведение анимации. Например, кривая Безье, показанная на рис. 10.8, начинается с крутого участка (анимация начинается в быстром темпе), затем переходит к середине в более пологую фазу (анимация замедляется), а потом кривизна снова возрастает (анимация быстро подходит к своей завершающей стадии). Для создания анимации такого профиля нужно добавить следующую строку кода:

```
transition-timing-function: cubic-bezier(.20, .96, .74, .07);
```

Вам совершенно не обязательно забивать себе голову кубическими кривыми Безье. Лучше воспользоваться одним из многочисленных интерактивных средств для создания и тестирования различных функций тайминга. Одним из лучших считается средство `Caesar`, разработанное Мэтью Лейном: tinyurl.com/4vebxwg. Интерактивное средство `Caesar` превращает создание кубических кривых Безье в сущий пустяк: нужно для изменения кривизны перетаскивать нижнюю левую и верхнюю правую контрольные точки. Чем круче линия, тем быстрее воспроизводится анимация на ее участке.

Как и в случае со свойством `transition-duration`, к различным свойствам можно применять разные тайминги.

Ceaser CSS EASING ANIMATION TOOL Tweet

1. Choose an easing type and test it out with a few effects.
2. If you don't quite like the easing, grab a handle and fix it.
3. When you're happy, snag your code and off you go.

Now that we can use CSS transitions in all the modern browsers, let's make them pretty. I love the classic Penner equations with Flash and jQuery, so I included most of those. If you're anything like me, you probably thought this about the default easing options: "ease-in, ease-out...yawn." The mysterious cubic-bezier has a lot of potential, but was cumbersome to use. Until now. Also, touch-device friendly!

"If you are anything like me, we should be friends @matthewleh"

Note: Bugfixes have landed, so the newest Webkit now supports values above 1 and below 0. For the time being, I am including fallback code for older Webkit that is clamped between 0 and 1 when needed.

Animation (%)

Time

Easing: custom (drag the handles) ▾

Duration: 500

Effect: Left Width Height Opacity

Code snippets, short and long-hand:

```
-webkit-transition: all 500ms cubic-bezier(0.580, 0, 1.000, 1); /* older webkit */
-webkit-transition: all 500ms cubic-bezier(0.580, -0.600, 1.000, 1.560);
-moz-transition: all 500ms cubic-bezier(0.580, -0.600, 1.000, 1.560);
-ms-transition: all 500ms cubic-bezier(0.580, -0.600, 1.000, 1.560);
-o-transition: all 500ms cubic-bezier(0.580, -0.600, 1.000, 1.560);
transition: all 500ms cubic-bezier(0.580, -0.600, 1.000, 1.560); /* custom */

-webkit-transition-timing-function: cubic-bezier(0.580, 0, 1.000, 1); /* older webkit */
-webkit-transition-timing-function: cubic-bezier(0.580, -0.600, 1.000, 1.560);
-moz-transition-timing-function: cubic-bezier(0.580, -0.600, 1.000, 1.560);
-ms-transition-timing-function: cubic-bezier(0.580, -0.600, 1.000, 1.560);
-o-transition-timing-function: cubic-bezier(0.580, -0.600, 1.000, 1.560);
transition-timing-function: cubic-bezier(0.580, -0.600, 1.000, 1.560); /* custom */
```

If this saves you time, or blows your mind, consider making a [Donation](#) to keep these projects alive.

Рис. 10.8. Создание кубической кривой Безье позволяет управлять широким диапазоном интересных функций тайминга анимации. Можно задать очень медленное начало и быстрое завершение или наоборот

Отсрочка перехода

И наконец, можно оттянуть время начала анимации перехода, воспользовавшись свойством `transition-delay`. Например, если следует подождать полсекунды, прежде чем начать анимацию, можно добавить следующий код:

```
transition-delay: .5s;
```


ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ**Использование JavaScript-сценариев для запуска переходов**

Переходы CSS — простые в применении средства анимации, встроенные непосредственно в браузеры (по крайней мере в *большинство* браузеров). Сложные эффекты можно создавать, определяя для этого набор исходных и конечных свойств CSS, возлагая все остальное на браузер. К сожалению, для запуска этих переходов у вас в распоряжении имеется всего лишь несколько селекторов. Чаще всего для изменения элемента при нахождении поверх него указателя мыши используется псевдокласс `:hover`. Для анимации элемента формы при щелчке на нем кнопкой мыши можно также воспользоваться псевдоклассом `:focus` (например, можно изменить высоту текстовой области при получении фокуса, а затем сжиматься до нескольких строк при щелчке за пределами этой области).

Но в языке CSS нет, к примеру, псевдокласса `:click`, поэтому при желании запустить переход по щелчку на элементе сделать это средствами каскадных таблиц стилей не получится. Не удастся также запустить анимацию какого-нибудь элемента при нахождении указателя мыши поверх другого элемента. . . . если использовать только CSS. Но переходы CSS работают

при любом изменении свойства CSS, даже если это делается с помощью JavaScript.

JavaScript — язык программирования, позволяющий добавлять интерактивность веб-страницам, создавать динамические пользовательские интерфейсы и осуществлять множество других полезных действий. Но JavaScript-код можно также использовать для простого добавления или удаления класса элемента или изменения какого угодно свойства CSS. Благодаря JavaScript можно добавить класс к изображению, когда посетитель щелкает на ссылке. Показать увеличенный вариант изображения. Этот новый класс просто увеличивает масштаб изображения (с помощью свойства `scale`). Добавляя к изображению переход, вы тут же получите анимацию.

Изучение JavaScript — довольно обширная тема, заслуживающая отдельной книги, которой как раз может стать «JavaScript и jQuery. Исчерпывающее руководство» этого же автора. Но если для начала вы захотите освоить простейший способ использования переходов CSS, запускаемых с помощью JavaScript, прочитайте краткое руководство на сайте tinyurl.com/qb9mrv7.

Чаще всего отсрочка перехода для всех свойств вам не понадобится, поскольку в этом случае снижается эффект интерактивности. В конце концов, преднамеренно заставлять посетителей ждать целую секунду, пока они увидят изменения в кнопке при наведении на нее указателя мыши, не очень хорошая идея. Но если анимируется сразу несколько свойств, может понадобиться выждать с изменением одного свойства, пока не закончится анимация перехода других свойств. Предположим, что у вас есть кнопка, чей цвет фона и текста нужно изменить, а затем неожиданно поменять цвет ее границы после завершения изменения первых двух свойств. Это можно сделать с помощью следующего кода:

```
.navButton {
  color: black;
  background-color: #FF6603;
  border: 5px solid #660034;
  transition-property: color, background-color, border-color;
  transition-duration: 1s, 1s, .5s;
  transition-delay: 0, 0, 1s;
}
.navButton:hover {
  color: white;
```

```
background-color: #660034;  
border-color: #FF6603;  
}
```

Как и в случае применения свойства `transition-duration`, для каждого свойства можно указать свое время отсрочки. Порядок перечисления показателей времени должен соответствовать порядку перечисления этих свойств для `transition-property`. Например, в предыдущем коде отсрочки для переходов цвета текста и цвета фона отсутствуют, но зато есть односекундная отсрочка до начала изменения цвета границы.

СОВЕТ

Обычно свойства перехода помещаются в начальный (`.navButtonon` в предыдущем примере), а не в конечный стиль (`.navButton:hover`). Но при использовании раскрывающегося меню CSS возникают некоторые сложности. Проблема в том, что при задании таких меню с помощью CSS они слишком быстро пропадают при случайном выходе указателя мыши за их пределы. Но с помощью свойства `transition-delay` можно заставить меню быстро появляться и медленно исчезать. Для этого добавьте в исходный стиль следующий код:

```
transition-delay: 1s;
```

Затем добавьте отсрочку к стилю `:hover`:

```
transition-delay: 0;
```

Каким бы странным этот код ни показался, он заставляет переход `:hover` происходить немедленно, без отсрочки. А вот возвращение к обычному стилю (при котором меню исчезает) занимает одну секунду. За это время посетитель успеет вернуть указатель мыши обратно на меню, пока оно не исчезло.

Сокращенная запись свойства `transition`

Запись всех свойств по отдельности — `transition-property`, `transition-duration`, `transition-timing-function` и `transition-delay` — может утомить. Особенно если потребуется создать и версию каждого из них с вендорным префиксом. На этот случай есть более быстрый способ задания переходов — свойство `transition`. Оно связывает все другие свойства. Нужно лишь перечислить через запятые свойство, длительность, тайминг и отсрочку. Например, для анимации всех свойств CSS на одну секунду, используя функцию `ease-in` и полусекундную отсрочку, нужно написать следующий код:

```
transition: all 1s ease-in .5s;
```

Список должен состоять из ключевого слова `all` или какого-нибудь определенного свойства CSS, а также из длительности, а тайминг и отсрочку можно не указывать. По умолчанию в качестве функции тайминга указывается `ease-in`, а отсрочка не используется. То есть, если нужно просто анимировать переход всех свойств CSS на одну секунду, нужно написать следующий код:

```
transition: all 1s;
```

Если нужно анимировать изменение фоновой окраски, включите в список именно это свойство:

```
transition: background-color 1s;
```

Здесь можно включить в список только одно свойство CSS, поэтому, если требуется анимировать сразу несколько свойств CSS (но не все), можно воспользоваться их списком с запятой в качестве разделителя, где элементами будут разделенные пробелами свойства переходов. Возьмем один из предыдущих примеров, где свойство `border-color` анимировалось отдельно от цвета текста и цвета фона. Его код можно переписать следующим образом:

```
transition: color 1s, background-color 1s, border-color .5s 1s;
```

Чтобы код было легче читать, многие веб-разработчики помещают каждый переход в отдельной строке:

```
transition: color 1s;
background-color 1s;
border-color .5s 1s;
```

Нужно только не забывать разделять переходы запятыми, а всю связку завершать точкой с запятой.

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Создание плавной анимации

Добавление анимации к различным свойствам может замедлить работу браузера. Используйте ли вы переходы или анимацию средствами каскадных таблиц стилей, браузерам необходимо выполнить много вычислений, чтобы анимировать изменения в свойствах CSS. Слишком много анимации и переходов, использующихся одновременно, могут существенно снизить скорость работы браузера или даже привести к его зависанию. Особенно это касается мобильных устройств и планшетов, процессоры которых гораздо менее производительны, чем в настольных компьютерах и ноутбуках.

Тем не менее существует четыре параметра, которые браузеры могут анимировать, не прибегая к значительным вычислительным затратам процессора: непрозрачность (см. приложение 1), а также параметры `translate`, `scale` и `rotate` свойства `transform` (см. предыдущий раздел). Эти четыре свойства обрабатываются более эффективно, чем другие, так что любые переходы или анимация, созданные с их помощью, будут осуществляться плавно (чтобы узнать технические подробности, перейдите по ссылке tinyurl.com/pjdx8le).

Кроме того, для ускорения расчетов можно переложить функции обработки анимации на *графический процессор (GPU)* (иными словами, видеокарту). Графические

процессоры — высокопроизводительные устройства, которые могут выполнять определенные типы вычислений гораздо быстрее, чем центральный процессор компьютера. Вы можете инструктировать браузер перенаправить расчет стиля в графический процессор, добавив в него свойство трехмерного преобразования. Например, если вы планируете анимировать цвет фона элемента, когда посетитель сайта установит поверх него указатель мыши, можно создать стиль с оригинальным цветом фона, который будет выглядеть следующим образом:

```
.highlight {
  background-color: rgb(231,0,23);
  transition: background-color 1s;
  transform: translateZ(0);
}
.highlight:hover {
  background-color: rgb(0,0,255);
}
```

Строка `transform: translateZ(0)` не вносит никаких визуальных изменений. Она инструктирует браузер переместить элемент на 0 пикселей вдоль оси *Z*; иными словами, элемент не двигается вообще. Однако, поскольку он использует трехмерное преобразование, браузер обрабатывает этот стиль с помощью графического процессора. Суть заключается в том, что анимация

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

может воспроизводиться более плавно, потому что графический процессор более производительный.

Но прежде, чем вы начнете применять свойство `transform: translateZ(0)` к каждому стилю, необходимо учесть, что графический процессор может обрабатывать много команд, однако передача большого количества визуальных эффектов может замедлить браузер.

Кроме того, большое количество анимации и переходов может оказывать негативное влияние на производительность страницы, особенно на мобильных устройствах пользователей. Более тщательно проверьте все переходы и эффекты анимации в различных браузерах и на мобильных устройствах, чтобы убедиться, что ваш сайт работает должным образом.

Анимация

В CSS предоставляется еще один, более богатый на свойства, механизм создания анимации. С помощью CSS-переходов можно анимировать только переход от одного набора свойств CSS к другому. Технология *анимации*, предусмотренная в CSS, позволяет анимировать переходы от одного набора свойств к другому, затем к третьему и т. д. Кроме того, можно задать повторяющуюся анимацию, приостановить ее выполнение при помещении указателя мыши поверх элемента и даже повернуть ее вспять, когда анимация завершится.

Анимация CSS сложнее переходов, но у нее есть дополнительное преимущество в том, что не нужен обязательный инициатор. Наряду с тем, что анимацию можно добавить к состоянию `:hover` для ее проигрывания при нахождении указателя мыши поверх элемента, запустить анимацию можно и при загрузке страницы. Тем самым можно привлечь внимание к баннеру или логотипу, анимируемому, когда посетитель заходит на сайт.

ПРИМЕЧАНИЕ

Анимация CSS, как и переходы, не поддерживается в Internet Explorer 9 и более ранних версиях. Но эти эффекты поддерживаются большинством других современных браузеров.

При создании анимации первым делом создаются ключевые кадры. *Ключевой кадр* в анимации — это отдельный кадр анимации, определяющий внешний вид сцены. Предположим, первый ключевой кадр содержит изображение мяча на одной половине футбольного поля. Добавив второй ключевой кадр, можно определить конечную точку анимации, например мяч в воротах на другой половине поля. После этого браузер создаст анимацию между этими двумя ключевыми кадрами, прорисовывая все промежуточные фазы, отображающие перемещение мяча по полю на его пути к голу в воротах.

Ели вы подумали, что в переходы заложена такая же идея, то так оно и есть. В переходе определяются два стиля, а на браузер возлагается задача анимировать изменения от одного стиля к другому. В этом смысле каждый из этих стилей можно представить в качестве ключевых кадров. Но анимация CSS позволяет определять *множество* ключевых кадров и создавать более сложные эффекты: например, футбольный мяч, перемещающийся с одной стороны поля к игроку, к другому игроку, а затем в ворота.

Создание анимации проходит в два приема.

1. Определение анимации.

Включает настройку ключевых кадров со списком анимируемых CSS-свойств.

2. Применение анимации к элементу.

После определения анимации ее можно применить к любому количеству элементов страницы. Можно даже задать для каждого элемента разные тайминги, отсрочки и другие свойства анимации. То есть одну и ту же анимацию на странице можно использовать с разными настройками несколько раз.

Определение ключевых кадров

Первым делом нужно настроить ключевые кадры. Реализующий их синтаксис может показаться немного странным, но его основная структура имеет следующий вид:

```
@keyframes имяАнимации {
  from {
    /* здесь перечисляются свойства CSS */
  }

  to {
    /* здесь перечисляются свойства CSS */
  }
}
```

Сначала указывается ключевое слово `@keyframes`, за которым следует имя — название анимации. В дальнейшем это имя будет использоваться при применении анимации к элементу страницы, поэтому оно должно быть описательным, например `fadeOut` или `fadeIn`.

ПРИМЕЧАНИЕ

Само понятие `@keyframes` является не свойством CSS, а так называемым правилом. В CSS есть и другие правила, например инструкция `@import` для загрузки внешней таблицы стилей из другой таблицы стилей и `@media` — для определения стилей для различных типов аппаратуры представления информации, к примеру принтера или устройств с экранами различного размера и разрешением.

Затем добавляются не менее двух ключевых кадров. В данном примере слова `from` и `to` используются для создания начального ключевого кадра (`from`) и конечного ключевого кадра (`to`). Внутри каждого ключевого кадра находится одно или несколько свойств CSS, словно создается стиль. Фактически каждый ключевой кадр можно представить в виде простого стиля, содержащего одно или несколько свойств каскадных таблиц стилей. Предположим, что нужно создать анимацию постепенного появления элемента. Можно начать со значения `opacity`, равного 0 (невидимый), и закончить значением этого свойства, равным 1 (полностью видимый):

```
@keyframes fadeIn{
  from {
    opacity: 0;
  }
}
```

```
to {
  opacity: 1;
}
}
```

Но вы не ограничены использованием только двух ключевых кадров. С помощью процентных значений можно определить несколько ключевых кадров. Процентное значение определяет момент общей длительности анимации, в который должно произойти изменение. Предположим, что нужно создать эффект изменения фона элемента с желтого на синий, а затем на красный. Для этого можно написать следующий код:

```
@keyframes backgroundGlow {
  from {
    background-color: yellow;
  }
  50% {
    background-color: blue;
  }
  to {
    background-color: red;
  }
}
```

В данном случае синий фон появится в середине анимации. Если нужно, чтобы желтый цвет присутствовал в фоне дольше, а синим он становился после трех четвертей времени анимации, используйте значение не 50%, а 75%. Таким же образом можно продолжать добавлять дополнительные ключевые кадры (например, на 25%, 66% и т. д.).

ПРИМЕЧАНИЕ

Ключевое слово `from` можно заменить значением 0%, а ключевое слово `to` — значением 100%.

Но одним свойством каскадных таблиц стилей можно не ограничиваться. Внутри ключевого кадра может находиться любое количество пригодных к анимации свойств — `background-color`, `opacity`, `width`, `height` и т. д.:

```
@keyframes growAndGlow {
  from {
    background-color: yellow;
  }

  50% {
    transform: scale(1.5);
    background-color: blue;
  }

  to {
    transform: scale(3);
    background-color: red;
  }
}
```

В этом примере наряду с трехкратным изменением цвета фона происходит увеличение масштаба элемента, размеры которого в итоге становятся втрое больше первоначальных.

Можно также усложнить использование процентных значений, добавив несколько таких значений к одному набору свойств CSS. Это может пригодиться в двух случаях: во-первых, при необходимости довести анимацию до некоторого момента, сделать паузу, а затем продолжить анимацию. Предположим, что вам нужно, чтобы сначала у элемента `div` был желтый цвет фона. Затем нужно, чтобы цвет сменился на синий, некоторое время оставался синим, а затем сменился на красный. То есть нужна пауза в середине анимации, при которой цвет не меняется, а затем нужна новая смена цвета. Эта задача решается с помощью следующего кода:

```
@keyframes glow {
  from {
    background-color: yellow;
  }
  25%, 75% {
    background-color: blue;
  }
  to {
    background-color: red;
  }
}
```

Обратите внимание на процентные значения 25%, 75% в строке 5. Они означают, что на 25 % от длительности всей анимации фоновый цвет элемента станет синим. Но он будет оставаться синим до момента 75 % длительности анимации. То есть от отметки 25 % до отметки 75 % фон будет сохранять синий цвет, перед тем как к окончанию анимации превратиться в красный. Если длительность этой анимации будет четыре секунды, то в средние две секунды у элемента будет синий фон.

Процентное значение можно также указывать при необходимости использования одного и того же набора свойств CSS для различных частей анимации. Предположим, нужно выполнить еще одну анимацию фоновых цветов, но на этот раз менять цвет от желтого к синему, затем к оранжевому, после чего к синему, к оранжевому и к красному. Синий и оранжевый цвета фигурируют в этом перечне дважды, поэтому вместо многократного упоминания об этих свойствах фоновых цветов можно воспользоваться следующим кодом:

```
@keyframes glow {
  from {
    background-color: yellow;
  }
  20%, 60% {
    background-color: blue;
  }
  40%, 80% {
    background-color: orange;
  }
  to {
```

```
background-color: red;
}
}
```

В данном случае фоновый цвет будет синим на отметке 20 %, превратится в оранжевый на отметке 40 %, затем опять в синий на отметке 60 % и, перед тем как стать в конце анимации красным, еще раз на отметке 80 % превратится в оранжевый.

К сожалению, как было сказано ранее, для использования анимации в браузерах Chrome, Safari и Opera вы должны указывать вендорные префиксы производителей. Другими словами, вам нужно создать одну анимацию для этих браузеров и другую (без вендорных префиксов) — для браузеров Firefox и Internet Explorer 10, для чего придется написать следующий код:

```
@-webkit-keyframes fadeIn {
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}
@keyframes fadeIn{
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}
```

Обратите внимание на два дефиса: один между символом @ и вендорным префиксом и второй — между вендорным префиксом и словом `keyframes`.

Применение анимации

Когда набор ключевых кадров определится, анимация будет готова. Чтобы заставить ее работать, нужно применить ее к какому-нибудь элементу страницы. Анимацию можно добавить к любому стилю любого элемента страницы. Если добавить анимацию к стилю, который сразу применяется к элементу, например к стилю элемента `h1`, то она будет активизирована при загрузке страницы. Этот прием можно применить для добавления на страницу вводной анимации, которая увеличивает масштаб логотипа, помещая его в верхний левый угол страницы, или подсвечивает некий блок информации, привлекая к нему внимание.

Кроме того, анимацию можно применить к одному из псевдоклассов, включая `:hover`, `:active`, `:target` или `:focus`, чтобы, к примеру, запустить анимацию при установке указателя мыши над ссылкой или при щелчке на элементе формы. И наконец, анимацию можно применить к классу и воспользоваться JavaScript-кодом для динамического применения этого класса в ответ на нажатие посетителем кнопки или щелчок на каком-нибудь другом элементе страницы.

В CSS предоставляется несколько свойств, связанных с анимацией, позволяющих управлять способом и временем проигрывания анимации (а также сокращенная запись, охватывающая все отдельные свойства). Как минимум, чтобы заставить анимацию выполняться, нужно указать имя, которое было присвоено исходной анимации (в правиле `@keyframes`), и длительность анимации.

Рассмотрим простой пример. Предположим, что нужно, чтобы при загрузке страницы плавно появился контейнер `div` с важным объявлением. Этому контейнеру был назначен класс с именем `announcement`:

```
<div class="announcement">
```

1. Создайте анимацию постепенного появления, задав правило `@keyframes`:

```
@keyframes fadeIn {  
  from { opacity: 0; }  
  to { opacity: 1; }  
}
```

СОВЕТ

Когда анимируется всего лишь одно свойство, его будет проще прочитать, если поместить код в одной строке:

```
from { opacity: 0; }
```

Но если анимируется множество свойств, то читать (и набирать) проще будет при распределении кода по нескольким отдельным строкам:

```
from {  
  opacity: 0;  
  color: red;  
  width: 50%;  
}
```

2. Примените эту анимацию к стилю, предназначенному для элемента `div`:

```
.announcement {  
  animation-name: fadeIn;  
  animation-duration: 1s;  
}
```

Свойство `animation-name` сообщает браузеру, какую анимацию нужно применить. Ему указывается то имя, которое представлялось анимации при выполнении шага 1. Свойство `animation-duration` устанавливает время, которое займет анимация от старта до финиша. В данном примере перечисляются лишь два свойства анимации, но вы можете (и, наверное, захотите) поместить в стиль и другие, не связанные с анимацией свойства. Например, в реальности к нашему стилю `.announcement` могут быть добавлены такие свойства, как `width`, `background-color`, `border` и т. д.

ПРИМЕЧАНИЕ

С технической точки зрения помещать имя анимации в кавычки ('fadeIn') не требуется, и в этом примере так не делается, но добавление кавычек поможет избежать конфликтов, возникающих при использовании в качестве имени анимации какого-нибудь ключевого слова CSS.

Как и в случае использования правила `@keyframes`, каждое из свойств анимации требует указания вендорных префиксов, поэтому показанный выше стиль `.announcement` для работы в максимально возможном количестве браузеров нужно переписать следующим образом:

```
.announcement {  
  -webkit-animation-name: fadeIn;  
  -webkit-animation-duration: 1s;  
  animation-name: fadeIn;  
  animation-duration: 1s;  
}
```

Возможно, покажется несколько неудобным определять анимацию с помощью правила `@keyframes` в одном месте, а затем применять ее в другом месте (в стиле), но после определения анимация может применяться несколько раз в любом количестве стилей. Например, можно создать общий тип анимации появления элемента на экране и применить этот тип к различным элементам. Более того, можно управлять анимацией для каждого стиля совершенно независимо, например, заголовок может быть виден в течение половины секунды, а другие элементы страницы — в течение пяти секунд.

Кроме того, к одному и тому же элементу можно применить более одной анимации. Предположим, вы создаете одну анимацию с именем `fadeIn`, чтобы анимировать появление элемента, и еще одну анимацию, `blink`, чтобы получить мигающий цвет фона. Чтобы применить к элементу обе анимации, нужно предоставить список имен с запятой в качестве разделителя:

```
animation-name: fadeIn, blink;
```

Чтобы задать этим анимациям разные длительности воспроизведения, нужно предоставить список данных параметров с запятой в качестве разделителя:

```
animation-name: fadeIn, blink;  
animation-duration: 1s, 3s;
```

Порядок применения параметров времени совпадает с порядком указания имен анимаций. Например, первая анимация получает время, указанное в списке первым. В показанном выше примере длительность `fadeIn` составляет одну секунду, а длительность `blink` — три секунды.

Можно также применить ряд других полезных свойств анимации, рассматриваемых далее.

Тайминг анимации

Как уже говорилось, свойство `animation-duration` позволяет управлять длительностью анимации. Как и в случае с переходами, для задания длительности можно использовать миллисекунды (например, `750ms`) или секунды (например, `.75s`).

Как и в случае с переходами, можно указывать функцию тайминга, которая управляет этой скоростью в заданном моменте анимации. Например, анимацию можно начинать медленно и завершать быстро, используя для этого кубическую кривую Безье или одно из предустановленных ключевых слов, определяющих ме-

год: `linear`, `ease`, `ease-in`, `ease-out`, `ease-in-out`. Все это работает точно так же, как и с переходами.

Свойство `animation-timing-function` можно использовать для управления всей анимацией или только для конкретных ключевых кадров. Например, чтобы применить функцию `ease-out` для представленной ранее анимации `fadeIn` (шаг 1 в предыдущем разделе), добавьте функцию тайминга к стилю `.announcement` (шаг 2 в предыдущем разделе):

```
.announcement {
  animation-name: fadeIn;
  animation-duration: 1s;
  animation-timing-function: ease-out;
}
```

Но можно также управлять функцией тайминга для анимации между ключевыми кадрами. Предположим, что создается анимация с тремя ключевыми кадрами с тремя различными цветами фона. Браузер будет осуществлять ее при превращении одного цвета в другой, а затем в третий. Возможно, вам захочется замедлить превращение первого цвета во второй с помощью кубической кривой Безье, а затем равномерно продолжить анимацию от середины и до ее окончания. Это можно сделать, добавив две функции тайминга: одну для первого ключевого кадра (она будет управлять анимацией от ключевого кадра 1 к ключевому кадру 2) и вторую — для второго ключевого кадра (для управления анимацией от второго ключевого кадра к третьему):

```
@keyframes growAndGlow {
  from {
    background-color: yellow;
    animation-timing-function: cubic-bezier(1, .03, 1, .115);
  }
  50% {
    transform: scale(1.5);
    background-color: blue;
    animation-timing-function: linear;
  }
  to {
    transform: scale(3);
    background-color: red;
  }
}
```

С помощью свойства `animation-delay` можно задержать начало анимации. Оно работает точно так же, как и `transition-delay`, использующееся для отсрочки переходов, и задает до начала анимации период ожидания на указанное количество миллисекунд или секунд. Например, если нужно подождать одну секунду, прежде чем инициировать постепенное появление на экране контейнера `div` с классом `announcement`, можно переписать стиль `.announcement` следующим образом:

```
.announcement {
  animation-name: fadeIn;
  animation-duration: 1s;
  animation-delay: 1s;
}
```

```
animation-delay: 1s;
}
```

Добавление отсрочки к анимации — неплохой способ привлечения внимания и добавления сюрприза к странице.

ПРИМЕЧАНИЕ

Свойства `animation-timing-function` и `animation-delay` для браузеров Chrome, Safari и Opera требуют указывать вендорный префикс `-webkit-`, поэтому не забудьте добавить его: `-webkit-animation-timing-function` и т. д.

Завершение анимации

С помощью каскадных таблиц стилей можно контролировать еще несколько аспектов анимации, включая ее повторение, направление, если она выполняется более одного раза, а также порядок форматирования браузером элемента по завершении анимации.

Переходы являются анимациями, выполняемыми однократно, например, когда указатель мыши находится поверх кнопки, она увеличивается в размерах. А вот анимации благодаря свойству `animation-iteration-count` могут запускаться сколько угодно раз или непрерывно. Если нужно запустить анимацию 10 раз (возможно, для десятикратного появления и исчезновения элемента), добавьте к анимируемому стилю следующий код:

```
animation-iteration-count: 10;
```

Обычно браузеры проигрывают анимацию только один раз, и если это все, что вам нужно, то свойство подсчета итераций можно оставить в покое. Если же нужно, чтобы анимация повторялась непрерывно, свойству `animation-iteration-count` передается ключевое слово `infinite`. Следовательно, для запуска анимации `fadeIn` для контейнера `div` с классом `announcement` бесконечное количество раз, следует создать такой стиль:

```
.announcement {
  animation-name: fadeIn;
  animation-duration: .25s;
  animation-iteration-count: infinite;
}
```

Но подобное решение, несомненно, утомит ваших пользователей, поэтому лучше от него отказаться. Тем не менее таким стилем можно воспользоваться для «пульсирующего» эффекта, при котором вызывается мягкое свечение кнопки **Подпишитесь на рассылку** (путем анимации ранее рассмотренного свойства `box-shadow`).

Обычно при многократном запуске анимации браузер запускает ее с самого начала. Поэтому, если анимируется превращение желтого фоновой цвета в синий, и это повторяется дважды, браузер покажет, как желтый блок превращается в синий, а затем внезапно появляется желтый блок, который превращается в синий еще раз. Этот эффект может быть слишком резким. В таком случае он будет выглядеть лучше, если через некоторое время после анимации браузер воспроизведет эффект в обратном порядке. Именно так работают переходы: например, когда указатель

мыши находится поверх элемента, браузер анимирует переход из обычного состояния к состоянию, заданному при нахождении поверх элемента. Когда указатель мыши выходит за пределы элемента, браузер воспроизводит анимацию в обратном порядке, возвращая внешний вид элемента в обычное состояние. Чтобы анимация при нечетных воспроизведениях шла в прямом порядке, а при четных — в обратном, используйте свойство `animation-direction` и ключевое слово `alternate`. Например, чтобы элемент постепенно исчезал, а затем снова появлялся, можно создать анимацию `fadeOut`:

```
@keyframes fadeOut {
  from { opacity: 1; }
  to { opacity: 0; }
}
```

А затем воспроизвести ее дважды, меняя при втором воспроизведении направление на противоположное:

```
.fade {
  animation-name: fadeOut;
  animation-duration: 2s;
  animation-iteration-count: 2;
  animation-direction: alternate;
}
```

Этот код запускает анимацию `fadeOut` в отношении любого элемента, имеющего класс `fade`. Анимация должна продолжаться две секунды, а затем повторяться. Поскольку свойство `animation-direction` имеет значение `alternate`, анимация в первый раз приведет к исчезновению элемента (его свойство непрозрачности от полностью непрозрачного (значение `opacity` равно 1) постепенно сменится на 0 — полностью прозрачное, а затем, при втором воспроизведении, все пойдет вспять (значение `opacity` будет изменяться от 0 до 1) и элемент снова появится на экране.

СОВЕТ

Чтобы анимация воспроизводилась несколько раз, но в конечном итоге возвращала элементу его первоначальный вид, используйте четное количество итераций и присвойте свойству `animation-direction` значение `alternate`.

Неважно, сколько раз вы будет воспроизводиться анимация, но как только она завершится, браузер отобразит элемент в его первоначальном виде. Предположим, что изображение анимируется таким образом, чтобы его размер медленно увеличивался до достижения двойного размера. Как только анимация будет завершена, браузер тут же вернет изображению его первоначальный размер, создавая слишком резкий визуальный эффект. В этом случае можно сохранить элементу тот вид, который был у него по завершении анимации. Для этого нужно присвоить свойству `animation-fill-mode` значение `forwards`.

```
animation-fill-mode: forwards;
```

Это свойство применяется к анимируемому элементу вместе с такими свойствами, как `animation-name`, `animation-duration`, и другими, имеющими отношение к анимации.

Сокращенная запись свойства animation

Как видите, существует множество свойств анимации, и набирать их вручную, да еще и со всеми версиями, включающими вендорные префиксы, может быть затруднительно. Поскольку вам по-прежнему нужно пользоваться версиями с вендорными префиксами, путь к упрощению ситуации лежит через использование сокращенной записи свойства `animation`. В одном этом свойстве сочетаются такие свойства, как `animation-name`, `animation-duration`, `animation-timing-function`, `animation-iteration-count`, `animation-direction`, `animation-delay` и `animation-fill-mode`. Можно, к примеру, взять следующий код:

```
.fade {
  animation-name: fadeOut;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: 5s;
  animation-iteration-count: 2;
  animation-direction: alternate;
  animation-fill-mode: forwards;
}
```

И придать ему такой вид:

```
.fade {
  animation: fadeOut 2s ease-in-out 5s 2 alternate forwards;
}
```

Всего одна строка кода вместо семи! Значения свойств нужно перечислить в ранее упомянутом порядке: имя, длительность, тайминг, количество повторов, направление, отсрочка и режим заполнения. Кроме того, важно указать между значениями запятые. Обязательны только имя и длительность. Все остальные значения опциональны.

Если к элементу нужно применить несколько анимаций, следует воспользоваться списком их свойств с использованием запятой в качестве разделителя. Например, чтобы применить две анимации (скажем, `fadeOut` и `glow`) к элементам, выбиравым стилем `.fade`, напишите следующий код:

```
.fade {
  animation: fadeOut 2s ease-in-out 5s 2 alternate forwards,
            glow 5s;
}
```

Разумеется, при реальном использовании этого свойства нужны еще и версии с вендорными префиксами:

```
.fade {
  -webkit-animation: fadeOut 2s ease-in-out 5s 2 alternate forwards,
                    glow 5s;
  animation: fadeOut 2s ease-in-out 5s 2 alternate forwards,
             glow 5s;
}
```

Ваш выбор должен быть за краткой формой записи: она намного лаконичнее и удобнее.

Приостановка анимации

В CSS доступно еще одно свойство анимации — `animation-play-state`. Оно предназначено для управления воспроизведением анимации. Этому свойству передается одно из двух ключевых слов — `running` или `paused`. Чтобы приостановить анимацию, нужно применить к стилю следующее объявление:

```
animation-play-state: paused;
```

Но фактически применить это свойство к CSS можно только с помощью псевдокласса. Как и в случае с переходами, для приостановки анимации нужен инициатор. Один из вариантов предусматривает приостановку анимации при помещении поверх анимируемого элемента указателя мыши. В следующем примере используется стиль для класса `fade`:

```
.fade {  
  animation: fadeOut 2s ease-in-out 2 alternate 5s forwards,  
            glow 5s;  
}
```

Этот код запускает две анимации — `fadeOut` и `glow` — для любого элемента, к которому применен класс `fade`. Предположим, что нужно позволить посетителям приостанавливать эту анимацию при помещении поверх элемента указателя мыши. Для этого нужно только добавить еще один стиль:

```
.fade:hover {  
  animation-play-state: paused;  
}
```

Разумеется, вам понадобится вендорный префикс `-webkit-`, то есть свойство примет вид `-webkit-animation-play-state`.

Более эффективным способом приостановки анимации будет динамическое применение свойства `animation-play-state` к элементу с помощью JavaScript-сценария. Можно будет создать сложную анимацию и добавить кнопку паузы, при нажатии которой анимация будет приостановлена. Дополнительные сведения о JavaScript и анимациях CSS представлены во врезке «Использование JavaScript-сценариев для запуска переходов».

Анимация при наведении указателя мыши

Все ранее показанные анимации запускаются при загрузках страницы. Но у вас есть еще несколько вариантов запуска анимации CSS, включая несколько псевдоклассов и использование JavaScript-сценариев. Чаще всего для анимации применяется псевдокласс `:hover`. С его помощью можно запустить анимацию при установке указателя мыши поверх некоего элемента, например, можно придать логотипу элемент динамичности, скрывая его со страницы и отображая снова. Чтобы применить анимацию к элементу при помещении поверх него указателя мыши, нужно сначала создать анимацию с помощью правила `@keyframes` (шаг 1 в предыдущем разделе). Затем для анимируемого элемента следует создать псевдокласс `:hover`. В стиль для этого псевдокласса добавляются свойства анимации

(шаг 2 в предыдущем разделе). Теперь анимация запускается, только когда посетитель устанавливает указатель мыши поверх элемента.

Практикум

В этом уроке к баннеру будут добавлены преобразования, анимация и переходы.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 10.

1. Откройте в редакторе HTML-кода файл `styles.css`, который находится в папке 10.

Файл `banner.html` включает баннер с изображением логотипа, заголовков и набор навигационных кнопок (рис. 10.9). Файл `styles.css` представляет собой таблицу стилей, форматирующую страницу `banner.html`. Сначала будет добавлено преобразование, увеличивающее масштаб изображения кнопки при помещении поверх нее указателя мыши.

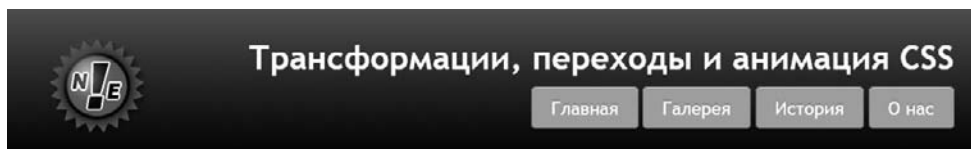


Рис. 10.9. Обычный статический баннер, ожидающий оживления с помощью анимаций, преобразований и переходов

2. В верхней части файла `styles.css` добавьте следующее правило:

```
nav a:hover {
  -webkit-transform: scale(1.5);
  -ms-transform: scale(1.5);
  transform: scale(1.5);
}
```

Кнопки на странице находятся в HTML5-элементе `nav`. Этот селектор потомков нацелен на содержимое элемента `nav` в состоянии помещения поверх него указателя мыши (то есть на момент нахождения указателя мыши посетителя поверх ссылки). Стилль применяет функцию `scale`, которая немного увеличивает кнопку. Стоит учесть, что для того, чтобы код работал в браузерах Safari и Internet Explorer 9, придется использовать вендорные префиксы `-webkit-` или `-ms-`.

3. Сохраните файл `styles.css` и просмотрите страницу `banner.html` в браузере. Установите указатель мыши поверх одной из ссылок-кнопок под заголовком.

Когда указатель будет находиться поверх кнопки, она станет больше, выделяясь на странице (рис. 10.10). Увеличенная кнопка никак не влияет на окружающий ее контент (например, она не расталкивает соседние кнопки). В этом заключается уникальность преобразований CSS. Неплохо, но было бы еще лучше, если бы ко всему этому добавили анимацию.

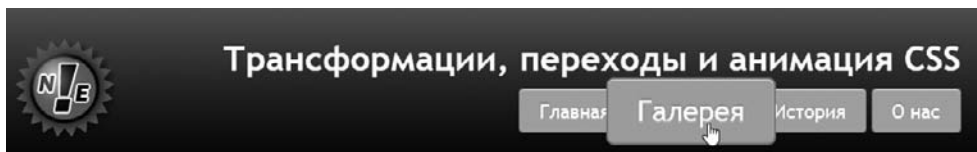


Рис. 10.10. Добавляя кнопке эффект масштабирования при нахождении поверх нее указателя мыши, можно выделить ее на странице

4. Добавьте перед только что созданным стилем `nav a:hover` еще один стиль:

```
nav a {
  transition: all .5s;
}
```

Этот код предписывает браузеру анимировать все изменения свойств элемента `a`, находящегося внутри элемента `nav`, и задает длительность анимации равной половине секунды. Чтобы еще больше оживить ситуацию, добавим к состоянию `hover` фоновый цвет.

5. Найдите стиль `nav a:hover` и добавьте к нему объявление `background-color: red`, чтобы стиль приобрел следующий окончательный вид:

```
nav a:hover {
  background-color: red;
  -webkit-transform: scale(1.5);
  -ms-transform: scale(1.5);
  transform: scale(1.5);
}
```

Если сейчас сохранить страницу и просмотреть ее в браузере, можно увидеть, что кнопки не только становятся крупнее при установке поверх них указателя мыши, но и приобретают красный фон. Но теперь нужно добавить для переходов разные тайминги, чтобы кнопки сначала становились крупнее, а затем приобретали красный фон.

6. Удалите код внутри стиля `nav a` и добавьте два других свойства:

```
nav a {
  -webkit-transition: -webkit-transform .5s,
                    background-color 1s ease-in .5s;
  transition: transform .5s,
              background-color 1s ease-in .5s;
}
```

Вам необходимо добавить два свойства: одно для браузера Safari, а другое — для остальных браузеров. Даже если Safari поддерживает свойство `transition`

(без вендорного префикса), нужно добавить префикс `-webkit-`, поскольку вы используете свойство `-webkit-transform`, которое бы не имело смысла для других браузеров.

Второе свойство — `transition` — предназначено для других браузеров.

Первая часть — `transform .5s` — сообщает браузеру о том, что нужно анимировать любые изменения в свойстве `transform` за полсекунды. Вторая часть — `background-color 1s ease-in .5s` — показывает, что следует анимировать изменение цвета фона за одну секунду, используя для тайминга метод `ease-in`, но до начала перехода взять паузу на полсекунды. Параметр `.5s` в конце играет важную роль, поскольку он соответствует параметру `.5s` в анимации `transform`. То есть браузер будет ждать, пока не завершится переход `transform`, и только потом приступит к изменению цвета фона.

7. Сохраните файл `styles.css` и просмотрите страницу `banner.html` в браузере.

Установите указатель мыши поверх кнопки. Сначала кнопка станет крупнее. Оставьте указатель поверх кнопки, и через некоторое время она станет красной. Можно, конечно, перенастроить порядок перехода, например удалить отсрочку для перехода `background-color`, и изменение цвета начнется одновременно с изменением размера. Но так как изменение цвета длится в течение одной секунды, оно еще будет продолжаться, когда кнопка уже достигнет своего окончательного размера.

Добавление анимации

Теперь настало время испытать в деле анимацию средствами каскадных таблиц стилей. Начнем с вращения и масштабирования изображения логотипа. Первым шагом в создании анимации CSS станет использование правила `@keyframes` для настройки ключевых кадров анимации. Этот пример начинается с анимации, состоящей всего лишь из двух ключевых кадров.

СОВЕТ

Поскольку анимация CSS требует большого объема кода, лучше начать с использования только одного вендорного префикса для наиболее применяемого браузера. Протестируйте страницу в этом браузере и после доведения ее до совершенства добавьте код для других браузеров.

В этом примере используется код, который будет работать в браузерах Chrome, Safari и Opera. Если вы предпочитаете Firefox, Internet Explorer 10 или его более ранние версии, пропустите часть с префиксом `-webkit-`.

1. Откройте файл `styles.css` и добавьте следующий код, указав его после стиля `nav a:hover` в нижней части файла:

```
@-webkit-keyframes logo {
  from {
    -webkit-transform: rotate(0) scale(.5);
  }
  to {
    -webkit-transform: rotate(-720deg) scale(1);
  }
}
```

В примере используется синтаксис `-webkit-`, следовательно, он будет работать в браузерах Chrome, Safari и Opera. Если в качестве основного браузера вы пользуетесь программой Firefox или Internet Explorer 10 (или более поздней версии), замените код `@-webkit-keyframes` фрагментом `@keyframes`.

Код сообщает браузеру, что он должен сначала показать элемент без вращения — `rotate(0)` — и в половинном размере — `scale(.5)`. Затем будут анимированы изменения от состояния `from` до состояния `to` — в данном случае это вращение на -720° , что означает три оборота против часовой стрелки, и возвращение элемента к его нормальному размеру. То есть эта анимация будет вращать элемент и сделает его крупнее.

Теперь нужно применить анимацию к какому-нибудь элементу страницы.

2. Создайте после только что добавленного стиля еще один:

```
.logo {
  -webkit-animation: logo 1s;
}
```

К логотипу баннера применен класс `logo`, следовательно, этот селектор класса применит анимацию к нему. Свойству `animation` можно передать множество значений, но обязательными являются только имя и длительность. Здесь указывается анимация, созданная в последнем действии — `logo`, и браузеру сообщается ее длительность, равная одной секунде.

3. Сохраните файл `styles.css` и просмотрите файл `banner.html` в браузере.

Логотип должен прокрутиться, увеличиться в размерах и остановиться. Если этого не произойдет, перепроверьте код и убедитесь, что просматриваете страницу в соответствующем браузере. Если все идет по плану, эта анимация должна выглядеть весьма привлекательно, но было бы еще лучше, если бы логотип появлялся из-за пределов страницы, а затем останавливался в предназначенной для него позиции баннера. Для этого нужно анимировать позицию элемента на странице.

4. Отредактируйте правило `@keyframe`, придав ему следующий вид:

```
@-webkit-keyframes logo {
  from {
    -webkit-transform: rotate(0) scale(.5);
    left: 120%;
  }
  to {
    -webkit-transform: rotate(-720deg) scale(1);
    left: 0;
  }
}
```

Здесь начальная позиция логотипа находится правее баннера и кнопок навигации у края экрана (вообще-то, левый край логотипа помещается на 1,2-кратную ширину баннера). Чтобы этот код сработал, нужно воспользоваться свойством `position`. Более подробно мы рассмотрим его в главе 15, а сейчас нужно только понять, что каскадные таблицы стилей дают вам возможность позиционировать любой элемент в любом месте окна браузера и даже за его пределами.

Завершающий ключевой кадр помещает логотип в позицию 0, где он обычно должен появляться на странице, то есть в левой части баннера.

5. Сохраните файл `styles.css` и просмотрите страницу `banner.html` в браузере.

Выглядит неплохо. Но можно сделать еще лучше. Добавьте еще один ключевой кадр, чтобы логотип перекачивался на свое место, а затем вращался в обратном направлении по мере приобретения более крупного размера.

6. Измените код, добавленный в шаге 4, придав ему следующий вид:

```
@-webkit-keyframes logo {
  from {
    -webkit-transform: rotate(0) scale(.5);
    left: 120%
  }
  50% {
    -webkit-transform: rotate(-720deg) scale(.5);
    left: 0;
  }
  to {
    -webkit-transform: rotate(0) scale(1);
  }
}
```

Теперь у вас три ключевых кадра: один в начале, второй точно посередине анимации и третий — в ее конце. Браузер будет анимировать свойства по мере их изменения от кадра 1 к кадру 2 и к кадру 3.

Первый ключевой кадр такой же, как и созданный в шаге 4: логотип не прокручен, имеет половинный размер и помещен за пределами правой стороны. Второй ключевой кадр сохраняет логотип в том же размере, три раза его прокручивает и перемещает в левую сторону баннера. И наконец, когда логотип встанет на свое место, браузер увеличит его масштаб, вращая назад с позиции -720° в позицию 0° , то есть логотип теперь будет вращаться три раза по часовой стрелке.

7. Сохраните файл `styles.css` и просмотрите страницу `banner.html` в браузере.

Логотип должен прокатиться с правой стороны баннера в левую, остановиться, увеличиться в размерах и повернуться по часовой стрелке. Если этот код не сработает, перепроверьте его на соответствие коду, добавленному на шаге 6. На данном этапе анимация проигрывается немного быстрее желаемого. Но это легко исправить.

8. Отредактируйте стиль `.logo` так, чтобы анимация продолжалась не одну, а три секунды:

```
.logo {
  -webkit-animation: logo 3s;
}
```

Теперь необходимо продублировать код и удалить префикс `-webkit-`, чтобы код работал в браузерах Firefox и Internet Explorer 10 (и более поздних версиях). Сначала нужно дополнить правило `@keyframes`.

9. Скопируйте правило `@keyframes` и вставьте копию сразу же после оригинала. Удалите префикс `-webkit-`. Ваш код должен выглядеть следующим образом:

```
@-webkit-keyframes logo {
  from {
    -webkit-transform: rotate(0) scale(.5);
    left: 120%
  }
  50% {
    -webkit-transform: rotate(-720deg) scale(.5);
    left: 0;
  }
  to {
    -webkit-transform: rotate(0) scale(1);
  }
}
```

```
@keyframes logo {
  from {
    transform: rotate(0) scale(.5);
    left: 120%
  }
  50% {
    transform: rotate(-720deg) scale(.5);
    left: 0;
  }
  to {
    transform: rotate(0) scale(1);
  }
}
```

Когда-нибудь в прекрасном будущем все браузеры станут использовать единое правило `@keyframes` без всяких префиксов и такое же, не имеющее префиксов, свойство `transform`. Но пока придется набирать весь этот код.

К счастью, самая тяжелая часть — добавление правил `@keyframes`. Добавление анимации к стилю `.logo` потребует меньшего объема работы.

10. Отредактируйте стиль `.logo`, добавив к нему дополнительную строку кода:

```
.logo {
  -webkit-animation: logo 3s;
  animation: logo 3s;
}
```

11. Сохраните файл `styles.css` и просмотрите страницу `banner.html` в браузере Firefox или Internet Explorer (рис. 10.11).

Логотип должен прокатываться по странице и увеличиваться во всех браузерах. Разумеется, в Internet Explorer 9 и более ранних версиях никакой анимации не будет. Но логотип все же будет помещен в нужное место на странице, не нарушая ее внешнего вида. Окончательную версию урока можно найти в папке `10_finished`.

Чтобы продолжить усовершенствование страницы, добавьте анимацию, подсвечивающую одну из кнопок, поверх которой установлен указатель мыши. (Подсказка: создайте анимацию, осуществляющую циклический переход через различные значения тени блока. Для добавления тени, размещающейся внутри блока, воспользуйтесь значением `inset`. Затем добавьте анимацию к стилю `nav a:hover`, чтобы она проигрывалась только при помещении указателя мыши поверх кнопки.)

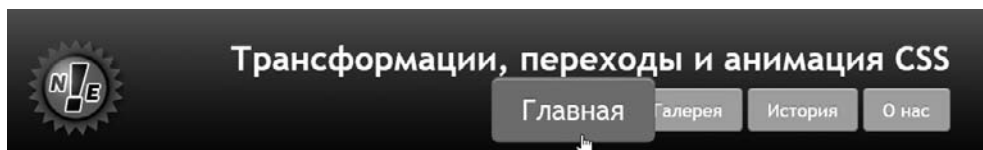


Рис. 10.11. Книги способны на многое, но, к сожалению, они не могут показать ту впечатляющую анимацию, которая только что была создана. Так должна выглядеть страница в ходе воспроизведения анимации и при нахождении указателя мыши над кнопкой

11 Форматирование таблиц и веб-форм

Возможности каскадных таблиц стилей не ограничиваются форматированием текста, изображений и ссылок. Вы можете создавать информационные таблицы с расписаниями, результатами спортивных мероприятий и списки воспроизведения с музыкой, которые становятся намного более читабельными, если к ним добавить границы, фоновые рисунки и другие визуальные улучшения. Ко всему прочему, вы можете применять каскадные таблицы стилей для разработки соответствующих элементов веб-форм, которые помогут вашим посетителям сделать заказ, подписаться на рассылку или воспользоваться вашими самыми новыми веб-приложениями. В этой главе вы узнаете, как вывести на экран таблицы и веб-формы с помощью HTML, а также как их скомпоновать и применить к ним соответствующее форматирование посредством каскадных таблиц стилей. В практикуме в конце главы вы создадите таблицу и веб-форму, используя приемы, которым попутно обучитесь.

Разумное применение таблиц

За короткую историю существования Всемирной паутины HTML-таблицы получили очень широкое распространение. Изначально созданные для отображения данных в табличном виде, они стали очень популярным инструментом для компоновки макетов. Столкнувшись с некоторыми ограничениями в HTML, дизайнеры творчески подошли к решению этой проблемы и начали применять столбцы и строки таблиц для размещения в них таких элементов страниц, как заголовки и боковые панели. Как вы увидите в части III этой книги, каскадные таблицы стилей намного лучше справляются с компоновкой веб-страниц. Вы можете сконцентрировать все ваше внимание на использовании таблиц по их прямому назначению — для отображения данных (рис. 11.1).

HTML (и XHTML) имеет в своем распоряжении внушительное количество элементов, предназначенных для создания таблиц. С помощью этого фрагмента HTML-кода создается очень простая таблица, которую вы можете видеть на рис. 11.2.

```
<table border="1">
<caption align="bottom">
  Table 1: CosmoFarmer.com's Indoor Mower Roundup
</caption>
```

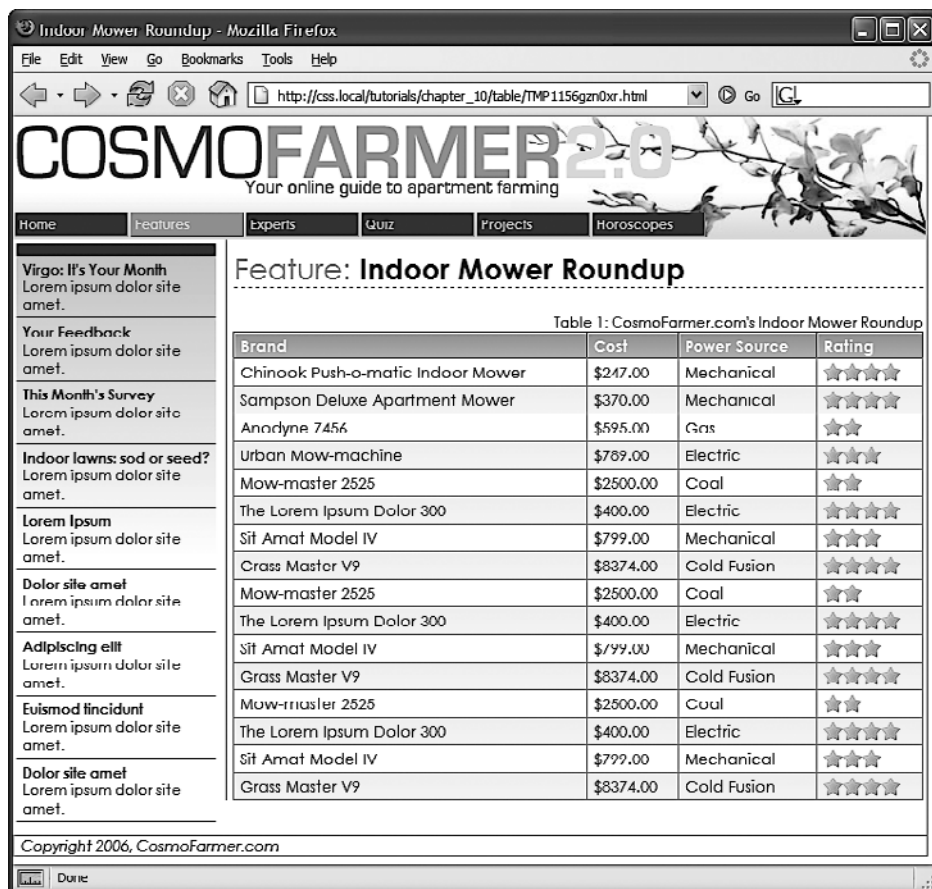


Рис. 11.1. Благодаря CSS в этой таблице, содержащей информацию о газонокосилках, шрифты изменяются на более привлекательные, создаются границы и меняются фоновые цвета, но вся основная структура создана с помощью HTML-кода

```

<colgroup>
  <col class="brand" />
  <col class="price" />
  <col class="power" />
</colgroup>
<thead>
<tr>
  <th scope="col">Brand</th>
  <th scope="col">Price</th>
  <th scope="col">Power Source</th>
</tr>
</thead>
<tbody>
<tr>
  <td>Chinook Push-o-matic Indoor Mower</td>

```



```

    <td>$247.00</td>
    <td>Mechanical</td>
  </tr>
  <tr>
    <td>Sampson Deluxe Apartment Mower</td>
    <td>$370.00</td>
    <td>Mechanical</td>
  </tr>
</tbody>
</table>

```

Даже имея в своем распоряжении лишь по три строки и столбца, таблица использует девять уникальных HTML-элементов: `table`, `caption`, `colgroup`, `col`, `thead`, `tbody`, `tr`, `th` и `td`. Вообще-то, чем меньше HTML-кода, тем лучше, и все эти элементы вам не нужны: вполне можно обойтись только следующими: `table`, `tr` и `td` (и в большинстве случаев еще `th`). Но различные табличные элементы предоставляют множество полезных зацепок для привязки к ним каскадных таблиц стилей. Заголовок для каждого из столбцов таблицы, задаваемый элементом `th`, может иметь различный вид в разных ячейках таблицы, если вы будете использовать при создании стиля элемент `th`, а элемент `colgroup` можно применять в качестве средства управления шириной столбца таблицы. Это избавит вас от нудной работы по созданию множества различных классов, таких как `.tableHeader`, а также от необходимости вручную задавать их для каждой ячейки таблицы. В следующем разделе вы познакомитесь с примерами использования различных элементов и ощутите все их преимущества.

Brand	Price	Power Source	Mini-Review
Chinook Pushomatic Indoor Mower	\$247.00	Mechanical	The latest model of the Chinook mower is a big improvement over last year's model. It's smooth gliding action is perfect for even massively overgrown sod. Its handling around corners is superb -- perfect for those tight areas around sofas and coffee tables.
Sampson Deluxe Apartment Mower	\$370.00	Mechanical	In our battery of 7 mowing tests, the Sampson scored 9 or above on each. The fine blades turn even large weeds into tiny cuttings, perfect for composting or salad garnishes.

Table 1: CosmoFarmer.com's Indoor Mower Roundup

Рис. 11.2. Заголовки таблицы с данными, как эта, обычно создаются с помощью элемента `th`. Заголовки ячеек определяют тип информации, отображаемой в строках и столбцах. Актуальные данные хранятся в таблице в элементах `td`

Форматирование таблиц

Вы можете использовать многие свойства каскадных таблиц стилей, о которых уже прочитали, чтобы добавить привлекательности таблицам и их контенту. Свойство

color, например, устанавливает цвет текста в таблице. Тем не менее вы обнаружите некоторые свойства, особо полезные при использовании их в таблицах, а также свойства, предназначенные исключительно для форматирования таблиц. Тот факт, что таблица состоит из нескольких HTML-элементов, помогает определить, к какому из них применить соответствующее свойство CSS. Отступы элемента table никак не влияют на отображение. В следующих нескольких разделах рассматриваются свойства CSS, используемые для форматирования таблиц, а также HTML-элементы, к которым они применяются.

Добавление отступов

Как вы прочитали в разделе «Управление размерами полей и отступов» главы 7, отступ — это расстояние между границей элемента и его содержимым. Вы можете использовать отступы, чтобы допустить немного свободного места между краями абзацев с текстом и их границами. Когда речь идет о таблицах, границы — это *края* ячейки, и благодаря отступам добавляется свободное пространство вокруг содержимого этой ячейки таблицы (см. рис. 11.2). С их помощью можно управлять расстоянием между контентом конкретной ячейки и ее границами.

Отступы применяются либо к заголовкам таблицы, либо к ее ячейкам, но никак не к самому элементу table. Итак, чтобы задать отступ размером 10 пикселей для всех ячеек таблицы, вам следует воспользоваться следующим стилем:

```
td, th { padding: 10px; }
```

Вы можете также контролировать отступы от каждого из четырех краев ячейки. Чтобы добавить 10 пикселей пространства сверху для каждой ячейки с данными в таблице, 3 пикселя снизу и по 5 пикселей слева и справа, создайте такой стиль:

```
td {  
  padding-top: 10px;  
  padding-right: 5px;  
  padding-bottom: 3px;  
  padding-left: 5px;  
}
```

Или используйте сокращенную запись свойства padding:

```
td {  
  padding: 10px 5px 3px 5px;  
}
```

Настройка горизонтального и вертикального выравнивания

Чтобы настроить расположение содержимого *внутри* самой ячейки, используйте свойства text-align и vertical-align.

Первое свойство применяется для управления горизонтальным выравниванием и может принимать значения left, right, center и justify (рис. 11.3). Это

наследуемое свойство (см. главу 4, чтобы узнать больше о наследовании). Если вы хотите выровнять содержимое всех ячеек таблицы по правому краю, создайте следующий стиль:

```
table { text-align: right; }
```

Это свойство удобно применять к элементам `th`, так как браузеры обычно выравнивают его по центру. Стиль вида `th { text-align: left; }` выравнивает заголовки с ячейками таблицы.

left	center	right	justified
Brand	Price	Power Source	Mini-Review
Chinook Push-o-matic Indoor Mower	\$247.00	Mechanical	The latest model of the Chinook mower is a big improvement over last year's model. It's smooth gliding action is perfect for even massively over grown sod. Its handling around corners is superb -- perfect for those tight areas around sofas and coffee tables.
Sampson Deluxe Apartment Mower	\$370.00	Mechanical	In our battery of 7 mowing tests, the Sampson scored 9 or above on each. The fine blades turn even large weeds into tiny cuttings, perfect for composting or salad garnishes.

Table 1: CosmoFarmer.com's Indoor Mower Roundup

Рис. 11.3. С помощью свойства `text-align` можно указать край ячейки, по которому будет выровнено ее содержимое

У ячеек в таблицах есть также такой параметр, как высота. Обычно браузеры выравнивают содержимое вертикально по центру ячейки (см. значение `middle` на рис. 11.4). Вы можете изменить это с помощью свойства `vertical-align`. Используйте одно из этих четырех значений: `top`, `baseline`, `middle` или `bottom`. Значение `top` помещает содержимое ячейки вверх, `middle` — по центру, а `bottom` — внизу. При указании значения `baseline` выравнивание происходит так же, как и при установке значения `top`, за исключением того, что браузер выравнивает первую строку текста в каждой ячейке заданной строки таблицы (рис. 11.4). Скорее всего, вы даже не заметите тонкостей работы значения `baseline`. В отличие от `text-align` свойство `vertical-align` не наследуется, поэтому вы можете использовать его только в стилях, которые применяются напрямую к элементам `th` и `td`.

СОВЕТ

Итак, изученное форматирование таблиц применяется сразу ко всем вашим таблицам. Если же вы хотите задать для определенной таблицы индивидуальный стиль (или для каждой ее ячейки), измените выбранный селектор. Чтобы использовать особый дизайн определенной таблицы, присвойте ей имя класса — `<table class="stocks">` — и создайте несколько селекторов потомков вида `.stocks td` или `.stocks th`, чтобы применить различное форматирование к отдельным ячейкам. Если вы хотите изменить форматирование определенной ячейки в таблице, примените класс `<td class="subtotal">` к элементу и создайте класс для форматирования этой ячейки.

Brand	Price	Power Source	Mini-Review	
Chinook Push-o-matic Indoor Mower	\$247.00	Mechanical	The latest model of the Chinook mower is a big improvement over last year's model. It's smooth gliding action is perfect for even massively over grown sod. Its handling around corners is superb -- perfect for those tight areas around sofas and coffee tables.	top
Sampson Deluxe Apartment Mower	\$370.00	Mechanical	In our battery of 7 mowing tests, the Sampson scored 9 or above on each. The fine blades turn even large weeds into tiny cuttings, perfect for composting or salad garnishes.	baseline
Sampson Deluxe Apartment Mower	\$370.00	Mechanical	In our battery of 7 mowing tests, the Sampson scored 9 or above on each. The fine blades turn even large weeds into tiny cuttings, perfect for composting or salad garnishes.	middle
Chinook Push-o-matic Indoor Mower	\$247.00	Mechanical	The latest model of the Chinook mower is a big improvement over last year's model. It's smooth gliding action is perfect for even massively over grown sod. Its handling around corners is superb -- perfect for those tight areas around sofas and coffee tables.	bottom

Table 1: CosmoFarmer.com's Indoor Mower Roundup

Рис. 11.4. Если в ячейке используются отступы, ее содержимое на самом деле никогда не выравнивается по верхним или нижним границам: всегда есть промежуток, равный размеру отступа

Создание границ

Свойство `border` (см. раздел «Добавление границ» главы 7) поддерживается таблицами так же хорошо, как и другими элементами, но вам нужно кое-что учитывать. Во-первых, использование границ в стиле, формирующем элемент `table`, выделяет только таблицу, а не какие-либо определенные ячейки. Во-вторых, при применении границ к ячейкам (`td { border: 1px solid black; }`) образуется видимый интервал между ячейками сверху, как это показано на рис. 11.5. Чтобы правильно управлять границами, необходимо понять принцип визуализации ячеек таблицы браузерами и разобраться со свойством `border-collapse`.

- **Управление промежутками между ячейками таблицы.** Если не указано другое, браузеры расставляют в таблице интервалы между ячейками около 2 пикселей. Этот промежуток заметен, когда вы задаете границы для ячеек. В CSS доступно свойство `border-spacing`, которое можно использовать, чтобы управлять размером этого промежутка. Свойство нужно применять к самой таблице, а если требуется удалить пустое пространство, добавляемое браузером между ячейками, присвойте свойству `border-spacing` значение 0:

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

Brand	Price	Power Source	Mini Review
Chinook Push-o-matic Indoor Mower	\$247.00	Mechanical	The latest model of the Chinook mower is a big improvement over last year's model. It's smooth gliding action is perfect for even massively over grown sod. Its handling around corners is superb -- perfect for those tight areas around sofas and coffee tables.

Table 1: CosmoFarmer.com's Indoor Mower Roundup

Brand	Price	Power Source	Mini-Review
Chinook Push-o-matic Indoor Mower	\$247.00	Mechanical	The latest model of the Chinook mower is a big improvement over last year's model. It's smooth gliding action is perfect for even massively over grown sod. Its handling around corners is superb -- perfect for those tight areas around sofas and coffee tables.

Table 1: CosmoFarmer.com's Indoor Mower Roundup

Brand	Price	Power Source	Mini-Review
Chinook Push-o-matic Indoor Mower	\$247.00	Mechanical	The latest model of the Chinook mower is a big improvement over last year's model. It's smooth gliding action is perfect for even massively over grown sod. Its handling around corners is superb -- perfect for those tight areas around sofas and coffee tables.

Table 1: CosmoFarmer.com's Indoor Mower Roundup

Рис. 11.5. Браузеры обычно добавляют промежутки между всеми ячейками в таблице (вероятно, вы не заметите дополнительное пространство, пока не добавите границы, как показано в верхнем изображении)

Но если вы предпочитаете оставлять пустое пространство между ячейками, то его можно добавить:

```
table {
  border-spacing: 2px;
}
```

- **Удаление двойных границ.** Даже если вы уберете промежутки между ячейками, границы, заданные для ячеек, будут удваиваться. Это происходит потому, что нижняя граница одной ячейки добавляется к верхней границе нижней ячейки, и в итоге образуется линия, которая в два раза толще заданной ширины границы (см. рис. 11.5, *посередине*). Самый лучший способ избавиться от этого (а также от промежутков между ячейками) — использовать свойство `border-collapse`. Ему можно присвоить одно из двух значений — `separate` или `collapse`. Значение `separate` эквивалентно тому, как обычно и отображаются таблицы: с промежутками между ячейками и двойными границами. Значение `collapse` схлопывает границы таблицы, позволяя избавиться от интервалов и удвоения границ (см. рис. 11.5, *внизу*). Используйте значение `collapse` следующим образом:

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

ПРИМЕЧАНИЕ

Если присвоить свойству `border-collapse` значение `collapse`, свойство `border-spacing` работать не будет.

- **Скругление углов.** Чтобы добавить к ячейкам таблиц (но не к самим таблицам) скругленные углы, можно воспользоваться свойством `border-radius`. Например, если нужно добавить ячейкам таблицы контуры и скругленные углы, создайте следующий стиль:

```
td {  
  border: 1px solid black;  
  border-radius: 5px;  
}
```

Учтите, что при присвоении свойству `border-collapse` значения `collapse` браузеры проигнорируют установку для ячеек таблицы свойства `border-radius` и визуализируют обычные прямые углы.

ПРИМЕЧАНИЕ

К таблицам и ячейкам можно применять свойство `box-shadow`. Если применить его к таблице, тень появится позади всей таблицы, а если его применить к отдельным ячейкам, то отдельная тень будет у каждой ячейки.

Форматирование строк и столбцов

Чередование затемненных строк в таблице, как показано на рис. 11.6, — это всего лишь один из обычных способов оформления таблицы. Изменив внешний вид каждой строки в таблице, вы позволите посетителям с комфортом просматривать нужные данные. Добавив ранее рассмотренный селектор `nth-of-type`, можно чередовать фон строк:

```
tr:nth-of-type(odd) { background-color: red; }  
tr:nth-of-type(even) { background-color: blue; }
```

Если применять один и тот же фон к чередующимся строкам всех таблиц не нужно, следует просто добавить к целевой таблице имя класса (например, `products`, для таблицы с информацией о товарах), а затем воспользоваться селектором потомка:

```
.products tr:nth-of-type(odd) { background-color: red; }  
.products tr:nth-of-type(even) { background-color: blue; }
```

Вы также не ограничены исключительно цветом. Можно использовать фоновые изображения (см. раздел «Добавление фоновых изображений» главы 8) или даже линейные градиенты (см. подраздел «Линейные градиенты» раздела «Использование градиентных фонов» главы 8) для создания более привлекательного вида, например тонкого затемнения строки таблицы с заголовками, как на рис. 11.6 (вы проработаете похожий в практикуме этой главы). Вы также можете задавать селектор потомков для каждой ячейки в этой строке. Это очень удобно, когда вы

применяете стили ко всем ячейкам в одном столбце, используя для каждой определенный стиль и вид, например `<td class="price">`.

ПРИМЕЧАНИЕ

Селектор `nth-of-type` в Internet Explorer 8 и более ранних версиях браузера не поддерживается.

В процессе форматирования столбцов надо немного схитрить. В языке HTML доступны элементы `colgroup` и `col`, определяющие группу столбцов и отдельный столбец соответственно. Вы можете добавить по одному элементу `col` к каждому столбцу в таблице и далее обозначить их с помощью класса или идентификатора (см. HTML-код в разделе «Разумное применение таблиц» этой главы).

Этими элементами поддерживаются только два набора свойств: `width` и свойства фона (`background-color`, `background-image` и т. д.). Однако и они могут быть очень полезными. Если вы хотите настроить ширину всех строк в столбце, то можете пропустить все атрибуты HTML и форматировать столбцы, применив стиль к элементу `col`. Допустим, используется следующий фрагмент HTML-кода: `<col class="price">`. Вы можете добавить этот стиль в таблицу CSS и установить ширину каждой ячейки в данном столбце равной 200 пикселям:

```
.price { width: 200px; }
```



Рис. 11.6. Благодаря чередующемуся фоновому цвету строк намного проще воспринимать данные

Чтобы выделить столбец, вы можете воспользоваться свойствами форматирования фона. И снова считайте, что у вас есть элемент `col`, к которому применен класс `price`:

```
.price { background-color: #F33; }
```

Имейте в виду, что фон в столбцах помещается позади ячеек, поэтому, если вы установите фоновый цвет или изображение в элементах `td` или `th`, фон столбцов не будет виден.

ПРИМЕЧАНИЕ

Обычно браузеры отображают границы и фоновый цвет каждой пустой ячейки в таблице. Каскадные таблицы стилей позволяют скрыть пустые ячейки. Для этого добавьте свойство `empty-cells:hide` к стилю, примененному к таблице:

```
table {  
  empty-cells: hide;  
}
```

Однако будьте осторожны. Если свойству `border-collapse` присвоить значение `collapse`, браузеры проигнорируют свойство `empty-cells` и отобразят границы и фоновый цвет даже пустых ячеек.

Форматирование веб-форм

Веб-формы — это основной способ коммуникации пользователя с сайтом. Передавая информацию в веб-форму, вы можете подписаться на рассылку, поискать какие-либо товары в базе данных, ввести изменения в профиль социальной сети или заказать конструктор LEGO *Star Wars*, о котором давно мечтали.

Совсем не обязательно, чтобы ваши веб-формы выглядели так же, как и большинство других форм во Всемирной паутине. Применяя каскадные таблицы стилей, вы можете форматировать элементы веб-формы, чтобы они выглядели так же, как и другие элементы сайта: использовали те же шрифты, фоновые изображения и поля. Нет никаких отдельных свойств каскадных таблиц стилей для веб-форм, и вы можете применять практически любые из описанных в этой книге свойств для форматирования элементов веб-форм.

В КУРС ДЕЛА!

Придерживайтесь одного стиля при оформлении веб-форм

Существует несколько причин, по которым надо с осторожностью подходить к изменению привычных и узнаваемых элементов интерфейса, таких как кнопки отправки данных или раскрывающиеся списки. Большинство пользователей уже привыкли к тому, как выглядят и работают элементы веб-форм. В целом внешний вид кнопки отправки данных не меняется от сайта к сайту. Когда люди видят ее, они уже знают, для чего предназначена эта кнопка и когда ее следует нажимать. Если вы *кардинально* измените внешний вид веб-формы, это может привести к тому, что у пользователей возникнут некоторые затруднения при ее заполнении.

Добавив пунктирную границу к какому-либо элементу веб-формы, вы можете добиться того, что пользователь в результате просто не обратит на него внимания и пропустит. А если этот элемент — текстовое поле, предназначенное для ввода адреса электронной почты, на который вы будете отправлять рассылки, можно в итоге потерять несколько пользователей из-за того, что они его попросту пропустили. И напоследок обязательно удостоверьтесь, что пользователи понимают, что перед ними размещена именно веб-форма для ввода данных, а не что-либо иное.

СОВЕТ

Прекрасные примеры дизайнов веб-форм вы найдете по адресу tinyurl.com/pjwtf6y.

Элементы веб-форм

Для создания веб-форм предназначен широкий спектр HTML-элементов. Некоторые из элементов веб-форм форматировать проще (например, текстовые поля), другие — сложнее (переключатели). Рассмотрим несколько простых элементов веб-форм, а также типы свойств, которые они поддерживают.

- **Группирование элементов веб-формы.** Элемент `fieldset` предназначен для группировки элементов, связанных друг с другом. Большинство браузеров нормально отображают фоновые цвета, фоновые изображения и границы этого элемента. При использовании отступов появляются промежутки между краями элементов `fieldset` и их содержимым.
- **Заголовки групп элементов веб-формы.** Элемент `legend` следует за `fieldset` и содержит название (текстовую метку) группы элементов веб-формы. Оно отображается в центре верхней границы элемента `fieldset`. Если элемент `fieldset` представляет, например, адрес доставки, можете добавить следующий код: `<legend>Адрес доставки</legend>`. С помощью каскадных таблиц стилей можно изменить свойства шрифта заголовка группы, добавить фоновые цвета и изображения, а также определить границы.
- **Текстовые поля.** Элементы `<input type="text">`, `<input type="password">` и `<textarea>` создают текстовые поля веб-формы. Эти элементы отлично поддерживаются браузерами. Вы можете изменить размер, семейство шрифтов, цвет и другие свойства шрифта текстовых полей, а также добавить границы, фоновые цвета и изображения. Вы можете задать ширину и высоту этих полей с помощью свойств `width` и `height` соответственно.
- **Кнопки.** Кнопки веб-формы, такие как `<input type="submit">`, позволяют пользователям передавать данные на сервер, сбрасывать содержимое веб-формы или предпринимать какие-либо другие действия. Многие браузеры позволяют экспериментировать с форматированием текста, границ, фона, с тенями и скруглением углов. Вы также можете выравнивать текст кнопки по левому или правому краю либо по центру с помощью свойства `text-align`. Особенно хорошо смотрятся на кнопках линейные градиенты.
- **Раскрывающиеся списки.** Списки, созданные с помощью элемента `select`, также можно форматировать, используя каскадные таблицы стилей. Большинство браузеров позволяют настраивать шрифт, размер шрифта, фоновый цвет/изображение и границы. Но фоновые цвета и изображения не всегда добавляются к раскрывающимся спискам, когда они раскрываются для отображения всех вариантов выбора.
- **Флажки и переключатели.** Большинство браузеров не позволяют применять форматирование к этим элементам, поэтому без лишней необходимости не стоит этого делать.

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Атрибут: основной селектор элементов веб-формы

Когда заходит речь об оформлении веб-форм, селекторы тегов — это не единственный способ. В конце концов, текстовые поля, переключатели, флажки, поля ввода паролей и кнопки используют один и тот же HTML-элемент `input`. Хотя ширина 200 пикселей вполне подходит для текстового поля, скорее всего, вам не захочется, чтобы элемент флажка был таким же большим, поэтому вы не сможете использовать элемент `input` для изменения ширины. Самым лучшим, учитывающим кросс-браузерность способом форматирования исключительно текстовых полей ввода будет создание отдельных классов для каждого из них, например `<input type="text" class="textfield" name="email">`, и стиля для форматирования.

Однако вы можете воспользоваться преимуществами более совершенного селектора атрибутов, позволяющего настроить внешний вид веб-формы, не обращаясь при

этом к классам. Он отбирает HTML-элементы на основе одного из атрибутов. Атрибут `type` определяет, какой тип элемента веб-формы создается с помощью `input`. Так, значение атрибута `type` для текстового поля — `text`. Чтобы создать стиль, который будет изменять фоновый цвет всех однострочных полей ввода на синий, необходимо создать следующий селектор и стиль:

```
input[type="text"] {
    background-color: blue;
}
```

Если вы измените значение `text` на `submit`, то создадите стиль исключительно для кнопок.

Селекторы атрибутов поддерживают все современные браузеры, даже Internet Explorer 7, поэтому применять их для форматирования веб-форм можно без опасений.

Компоновка веб-форм с помощью каскадных таблиц стилей

Все, что надо для создания веб-формы, — добавить несколько текстовых элементов на веб-страницу. Однако зачастую это выглядит хаотично (рис. 11.7, *слева*). Веб-формы обычно выглядят лучше, когда их метки и элементы выровнены по столбцам (см. рис. 11.7, *справа*).

Вы можете добиться этого несколькими путями. Наиболее простой способ — использовать HTML-таблицы. Хотя элементы и метки веб-форм — это, по сути, не табличные данные, они хорошо приспособляются к расположению в формате столбцов/строк. Нужно лишь расположить текстовые метки («Имя», «Номер телефона» и т. д.) в одном столбце, а элементы веб-формы — в другом.

Используя каскадные таблицы стилей, вы также можете создать двухколоночную веб-форму, показанную на рис. 11.7 (преимущество — меньше HTML-кода). Рассмотрим основы этого подхода.

1. Поместите каждую текстовую метку в элемент.

Очевидным выбором для этого будет элемент `label`, так как он создан именно для меток веб-формы. Но вы не можете *всегда* использовать только его. Рядом с переключателями обычно располагаются вопросы типа «Какой ваш любимый цвет?», а для каждого положения переключателя используется метка в элементе `label`. Какой же элемент вы будете использовать для применения к тексту вопроса? На помощь придет элемент `span`: `Какой ваш любимый цвет?`. Затем добавьте класс к каждому из этих элементов: `` и на-

значьте класс только тем элементам `label`, которые следует отобразить в левом столбце (в веб-форме, показанной на рис. 11.7, это были бы метки `First name`, `Last name` и т. д., но не элементы `label` для переключателей).

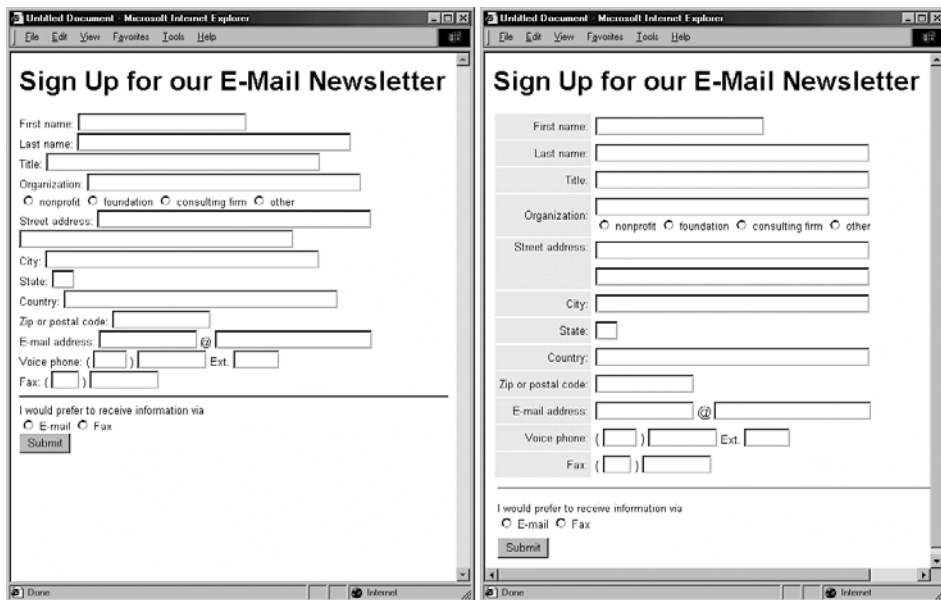


Рис. 11.7. Зачастую элементы веб-формы не очень хорошо компонуются с текстом, что приводит к тому, что они располагаются хаотично и ухудшают читабельность веб-формы (слева). Решением данной проблемы может стать организация элементов веб-формы в столбцы (справа) с использованием HTML-таблиц или CSS

2. Присвойте свойству `display` значение `inline-block` и укажите ширину.

По умолчанию элементы `label` и `span` — строчные, игнорирующие многие настройки, доступные блочным элементам, включая свойства `width`, `height` и `text-align`. Но если превратить метки в блочные элементы (с помощью значения `inline-block`), они по-прежнему будут располагаться рядом с элементами веб-формы (оправдывая свое строчное происхождение). Значение `width` предоставит достаточно пространства, позволяя уместить весь текст метки в одну строку, если это возможно. Можно создать класс, код которого будет иметь следующий вид:

```
.label {
  display: inline-block;
  width: 20em;
}
```

Настройки `width` и `inline-block` превращают метки с классом `label` в небольшие равные по размеру блоки и позволяют выровнять все элементы веб-формы по левому краю.

3. Отформатируйте метки и элементы веб-формы.

Есть еще несколько улучшений, которые вы можете использовать. Вам нужно присвоить значение `top` свойству `vertical-align`, чтобы верхний край текста метки

был выровнен с верхним краем элемента веб-формы. Нужно также выровнять текст метки по правому краю, чтобы каждая метка отображалась рядом с соответствующим элементом веб-формы. И наконец, немного увеличив поле справа, можно создать небольшой промежуток между метками и элементами веб-формы.

```
.label {
  float: left;
  width: 20em;
  vertical-align: top;
  text-align: right;
  margin-right: 15px;
}
```

В итоге получится аккуратная веб-форма. Можно добавить и другие улучшения, выделив, например, метки полужирным шрифтом и изменив их цвет. Во втором практикуме этой главы приведен пример, в котором пошагово расписаны все необходимые для этого действия.

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Псевдоклассы веб-формы

Существует несколько псевдоклассов, предназначенных для форматирования веб-форм. Псевдокласс `:focus` позволяет создать селектор, изменяющий внешний вид текстового поля при щелчке на нем кнопкой мыши или при переходе на него нажатием клавиши Tab (это называется *фокусировкой* на элементе веб-формы). Этим можно воспользоваться для изменения размера, цвета фона, шрифта и других применяемых к полю свойств каскадных таблиц стилей. Практический пример приведен в практикуме.

Псевдокласс `:checked` поддерживается переключателями и флажками. Он предназначен для форматирования этих элементов, но обычно браузеры в отношении оформления этих элементов веб-формы ведут себя «сдержанно» и большинство свойств к ним не применяют. Но, если хотите внести какие-то изменения, можно создать стиль с именем `:checked` и добавить какие-нибудь свойства CSS. Все указанное в этом стиле будет применяться только к переключателям и флажкам веб-формы. Если нужно отформатировать конкретный флажок при его установке, можно создать следующий класс:

```
.special:checked
```

А затем применить его только к данному флажку:

```
<input type="checkbox" class="special"
```

Элементы веб-формы можно делать активными и неактивными. Неактивный элемент веб-формы изменить невозможно: например, нельзя ввести текст в неактивное текстовое поле или установить/снять неактивный флажок. Изменить состояние элемента веб-формы можно только с помощью JavaScript-сценария, поэтому для использования псевдоклассов `:enabled` и `:disabled` нужно изучить JavaScript. Тем не менее эти псевдоклассы могут оказаться весьма полезными.

Предположим, у вас есть веб-форма заказа с адресными полями «выставление счета» и «доставка». Если рядом с адресом доставки указать флажок «адрес доставки идентичен адресу выставления счета», посетитель может установить этот флажок. Используя JavaScript-сценарий, можно сделать неактивным поле адреса доставки, чтобы в него нельзя было случайно что-нибудь ввести. Для наглядности цвет элемента веб-формы можно изменить, к примеру, на серый:

```
:disabled {
  background-color: #333;
}
```

Существуют и другие псевдоклассы, предназначенные для работы с такими специальными возможностями HTML5-форм, как встроенная проверка правильности заполнения веб-формы. Чтобы узнать о них, посетите сайт tinyurl.com/7hg9rpt.

Практикум: форматирование таблиц

Язык HTML отлично подходит для создания таблиц, но вам необходим также и CSS для создания стилей. Как вы узнали из раздела «Разумное применение таблиц» этой главы, для создания HTML-таблицы достаточно довольно простого кода. Загрузив примеры к книге, вы найдете уже готовую HTML-таблицу, на которой можете потренироваться в использовании каскадных таблиц стилей. В этом практикуме вы отформатируете строки, столбцы и ячейки таблицы, замените установленный шрифт более привлекательным, а также зададите фоновый цвет.

Чтобы начать обучение, вы должны иметь в распоряжении файлы примеров. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке **11**.

1. Загрузите браузер и откройте в нем файл `table.html`, расположенный в папке `11\table`.

Эта страница содержит простую HTML-таблицу. У нее есть название, строка заголовков и девять строк ячеек с данными (рис. 11.8). Кроме того, три раза используется элемент `col` для определения трех столбцов с данными. Как вы скоро увидите, элемент `col` весьма полезен при форматировании, поскольку позволяет устанавливать ширину всех ячеек в столбце.

2. Откройте файл `main.css`, расположенный в папке `11\table\css`, в редакторе HTML-кода.

Эта внешняя таблица стилей, связанная с файлом `table.html`. Для начала создадим стиль, который задает ширину таблицы и шрифт текста. К этой таблице применен класс `inventory`, поэтому вы можете использовать селектор класса для форматирования только этой таблицы.

ПРИМЕЧАНИЕ

Браузеры кэшируют файлы, поэтому, если вы внесли изменения в файл `main.css`, но не видите никаких изменений на странице, необходимо обновить кэш браузера. Чтобы узнать, как это сделать, перейдите по ссылке tinyurl.com/brfbqmx.

3. Добавьте следующий стиль в нижней части файла `main.css`:

```
.inventory {
  font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
  width: 100%;
  margin-top: 25px;
}
```

Если вы не установите ширину таблицы, она растянется по размеру контента внутри нее. В данном случае мы установим ширину, равную 100 %, чтобы таблица растянулась по всей ширине ее контейнера (здесь это элемент `body` с установленной шириной и автоматическими значениями полей слева и справа для центровки в окне браузера). При изменении шрифта в элементе `table` используется наследование, чтобы изменился шрифт и во всех остальных элементах

`<table id="inventory">`

Product	Price	Rating
Vitae Quam Lorem	\$19.95	★★★★★
In Tempus Velit	\$14.55	★★★★
Lorem Ipsum Dolor Sat	Priceless	★★★★★
Quis Felis Fringilla	\$29.95	★★
Nunc Sem Pharetra	\$75.99	★★★
Vel Faucibus Elit	\$82.00	★
Non Adipiscing Vitae	\$1.95	★★★★
Aenean Orci Ante	\$17.95	★★★★
Venenatis Non Adipiscing	\$44.00	★★★★★

`<caption>`
`<th>`
(шапка таблицы)
`<td>`
(ячейки таблицы)

`<col id="product">` `<col id="price">` `<col id="rating">`

Table 1: Current Inventory

PRODUCT	PRICE	RATING
Vitae Quam Lorem	\$19.95	★★★★★
In Tempus Velit	\$14.55	★★★★
Lorem Ipsum Dolor Sat	Priceless	★★★★★
Quis Felis Fringilla	\$29.95	★★
Nunc Sem Pharetra	\$75.99	★★★
Vel Faucibus Elit	\$82.00	★
Non Adipiscing Vitae	\$1.95	★★★★
Aenean Orci Ante	\$17.95	★★★★
Venenatis Non Adipiscing	\$44.00	★★★★★

Рис. 11.8. Отформатировав таблицу с помощью границ, фоновых цветов и других свойств каскадных таблиц стилей, можно не только разукрасить скучные HTML-таблицы (*вверху*), но и расположить данные в таблице в более читабельной форме (*внизу*)

внутри страницы — `caption`, `th`, `td` и т. д. И наконец, свойство `margin-top` смещает таблицу вниз от находящегося выше ее заголовка.

Далее создадим стиль для заголовка таблицы.

- Добавьте еще один стиль после только что созданного:

```
.inventory caption {
  text-align: right;
  font-size: .85em;
  margin-bottom: 10px;
}
```

Этот селектор потомков влияет только на элемент `caption`, который появляется внутри другого элемента с классом `inventory` (элемент `table` в нашем примере). Элемент `caption` сообщает о содержимом таблицы. В нашем случае он не должен быть в центре внимания, поэтому мы оставили шрифт небольшого размера и выровняли текст по правому краю. Свойство `margin-bottom` добавляет небольшое пространство между заголовком и таблицей.

Когда вы читаете данные в таблице, очень легко сбиться. Необходим хороший визуальный ориентир. Если мы добавим границы вокруг ячеек, они визуально выделят информацию.

5. Добавьте еще один групповой стиль:

```
.inventory td, .inventory th {
  font-size: 1.1em;
  border: 1px solid #DDB575;
}
```

Групповой селектор предназначен для форматирования элементов заголовков таблицы (th) и ячеек (td), он уменьшает размер шрифта и рисует границу вокруг каждого заголовка и каждой ячейки. Как правило, браузеры добавляют промежутки между всеми ячейками, поэтому вокруг границ и появляются небольшие промежутки (рис. 11.9). Из-за них вся таблица выглядит какой-то разрозненной. Сейчас мы это исправим.

6. Добавьте свойство border-collapse в стиль, который вы создали на шаге 3. Теперь его содержимое будет таким:

```
.inventory {
  font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
  width: 100%;
  margin-top: 25px;
  border-collapse: collapse;
}
```

Свойство border-collapse убирает промежутки между ячейками. Оно также способствует тому, что соприкасающиеся границы ячеек схлопываются, что избавляет таблицу от толстых, некрасивых границ. Если не воспользоваться свойством border-collapse, то нижняя граница заголовка таблицы соединится с верхней границей ячеек таблицы, что приведет к удвоению границы, и ее толщина станет равна 2 пикселям.

Если вы сейчас взглянете на таблицу, то увидите, что данные в ней выглядят намного лучше, но информация в каждой из ячеек немного сжата. Добавим небольшой отступ, чтобы избавиться от этого.

7. Добавьте отступы с помощью группового селектора, созданного в шаге 5:

```
.inventory td, .inventory th {
  font-size: 1.1em;
  border: 1px solid #DDB575;
  padding: 3px 7px 2px 7px;
}
```

Хотя верхняя строка таблицы с заголовками и выделяется из-за использования в ней полужирного шрифта, вы можете сделать ее более наглядной.

8. Создайте новый стиль после стиля .inventory td, .inventory th:. Он будет использоваться только для форматирования заголовка таблицы:

```
.inventory th {
  text-transform: uppercase;
```

```

text-align: left;
padding-top: 5px;
padding-bottom: 4px;
}

```

Этот стиль — отличный пример каскадности. Групповой селектор `td, th` определяет общие свойства форматирования для двух типов ячеек. Указав стиль, предназначенный *только* для элемента `th`, вы в дальнейшем сможете использовать его для изменения внешнего вида заголовков таблицы. Например, свойства `padding-top` и `padding-bottom` здесь заменяют аналогичные настройки, определенные в селекторе в шаге 7. Но, поскольку вы не заместили настройки левого и правого отступа, элементы `th` сохраняют 7 пикселей каждого отступа, определенного в шаге 7. Этот стиль также делает все буквы прописными и выравнивает текст по левому краю ячейки.

Заголовки таблицы по-прежнему недостаточно привлекательны, и, кроме того, складывается впечатление, что таблица размещена на фоне страницы. Чтобы исправить положение, можно добавить фоновое изображение.

Product	Price	Rating
Vitae Quam Lorem	\$19.95	★★★★★
In Tempus Velit	\$14.55	★★★
Lorem Ipsum Dolor Sat	Priceless	★★★★★
Quis Felis Fringilla	\$29.95	★★
Nunc Sem Pharetra	\$75.99	★★★
Vel Faucibus Elit	\$82.00	★
Non Adipiscing Vitae	\$1.95	★★★★★
Aenean Orci Ante	\$17.95	★★★★★
Venenatis Non Adipiscing	\$44.00	★★★★★

Product	Price	Rating
Vitae Quam Lorem	\$19.95	★★★★★
In Tempus Velit	\$14.55	★★★
Lorem Ipsum Dolor Sat	Priceless	★★★★★
Quis Felis Fringilla	\$29.95	★★
Nunc Sem Pharetra	\$75.99	★★★
Vel Faucibus Elit	\$82.00	★
Non Adipiscing Vitae	\$1.95	★★★★★
Aenean Orci Ante	\$17.95	★★★★★
Venenatis Non Adipiscing	\$44.00	★★★★★

Рис. 11.9. Обычно браузеры добавляют промежутки между ячейками таблицы (*вверху*). В позиции их соприкосновения граница удваивается. Свойство `border-collapse` с присвоенным значением `collapse` решает обе проблемы (*внизу*)

9. Измените стиль `th`, добавив линейный градиент и изменив цвет шрифта:

```
.inventory th {
  text-transform: uppercase;
  text-align: left;
  padding-top: 5px;
  padding-bottom: 4px;
  background: rgb(229,76,16);
  background: linear-gradient(to bottom, rgb(229,76,16), rgb(173,54,8));
  color: white;
}
```

Сначала в качестве фона добавляется сплошной RGB-цвет, благодаря этому Internet Explorer 9 и более ранние версии (которые не поддерживают градиенты) отобразят по крайней мере цвет. В следующей строке добавляется линейный градиент с использованием официального синтаксиса. И наконец, цвет текста меняется на белый.

Когда в таблице используется много данных, расположенных во множестве строк и столбцов, иногда бывает трудно определить, к какой строке какие данные относятся. Для повышения читабельности таблицы можно использовать чередующиеся строки, применив селектор `nth-of-type`.

10. Добавьте еще два стиля в файл `main.css`:

```
.inventory tr:nth-of-type(even){
  background-color: rgba(255,255,255,.1);
}
.inventory tr:nth-of-type(odd){
  background-color: rgba(229,76,16,.1);
}
```

Как уже говорилось, селектор `nth-of-type` используется для выбора дочерних элементов, отвечающих той или иной числовой схеме, например каждый пятый абзац. В данном случае первый стиль выбирает каждый четный элемент `tr`, а второй — каждый нечетный элемент `tr`.

И наконец, нужно подогнать ширину тех ячеек, которые находятся ниже в столбцах «Цена» и «Рейтинг». Один из способов заключается в методичном добавлении к этим ячейкам имен классов и создании стиля класса с установленной шириной ячейки. Но лучше будет воспользоваться элементом `col`, что позволит назначить класс или идентификатор всей совокупности ячеек столбца. На рис. 11.9, показано, что эти два столбца имеют идентификаторы `price` и `rating`. С помощью одного группового селектора можно упростить задание ширины для этих двух столбцов.

11. Добавьте еще один стиль в файл `main.css`:

```
#price, #rating {
  width: 15%;
}
```

Сохраните таблицу стилей и откройте файл `table.html` в браузере. Она должна выглядеть так, как показано на рис. 11.9, *внизу*. Готовый пример находится в папке `11_finished\table`.

Практикум: форматирование веб-форм

В этом уроке вы узнаете, как с помощью каскадных таблиц стилей привести веб-форму в порядок, а также сделать ее более привлекательной. Если вы откроете в браузере страницу `form.html` из папки `11\form`, то увидите, что на ней опубликована простая веб-форма для регистрации подписчика на вымышленном сайте (рис. 11.10). Форма задает несколько вопросов, и в ней используются несколько элементов ввода, таких как текстовые поля, переключатели и раскрывающиеся списки. Если рассматривать ее в качестве веб-формы для подписки, выглядит она вполне нормально, но слегка мрачновато. На следующих страницах мы отформатируем текст, выделим вопросы и элементы веб-формы, а также добавим несколько других улучшений.

1. Откройте файл `global.css`, расположенный в папке `11\form`, в редакторе HTML-кода.

Сейчас вы добавите новый стиль к внешней таблице стилей, связанной с файлом `form.html`. Начнем с увеличения размера шрифта, используемого в веб-форме, чтобы сделать ее более читабельной.

2. Добавьте следующий стиль в нижней части таблицы стилей:

```
.subform {  
  font-size: 1.2em;  
  color: white;  
  font-family:Tahoma, Geneva, sans-serif;  
}
```

Этой веб-форме присвоен класс `subForm`, поэтому новый стиль изменяет шрифт, его размер и цвет для всего текста между открывающим и закрывающим тегами элемента `form`.

Пришло время поработать над макетом. Чтобы удачнее скомпоновать элементы веб-формы, воспользуемся двумя столбцами, в одном из которых будут находиться вопросы (текстовые метки), а в другом — ответы (элементы веб-формы).

3. Добавьте следующий код в таблицу стилей:

```
.subform .label {  
  display: inline-block;  
  width: 200px;  
  vertical-align: top;  
}
```

Этот селектор потомков используется для всех элементов класса `.label`, расположенных в веб-форме. Стиль превращает метки из строчных элементов (которые не поддерживают изменение ширины) в блочные. Свойство `width` устанавливает для области меток значение ширины, равное 200 пикселям, а свойство `vertical-align: top` выравнивает текст меток по верхнему краю соответствующих элементов веб-формы. В результате после применения этого стиля к каждому вопросу в веб-форме вы получите выровненный по ширине столбец. Чтобы оценить полученный результат, вам необходимо применить созданный класс к соответствующим элементам веб-формы.

Рис. 11.10. Элемент `table` обычно используется для размещения вопросов в веб-форме. Можно также задействовать CSS, чтобы превратить хаотичный набор текстовых меток и элементов веб-формы (наподобие той, что вы видите на рисунке) в организованную и привлекательную структуру

- Откройте файл `form.html`. Найдите следующий код `<label for="name">` и добавьте класс `class="label"` таким образом:

```
<label for="name" class="label">
```

Вы должны повторить то же самое для всех вопросов на веб-форме.

- Повторите шаг 4 для следующих фрагментов HTML-кода: `<label for="email">`, `<label for="refer">` и `<label for="comments">`.

В веб-форме есть еще один дополнительный вопрос — **Rate your apartment farming skills**. Он не размещен в элементе `label`, так как для ответа на него используется одно из положений переключателя, каждое из которых имеет отдельную метку. Вам нужно добавить элемент `span` к тексту, чтобы можно было воспользоваться стилем `label`.

- Найдите текст **Rate your apartment farming skills** и поместите его в элемент `span`, использующий класс `label`:

```
<span class="label"> Rate your apartment farming skills</span>
```

Теперь вопросы размещены в отдельном столбце (рис. 11.11, *вверху*). Но они бы выглядели лучше, если бы были немного смещены и выделены соответствующим образом.

- В файле `global.css` измените стиль `#subForm .label`, который вы создали в шаге 3, следующим образом:

```
.subform .label {
  display: inline-block;
  width: 200px;
  vertical-align: top;
  text-align: right;
```

Регистрация подписчика

Как вас зовут?

Ваш адрес электронной почты

Навыки квартирного ремонта Новичок Опытный Профи

Откуда вы узнали о нас?

Дополнительные комментарии

Регистрация подписчика

Как вас зовут?

Ваш адрес электронной почты

Навыки квартирного ремонта Новичок Опытный Профи

Откуда вы узнали о нас?

Дополнительные комментарии

Рис. 11.11. Иногда небольшие и едва заметные изменения могут сделать веб-форму более читабельной

```
margin-right: 10px;
font-weight: bold;
color: rgba(255,255,255,.5);
}
```

Просмотрите страницу в браузере. Веб-форма должна выглядеть так, как показано на рис. 11.11, *внизу*.

Веб-форма выглядит уже намного лучше, но кнопка `Subscribe`, размещенная у левого края, смотрится не на своем месте. Выровняем ее, как и остальные элементы веб-формы.

8. Добавьте еще один стиль в файл `global.css`.

```
.subform input[type="submit"] {
  margin-left: 220px;
}
```

Поскольку кнопки отправки данных создаются путем добавления к элементу `input` атрибута `type="submit"`, выбрать их можно с помощью селектора атрибута. Тогда не придется создавать класс и применять его к кнопкам отправки данных.

9. Измените только что созданный стиль кнопки **Subscribe**, добавив дополнительные свойства:

```
.subform input[type="submit"] {
  margin-left: 220px;
  padding: 10px 25px;
  font-size: 1em;
  color: white;
}
```

Свойство `padding` добавляет пространство между текстом и краями кнопки, а свойства `font-size` и `color` форматируют шрифт текста кнопки.

Теперь можно проявить изобретательность и добавить к кнопке градиент.

10. Отредактируйте стиль кнопки **Subscribe** следующим образом:

```
.subform input[type="submit"] {
  margin-left: 220px;
  padding: 10px 25px;
  font-size: 1em;
  color: white;
  background: rgb(0,102,153);
  background: linear-gradient(to bottom, rgba(255,255,255,.1) 40%,
  rgba(255,255,255,.5));
}
```

С помощью данного кода установлен стандартный цвет фона (для Internet Explorer 9 и более ранних версий), а затем — линейный градиент. В нем заданы два цветовых узла. Первый цвет простирается от верхнего края до 40 % внутрь кнопки, затем начинается плавный переход цвета (градиенты и цветовые узлы рассматриваются в разделе «Использование градиентных фонов» главы 8).

И наконец, еще немного скорректируйте внешний вид. Удалите стандартную рамку, скруглите углы и задайте эффект свечения вокруг блока.

11. Отредактируйте стиль кнопки **Subscribe** в последний раз (изменения выделены полужирным шрифтом):

```
.subform input[type="submit"] {
  margin-left: 220px;
  padding: 10px 25px;
  font-size: 1em;
  color: white;
  background: rgb(0,102,153);
  background: linear-gradient(to bottom, rgba(255,255,255,.1) 40%,
  rgba(255,255,255,.5));
}
```

```
border-radius: 5px;
box-shadow: 0 0 4px white;
}
```

Присвоение свойству `border` значения `none` удаляет рамку, которую браузер обычно визуализирует вокруг кнопки, а свойство `border-radius` скругляет углы кнопки. И наконец, добавление тени без горизонтального или вертикального смещения (имеется в виду часть `0 0`) позволяет получить эффект свечения элемента, которое выглядит как легкая белая подсветка, исходящая из-под кнопки.

ПРИМЕЧАНИЕ

Усовершенствование дизайна кнопки можно продолжить. Создайте эффект ролловера — `.subform input[type="submit"]:hover` — и измените фоновый цвет. Используя материал предыдущей главы, вы можете этот переход даже анимировать!

Теперь текстовые метки и кнопка **Subscribe** смотрятся великолепно, но зачем на этом останавливаться? Настало время приукрасить элементы веб-формы. Начнем с изменения их шрифта и фонового цвета.

- Отформатируйте раскрывающийся список:

```
.subform select {
  font-size: 1.2em;
}
```

Этот стиль немного укрупняет текст. Можно выбрать шрифт, добавить цвет фона и внести другие изменения. Но некоторые браузеры (например, Safari) не позволяют вносить существенные изменения в форматирование раскрывающихся списков, поэтому все внесенные в них изменения стилей нужно тестировать.

Теперь пришла пора внести изменения в текстовые поля.

- Создайте новый групповой селектор для трех текстовых полей:

```
.subform input[type="text"], .subform textarea {
  border-radius: 5px;
  border: none;
  background-color: rgba(255,255,255,.5);
  color: rgba(255,255,255,1);
  font-size: 1.2em;
  box-shadow: inset 0 0 10px rgba(255,255,255,.75);
}
```

Этот групповой стиль выбирает все элементы `input`, имеющие тип `text`, а также многострочные текстовые области (элемент `textarea`). В нем применяются различные свойства, с которыми вы уже должны быть знакомы, такие как `border-radius`, `background-color`, `font-size` и `box-shadow`. Текстовые поля выглядят мелко, поэтому для них устанавливается ширина и к ним добавляются небольшие отступы.

- Отредактируйте только что созданный стиль, изменив ширину и применив отступы (изменения выделены полужирным шрифтом):

```
.subform input[type="text"], .subform textarea {
  border-radius: 5px;
  border: none;
  background-color: rgba(255,255,255,.5);
  color: rgba(255,255,255,1);
  font-size: 1.2em;
  box-shadow: inset 0 0 10px rgba(255,255,255,.75);
  width: 500px;
  padding: 5px;
}
```

Вы можете сделать форму более удобной для пользователей, выделив активные элементы с помощью специального псевдокласса `:focus` (см. раздел «Выборка форматлируемых ссылок» главы 9). Мы добавим его на следующем шаге.

15. В конце внутренней таблицы стилей добавьте стиль для раскрывающегося списка и трех текстовых полей:

```
.subform input[type="text"]:focus, .subform textarea:focus {
  background-color: white;
  color: black;
}
```

Просмотрите страницу в браузере. Сейчас она должна выглядеть так, как показано на рис. 11.12. Готовую версию созданной в этом уроке веб-формы вы можете найти в папке `11_finished\form`.

Рис. 11.12. Используя псевдокласс `:focus`, вы можете сделать веб-форму более удобной для пользователя: будет подсвечиваться активный элемент веб-формы. Здесь можно увидеть, что все готово к выбору пункта из раскрывающегося списка, поскольку он имеет белый цвет фона

ЧАСТЬ III

Верстка страниц с помощью CSS

Глава 12. Введение в CSS-верстку

Глава 13. Макеты на основе обтекаемых элементов

Глава 14. Позиционирование элементов на странице

Глава 15. Адаптивный веб-дизайн

Глава 16. Система модульной верстки Skeleton

Глава 17. Профессиональная flexbox-верстка

12 Введение в CSS-верстку

Каскадные таблицы стилей удобны при форматировании текста, навигационных панелей, изображений и других элементов веб-страницы, но их по-настоящему потрясающие способности проявляются, если нужно скомпоновать макет всей веб-страницы. В то время как HTML-разметка обычно отображает контент страницы на экране сверху вниз, размещая блочные элементы друг за другом, каскадные таблицы стилей позволяют создавать расположенные бок о бок колонки и помещать изображения или текст в любой позиции на странице (и даже наслаивать поверх других элементов), поэтому веб-страницы имеют намного более интересный внешний вид.

CSS-верстка — это достаточно обширная тема. В этой части книги вы подробно узнаете о двух наиболее важных техниках CSS-верстки. А в этой главе приведен краткий обзор принципов, на которых построена разметка, и немного полезных инструкций для решения возникающих в процессе работы проблем.

Типы макетов веб-страниц

Быть веб-дизайнером означает иметь дело с неизвестным. Какими браузерами пользуются ваши посетители? На каких устройствах они просматривают ваш сайт: на смартфоне под управлением операционной системы Android или на iPad? Самая большая проблема, с которой сталкиваются разработчики, состоит в создании привлекательных дизайнов, учитывающих различные размеры экрана. Мониторы различаются размерами и разрешением: от маленьких 15-дюймовых с разрешением 640×480 пикселей до огромных 30-дюймовых экранов с разрешением примерно 4096×2160 пикселей. Не говоря уже о небольших экранах мобильных телефонов и планшетных компьютеров.

Верстка веб-страниц предполагает три основных подхода к решению упомянутой проблемы. Дизайн практически каждой веб-страницы сводится к использованию одного из двух вариантов: *фиксированного* либо *резинового* дизайна. Фиксированные макеты дают вам наибольший контроль над компоновкой элементов страницы, но могут доставить неудобства отдельным посетителям, использующим устройства, разрешение экранов которых не соответствует предусмотренному вами. Пользователям с устройствами с действительно маленькими экранами придется прокручивать страницу вправо, чтобы все увидеть, а те, у кого устрой-

ства оборудованы экранами с большой диагональю, будут видеть пустоту в области экрана, где мог бы отобразиться контент вашей страницы. Кроме того, владельцам смартфонов для получения нужного им контента придется прибегать к сужению и расширению выводимой на экран информации. Резиновые дизайны (страница увеличивается или уменьшается, чтобы соответствовать размеру окна браузера) делает управление дизайном страницы серьезным испытанием, но при этом эффективнее используется окно браузера. Один из мощных инструментов в подходе к построению веб-страниц — *адаптивный веб-дизайн* — старается решить проблему сильно различающихся по размеру экранов, но за счет усложнения процесса верстки.

- **Фиксированный дизайн.** Некоторые дизайнеры предпочитают плотно распределять страницу по ширине, как показано на рис. 12.1, *вверху*. Независимо от ширины окна браузера ширина контента страницы не изменяется. В некоторых случаях страница «цепляется» к левому краю окна браузера или, что бывает чаще, центрируется. С фиксированным дизайном вам не нужно волноваться по поводу того, что случится с вашей страницей на очень большом (или маленьком) экране.

Многие сайты с фиксированным дизайном поддерживают ширину экрана менее 1000 пикселей, позволяя окну, полосам прокрутки и другим элементам браузера подходить по размерам для экранов с разрешением 1024 × 768. Очень распространенной является ширина 960 пикселей. Это значение используют *большинство* сайтов с фиксированным дизайном, но в последние годы положение дел существенно изменилось благодаря широкому внедрению смартфонов и планшетных компьютеров, экран которых зачастую имеет меньшую ширину (разрешение), чем среднестатистический сайт с фиксированным дизайном. Ввиду вышесказанного сайтов с фиксированным дизайном становится все меньше. Им на смену приходит адаптивный дизайн, о котором мы и поговорим.

ПРИМЕЧАНИЕ

Чтобы увидеть пример сайта с фиксированным дизайном, посетите ресурс nytimes.com (издание New York Times предоставляет различные версии сайта для мобильных устройств, поэтому для этого примера используйте версию, предназначенную для компьютера).

- **Резиновый дизайн.** Иногда легче плыть по течению вместо того, чтобы бороться с ним. Сайт с резиновым дизайном адаптируется так, чтобы соответствовать любой ширине окна браузера, путем задания процентного значения ширины вместо абсолютного в пикселях. Страница становится шире либо уже, если посетитель изменяет размеры окна браузера (см. рис. 12.1, *посередине*). Хотя этот тип дизайна наилучшим образом использует доступное пространство окна браузера, вам придется приложить больше усилий, чтобы убедиться в том, что страница хорошо выглядит при различных размерах окна браузера. На очень больших мониторах такой тип дизайна может смотреться слишком размашисто, с длинными строками текста, неудобными для чтения. Существует несколько способов реализации резинового дизайна, например *обтекаемые* (глава 13) и *flex-элементы* (глава 14).



Рис. 12.1. Справиться с широким спектром размеров окон браузеров и экранных шрифтов можно несколькими способами

ПРИМЕЧАНИЕ

Чтобы увидеть пример сайта с резиновым дизайном, зайдите на сайт maps.google.com.

- **Адаптивный дизайн.** Эта технология, представленная веб-дизайнером Этаном Маркоттом, предлагает другой способ решения проблемы, возникшей из-за большого разнообразия размеров экранов смартфонов, планшетов и компьютеров. Вместо представления единого макета для всех устройств адаптивный веб-дизайн проводит коррекцию ширины для различных окон браузеров путем изменения их представлений. Например, адаптивная веб-страница может ужиматься с широкого, многоколоночного макета для компьютерных мониторов до одноколо-

ночного макета для смартфонов. Таким образом, адаптивный веб-дизайн сильно напоминает резиновые макеты — конструкции, использующие процентные отношения с целью расширения или сужения в ответ на задаваемую ширину окна браузера. Но в новом дизайне технология пошла дальше путем использования более сложного CSS-кода, так называемых *медиазапросов* для передачи различных дизайнов для браузеров устройств с экранами разной диагонали. Это позволяет создавать существенно отличающиеся макеты в зависимости от устройств, на которых просматривается страница.

В практикуме в конце следующей главы вы создадите страницы с фиксированным и резиновым дизайном. Техника адаптивного веб-дизайна будет подробно рассмотрена в главе 13.

ПРИМЕЧАНИЕ

Свойства `max-width` и `min-width` предлагают компромисс между фиксированным и резиновым дизайном (см. раздел «Изменение высоты и ширины» главы 7).

Принцип CSS-верстки

Как говорилось в главе 1, на заре развития Всемирной паутины ограничения HTML-кода вынудили дизайнеров придумывать новые способы оформления сайтов. Самым распространенным инструментом был элемент `table`, который, как изначально предполагалось, должен был служить для отображения информации в виде таблицы, составленной из строк и столбцов с данными. Разработчики использовали HTML-таблицы, чтобы создать своего рода основу для организации контента страницы (рис. 12.2). Однако, поскольку элемент `table` не был предназначен для разметки, дизайнерам часто приходилось управлять им непривычными способами (например, помещая таблицу внутри ячейки *другой* таблицы), чтобы получить нужный результат. Этот метод занимал много времени, добавлял лишний HTML-код и усложнял изменение дизайна в дальнейшем. Но это все, что было у дизайнеров до каскадных таблиц стилей.

В заслугу HTML-таблицам можно поставить следующее: они предоставляют хорошо организованную структуру с унифицированными отдельными ячейками. В CSS-дизайне имеются отдельные элементы с контентом, который нужно разместить в одной из областей страницы. Путем группировки контента во многих отдельных контейнерах и позиционирования этих контейнеров можно создать сложные дизайнерские решения, состоящие из колонок и строк. Самый распространенный элемент, который используется для достижения данной цели, — элемент `div`.

Элемент `div`

Компоновка макетов приводит к расположению контента в различных областях страницы. В каскадных таблицах стилей для организации контента обычно применяется элемент `div`. Как вы узнали из раздела «Верстка HTML-кода вместе с CSS» главы 1, `div` — это HTML-элемент, у которого нет собственных свойств форматирования (помимо того факта, что браузеры рассматривают элемент как блок с разрывом

строки до и после него). Вместо этого он используется для разметки логического группирования элементов, или *создания разделов* на странице.

В элемент `div` обычно заключается фрагмент HTML-кода, в котором все элементы объединены общим смыслом. Такой элемент, как панель навигации (см. рис. 12.2), обычно занимает верхнюю часть страницы, так что есть смысл задавать элемент `div`, оборачивающий и ее. Кроме того, придется определить элемент `div` для всех основных областей вашей страницы, таких как баннер, область основного контента, боковая панель, нижний колонтитул и т. д. Есть возможность обернуть элементом `div` один или более дополнительных разделов. Одна из распространенных методик состоит в оборачивании в контейнер `div` HTML-кода элемента `body`. Тогда вы сможете установить некоторые основные свойства страницы, применяя каскадные таблицы стилей к этой *обертке* (то есть к элементу `div`). Появится возможность установить ширину для всего контента страницы, задать поля слева/справа или расположить весь контент по центру экрана и добавить фоновый цвет или изображение к основной колонке контента.

Как только будут определены все `div`-контейнеры, добавьте к каждому из них либо класс, либо идентификатор, чтобы было удобнее форматировать каждый контейнер по отдельности. Элемент `div` для области баннера страницы может иметь следующий вид: `<div class="banner">`. Можно обойтись идентификаторами, но один и тот же идентификатор используется на странице однократно, поэтому при наличии элемента, появляющегося несколько раз, следует применять класс. Если, к примеру, существует несколько `div`-контейнеров, оборачивающих фотографии и подписи под ними, можно заключить такие элементы в `div`-контейнер и добавить следующий класс: `<div class="photo">`. (Чтобы разобраться, в каких случаях следует применять `div`-контейнер, см. врезку «HTML-элементы `div` и `span`» в главе 3.) Кроме того, если дело касается каскадности (см. главу 5), идентификаторы проявляют всю свою мощь и легко замещают другие стили, зачастую заставляя вас создавать длинные селекторы вроде `#home #banner #nav` для того, чтобы заместить ранее созданные селекторы с использованием идентификаторов. Поэтому многие веб-дизайнеры стараются не добавлять идентификаторы, отдавая предпочтение классам.

Как только будут размечены контейнеры `div`, добавьте к каждому из них либо класс, либо идентификатор, и тогда вы сможете создавать стили CSS для позиционирования блоков на странице, используя обтекаемые элементы (см. главу 13).

ВАЖНОЕ ЗАМЕЧАНИЕ

Находим разумный баланс

Хоть контейнеры `div` крайне необходимы при CSS-верстке, не бросайтесь активно применять их на вашей странице. Распространенное заблуждение состоит в том, что вы должны оборачивать в контейнеры `div` все элементы на странице. Допустим, что основная навигационная панель на вашем сайте — неупорядоченный список ссылок (как, например, та, что описана в разделе «Создание панелей навигации» главы 9). Поскольку это важный элемент, вы,

возможно, пожелаете обернуть его в контейнер `div`:

```
<div id="mainNav"><ul>...</ul></div>
```

Однако нет никаких причин добавлять контейнер `div`, если элемент `ul` настолько же удобен. Если элемент `ul` содержит ссылки основной панели навигации, вы можете добавить ваш класс к этому элементу: `<ul class="mainNav">`.

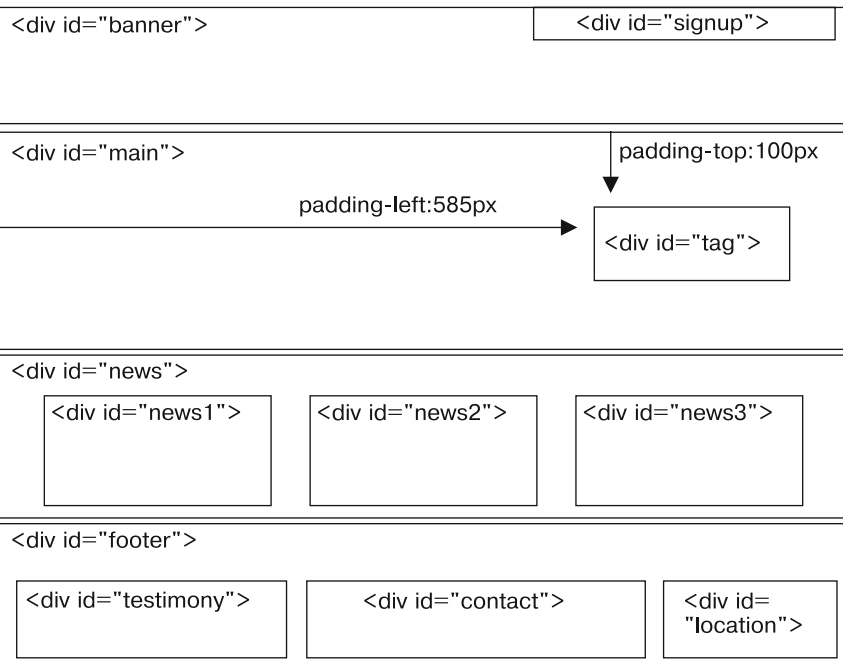
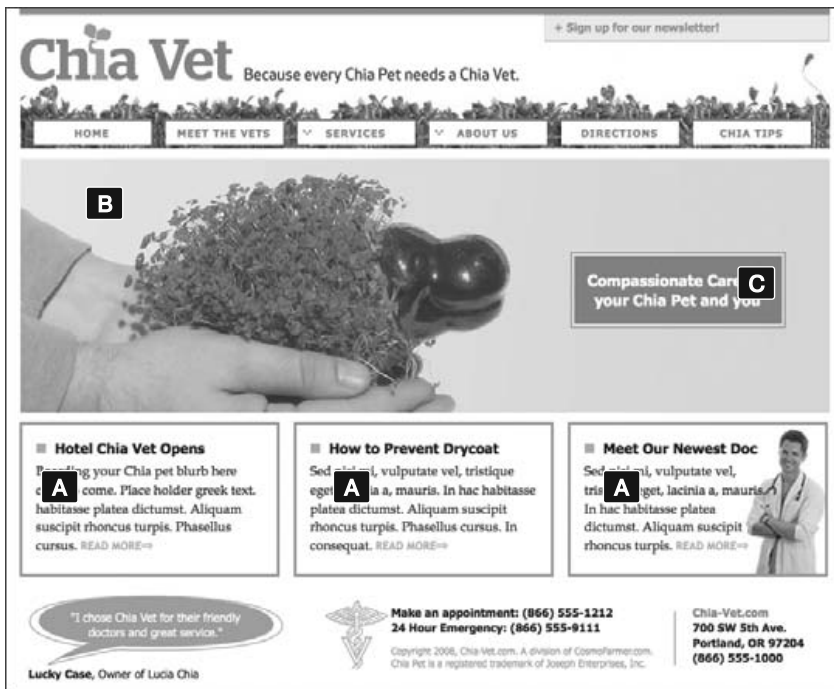


Рис. 12.2. Корректное расположение HTML-элементов, необходимое для преобразования макета Photoshop в код HTML и CSS, включает в себя просмотр структуры страницы и оборачивание связанных групп HTML-элементов в контейнеры div

ВАЖНОЕ ЗАМЕЧАНИЕ

Кроме того, нет необходимости использовать контейнер `div`, если под рукой есть другой, более подходящий HTML-элемент. Например, вы захотели процитировать кусок текста — создать блок, выровненный по правому краю, содержащий важную цитату из контента страницы. В этом случае вместо `div` вы можете использовать элемент `blockquote`, позиционируя его с помощью свойства `float`, которое рассматривалось в разделе «Управление обтеканием контента с помощью плавающих элементов» главы 7.

Но это не значит, что нужно избегать использования элементов `div`. Добавляя их в разумных пределах, вы не увеличите размер файла и не замедлите загрузку страницы. Если контейнер `div` поможет вам

решить проблему и ни один из других элементов не подходит, естественно, используйте `div`. Кроме того, контейнер `div` — это единственный способ сгруппировать в единое целое несколько *различных* HTML-элементов. На самом деле не так редко можно увидеть контейнер `div`, обернутый несколькими другими.

Основное правило при работе с HTML — стараться свести количество кода к минимуму, но при этом применять его столько, сколько нужно. Если добавление пары лишних элементов `div` имеет смысл для реализации дизайна, то смело используйте их. Или же попробуйте воспользоваться семантическими элементами, рассматриваемыми в следующем разделе.

Семантические элементы HTML5

До появления HTML5 для организации контента дизайнеры использовали в основном элемент `div`. Но последние версии языка HTML включают другие HTML-элементы, предназначенные для группировки контента конкретного типа. Например, элемент `article` предназначен для содержимого, составляющего одну самостоятельную композицию, например пост блога. Элемент `header` предназначен для представления шапки сайта, навигационных анонсирующих элементов и другого вводного материала страницы или ее раздела. Есть также элемент `footer`, предназначенный для оформления завершающей информации, такой как сведения об авторских правах, имя создателя сайта, дата публикации страницы и т. д. В общем, в HTML5 предоставляются элементы, которые берут на себя работу контейнера `div` для определенных типов контента. (Краткое введение в семантические элементы HTML5 можно найти по адресу tinyurl.com/nsk9768.)

Указанные HTML5-элементы предназначены для *семантического* деления контента веб-страницы. То есть имя элемента, к примеру `header` (в пер. с англ. «шапка» или «верхний колонтитул»), информирует о содержимом этого элемента. Элемент `figure` предназначен для содержания рисунка, как понятно из имени, если перевести его с английского. Иначе говоря, если нужно поместить внутрь элемента содержимое конкретного типа, посмотрите, какой из элементов отвечает этим требованиям. Если содержимое является нижним колонтитулом вашей страницы, то вместо старого доброго контейнера `div` следует применять элемент `footer`. Элемент `div` по-прежнему пригоден для использования. Но теперь он в основном задействуется для группировки контента по соображениям форматирования, например, если нужно добавить границу по краям группы элементов или выровнять эти элементы по левому краю страницы, используя свойство `float`. (Как все это делается, будет рассмотрено в главе 13.)

Все это говорит о том, что, кроме демонстрации превосходства над конкурентами на очередной встрече веб-разработчиков, использование этих семантических

элементов в данный момент не дает никаких особых преимуществ. В будущем программные средства смогут просматривать HTML-код и выделять отдельные публикации на основе элемента `article` или использовать семантические элементы каким-либо другим образом, чтобы улучшить анализ компонентов вашей веб-страницы. Но если вы оттачивали свое мастерство на применении `div`-контейнеров, то пока можете по-прежнему использовать только элементы `div`.

Технологии CSS-верстки

В большинстве веб-страниц для верстки используется свойство `float` (другие варианты CSS-верстки рассмотрены в следующей врезке с информацией для опытных пользователей). Вы уже сталкивались с этим, казалось бы, простым свойством в главе 8, где оно было представлено как способ позиционирования изображения внутри колонки с текстом выравниванием его либо по левому, либо по правому краю. Тот же принцип применяется и к другим элементам: устанавливая для них ширину и выравнивая по левому или по правому краю, вы можете создать колонку (текст, следующий за элементом, обтекает выровненный элемент, как будто тот является еще одной колонкой). Используя свойство `float` с множеством контейнеров `div` или других элементов, вы получаете возможность создавать многоколоночные дизайны. Применяя эту методику дальше, вы будете быстро создавать сложные дизайны, помещая одни обтекаемые элементы `div` внутри других.

Еще одна CSS-технология, *абсолютное позиционирование*, позволяет поместить элемент в любой позиции страницы с точностью до одного пиксела. Вы можете поместить элемент, например, в 100 пикселах от верхнего края окна браузера и в 15 пикселах от левого края. Такие программы предпечатной подготовки, как Adobe InDesign или Apple Pages, работают именно так. К сожалению, изменчивая природа веб-страниц, а также некоторые странные свойства абсолютного позиционирования затрудняют возможность полного контроля над компонентами макета при использовании этой технологии. Как вы узнаете в главе 14, сверстать страницу с помощью абсолютного позиционирования можно, но этот способ лучше подходит для небольших задач вроде позиционирования логотипа в конкретной позиции на странице.

Если пока все эти понятия воспринимаются вами слишком абстрактно, переживать не стоит — работа с использованием всех этих технологий будет рассмотрена в следующих трех главах.

Стратегии верстки

Верстка веб-страницы с помощью каскадных таблиц стилей — это скорее искусство, чем наука. Поэтому нет четкой формулы, которой нужно придерживаться для разметки контента с помощью HTML и форматирования посредством CSS. То, что работает с одним дизайном, может не подойти для другого.

Изучение CSS-верстки происходит на личном опыте. Нужно узнавать, как работают различные свойства каскадных таблиц стилей (особенно абсолютное

позиционирование и обтекаемые элементы), читать о методах верстки, следовать примерам из практикумов, таким как представленные в следующих двух главах, и много-много практиковаться.

Тем не менее определенно есть некоторые стратегии, которые можно принять, приступая к верстке. Они больше похожи на установки или инструкции, а не на жесткие правила. Но, как только вы начнете проектировать дизайн для каких-либо проектов, имейте в виду эти инструкции.

Начиная с контента

Многие дизайнеры хотели бы сразу перейти непосредственно к своей теме — цветам, шрифтам, значкам и изображениям. Но, начиная с визуального дизайна, вы ставите телегу впереди лошади.

Самые важные элементы веб-страницы — это ее контент (заголовки, абзацы текста, фотографии, навигационные ссылки, видео, а также все то, ради чего люди посетят ваш сайт). Посетители захотят выяснить для себя, что ваш сайт может предложить им. Контент стоит во главе угла, и вам сначала нужно подумать, что вы хотите сказать, прежде чем решить, как это должно выглядеть. В конце концов, создавая фантастическую по красоте трехмерную боковую панель, вы ничего не добьетесь, если вам особо нечего отобразить на ней.

Кроме того, идея страницы должна диктовать ее дизайн. Если вы решите, что на домашней странице нужно продавать услуги вашей компании, и захотите выделить выгодное предложение для клиентов, то вполне вероятно, что большое значение будет иметь крупная фотография с приветливыми сотрудниками, как и цитата отзыва одного из довольных клиентов. Поскольку оба этих элемента важны для страницы, вы должны сделать их заметными и неотразимыми.

Стратегия Mobile First

В связи с ростом количества смартфонов и планшетных компьютеров разработчикам пришлось призадуматься насчет сокращения контента до ключевых сообщений и фактов. Это движение под названием *Mobile First* связано с малым размером экрана смартфонов, а также с ограниченным вниманием людей, находящихся в пути. Конструкции Mobile First касаются приоритетного внимания к контенту, а также избавления от его излишнего захламления, включая дополнительную информацию, которая прекрасно помещается на больших экранах компьютерных мониторов, но мешает на экранах значительно меньшего размера и отвлекает от основной информации, которую вы надеялись донести до посетителя.

Учтите, если ваш сайт будет посещать существенное количество людей, использующих смартфоны или планшетные компьютеры с небольшими экранами, не каждый из них захочет прокручивать длинную страницу с контентом (или масштабировать ее, чтобы увидеть все, что доступно на странице). Вместо попытки заполнить каждый оставшийся квадратный сантиметр на 32-дюймовом компьютерном мониторе подумайте об упрощении своего контента, чтобы его усвоение давалось посетителям легко и непосредственно.

ПРИМЕЧАНИЕ

Понятие Mobile First было придумано Люком Врублевски, который написал фантастически короткий трактат на эту тему, доступный по адресу tinyurl.com/kq92pws. Еще одна короткая статья, посвященная стратегии Mobile First, доступна по адресу tinyurl.com/nfew8nz.

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ**Дополнительные способы верстки с помощью CSS**

Сначала для создания многоколоночных макетов веб-дизайнеры использовали таблицы. Затем появилось свойство `float`, благодаря чему был предоставлен другой, менее объемный по коду метод верстки веб-страницы. И хотя большинство разработчиков все еще используют свойство `float`, существуют и другие свойства: часть из них можно применять уже сегодня, а некоторые станут доступны в будущем.

Рабочая группа Консорциума W3C настойчиво ведет к финальным версиям некоторые другие технологии CSS-верстки. Например, модуль *многоколоночной* верстки предоставляет способ получения длинной колонки текста и ее отображения в нескольких смежных колонках. Разговор о нем пойдет в практикуме следующей главы.

Модуль *flex-верстки* предоставляет еще один способ выстраивания блоков содержимого — вертикально,

горизонтально, а также в различных направлениях и порядках следования. Обратите внимание: этот модуль имеет хорошую поддержку во многих современных браузерах. Он будет рассмотрен в главе 15.

И наконец, *модульные сетки* являются, наверное, наиболее амбициозной попыткой изменить способ компоновки веб-страниц. Они позволяют разбить страницу на набор строк и столбцов, затем аккуратно прикрепить элементы страницы к этой сетке. Он предназначен для упрощения создания веб-приложений и больше похож на создание компьютерных программ. Макеты с модульной системой верстки довольно сложны и (на момент написания данной книги) не поддерживаются большинством браузеров. Как ни странно, Internet Explorer — первый браузер, который начал поддерживать концепцию модульных сеток. Дополнительные сведения о методе можно получить по адресу tinyurl.com/nvqbktp.

Первые наброски

После решения вопроса с контентом можно подумать о его визуальной организации. Некоторые разработчики начинают с HTML-кода: создания `div`-контейнеров, добавления элементов `header`, `article`, `footer` и др. Это вполне разумный подход, поскольку он предоставляет отправную точку по созданию вашего сайта путем перехода непосредственно к HTML-коду.

Но перед переходом к HTML-коду нужно хотя бы набросать план контента веб-страницы. Здесь не требуется ничего особого, сгодятся бумага и карандаш. Поскольку контент планируется поместить в блоки (`div` и другие HTML-элементы) и расположить их на странице, простой набросок группы блоков с колонками и т. д. — быстрый и простой способ проработки макетов различных страниц. Можно быстро понять, куда должен помещаться контент, насколько объемным он будет и каким нужно сделать общий цветовой тон (к примеру, светлым или темным).

СОВЕТ

Ресурс Yahoo! предлагает бесплатный инструмент Stencil Kit (tinyurl.com/pl7rucb), который может применяться в Illustrator, Visio, OmniGraffle и других графических редакторах для создания макетов веб-страниц. Предоставляемые элементы пользовательского интерфейса, такие как

кнопки, элементы веб-форм, окна и инструменты навигации, позволят составить набросок макета страницы путем простого перетаскивания значков.

При наличии опыта работы с графическими редакторами наподобие Photoshop, Illustrator или Fireworks вы можете воспользоваться ими для создания визуального дизайна. Если вы не сильны в работе с такими программами, то простой рисунок блоков для обозначения различных позиций расположения элементов страницы поможет воплотить замысел о макете страницы. Намного проще будет изменить двухколоночный дизайн на четырехколоночный изменением размеров блоков в программе Illustrator, чем переписыванием кода HTML и CSS.

Но при использовании графических редакторов не стоит тратить слишком много времени на улучшение визуального дизайна. Не тратьте слишком много усилий для воссоздания в Photoshop или Illustrator форматирования, получаемого при использовании свойств каскадных таблиц стилей; для его переделки нужно будет воспользоваться CS-кодом. То есть утвердите предварительный вид своего сайта на бумаге, а затем переходите к редактору HTML-кода, в который поместите контент и используйте свойства каскадных таблиц стилей, изученные с помощью этой книги, для тестирования форматирования.

Определение блоков

После того как вы создали визуальный макет, нужно подумать о том, как создать разметку HTML и CSS для достижения дизайнерских задумок. Этот процесс обычно включает представление различных структурных блоков страницы и идентификацию элементов, которые выглядят как отдельные разделы. Например, на рис. 12.2 присутствует немало элементов, которые выглядят как небольшие разделы: наиболее крупным является раздел с тремя объявлениями внизу (отмечены на рис. 12.2 буквой А). Каждый раздел, как правило, является кандидатом на оборачивание в отдельный элемент `div` (при условии, что нет подходящего HTML-элемента, что мы обсуждали ранее).

Часто визуальная подсказка в макете может помочь решить, нужен ли вообще элемент `div`. Например, линия границы по краям заголовка и нескольких абзацев, означает, что вам понадобится обернуть эту группу HTML-элементов контейнером `div`, к которому применена граница.

Кроме того, если вы видите фрагменты текста, расположенные друг рядом с другом (например, три фрагмента с контентом в нижней части рис. 12.2), вы знаете, что нужно каждый из них заключить в отдельный элемент `div`. HTML-элементы, как правило, сами по себе не располагаются друг рядом с другом, поэтому вам придется использовать некоторые техники верстки (например, обтекаемые элементы, описанные в следующей главе).

Желательно также группировать элементы `div` (и другие элементы), расположенные рядом, в один общий контейнер `div`. Например, в нижней части рис. 12.2 вы видите набор контейнеров `div`, которые обеспечивают структуру страницы. Элементы `news` и `footer` являются контейнерами для своих собственных наборов `div`-контейнеров. Хотя это и не всегда необходимо, такой подход помогает обеспечивать гибкость. Например, вы можете уменьшить область основного контента

(изображение рук и рекламный слоган) по ширине и переместить раздел с новостями по его правую сторону, который бы сформировал отдельную колонку. Новости могли бы идти друг за другом, а не рядом друг с другом.

Следуя потоку

Элементы обычно не располагаются рядом и не наслаиваются друг на друга. Как правило, они действуют подобно тексту в текстовых редакторах: заполняют всю ширину страницы и следуют от верхнего до нижнего края. Каждый блочный элемент — заголовок, абзац, маркированный список и т. д. — располагается выше следующего блочного элемента. Поскольку это стандартный подход, вам обычно не нужно ничего делать, если вы планировали располагать элементы `div` один за другим.

Например, на рис. 12.2 четыре элемента — `header`, `div`-контейнер `main`, `section` и `footer` — охватывают всю ширину своего контейнера (элемента `body`) и расположены друг за другом по вертикали. Это типичная ситуация для блочных элементов, и вам не нужно делать ничего особенного, применяя каскадные таблицы стилей, чтобы расположить эти четыре элемента `div` таким образом.

Не забывая о фоновых изображениях

Вы, несомненно, видели мозаичные изображения, заполняющие фон веб-страницы, или градиенты, добавляющие эффект глубины баннеру. Свойство `background-image` предоставляет еще один способ добавления фотографий на страницу, не прибегая к помощи элемента `img`. Вставка изображения в качестве фона существующего элемента не только позволяет сохранить пару байтов данных, которые были бы потрачены на элемент `img`, но и упрощает решение некоторых проблем, связанных с компоновкой.

Например, на рис. 12.2 изображение рук по центру (B) на самом деле является обычным фоновым изображением. Это облегчает размещение другого элемента `div` со слоганом *Compassionate care...* (C), поскольку он расположен поверх фона родительского элемента `div`. Кроме того, фотография врача чуть ниже, в правой части страницы, — обычное фоновое изображение, размещенное в текущем элементе `div`. Добавление небольшого отступа справа сместит текст в этом разделе от фотографии.

ПРИМЕЧАНИЕ

Существуют недостатки использования фотографий в качестве фона контейнеров `div` (или других HTML-элементов). Во-первых, браузеры обычно не печатают фон, поэтому, если на странице должна быть изображена карта, содержащая какие-либо важные маршруты, вставьте ее с помощью элемента `img`, а не в качестве фонового изображения. Кроме того, поисковые системы игнорируют каскадные таблицы стилей, поэтому, если вы думаете, что изображение может привлечь дополнительный трафик на ваш сайт, используйте элемент `img` и добавьте описательный атрибут `alt`.

Кусочки пазла

Этот совет можно было подавать в разделе «креативного решения проблем». Часто то, что выглядит как одно целое, на самом деле состоит из нескольких частей. Вы

можете увидеть простой пример вышесказанного на рис. 12.2. Здесь есть четыре расположенных друг за другом элемента `div`, каждый из которых имеет белый фон.

Если у вас возникли проблемы с размещением больших элементов на странице (очень большого рисунка, занимающего несколько колонок, или сплошного цвета, заливающего не одну область страницы), подумайте о том, можно ли добиться желаемого внешнего вида страницы, разбив большой элемент на мелкие куски, которые потом сложатся, как кусочки пазла.

Элементы макета

Если вы работали в программе Photoshop, Illustrator или Fireworks, то, вероятно, понимаете концепцию слоев. Слои позволяют создавать отдельные холсты, которые накладываются друг на друга и образуют единое изображение. В этих программах можно легко поместить логотип поверх заголовка с текстом или одну фотографию поверх другой. Если вы хотите сделать то же самое на веб-странице, у вас есть несколько вариантов.

Часто самый простой способ наложить слой поверх фотографии заключается в добавлении изображения в качестве фона другого элемента. Поскольку фоновое изображение следует за элементом, все, что находится внутри этого элемента, — текст, другие изображения — будет расположено поверх фотографии.

Но что делать, если вы хотите наложить фотографию поверх того или иного текста? В таком случае нужно обратиться к единственному свойству, позволяющему наслаивать элементы, — `position`. Мы рассмотрим его в главе 15, поскольку размещение элементов поверх чего-либо требует абсолютного позиционирования.

Не забывая о полях и отступах

Иногда самое простое решение оказывается лучшим. Вам не всегда нужен причудливый CSS-код, чтобы поместить элемент в нужное место на странице. Вспомните, что отступы и поля представляют собой обычное пустое пространство и, используя их, вы можете перемещать элементы по странице. Например, рекламный слоган (см. рис. 12.2, С) размещается простой установкой верхнего и левого отступа для его родительского элемента. Как вы можете видеть на схеме в нижней части на рис. 12.2, слоган помещен внутри другого элемента `div` (`<div class="main">`). Этот элемент на самом деле не имеет другого контента, кроме слогана, — фотография является фоновым изображением, так что добавление отступа перемещает элемент `div` слогана вправо и вниз.

13 Макеты на основе обтекаемых элементов

Верстка на основе обтекаемых элементов использует преимущества свойства `float` для позиционирования элементов рядом друг с другом, создавая на веб-странице колонки. Как было описано в разделе «Управление обтеканием контента с помощью плавающих элементов» главы 7, вы можете использовать это свойство для создания эффекта обтекания, скажем, фотографии, но, когда вы применяете его к элементу `div`, оно становится мощным инструментом верстки страницы. Свойство `float` перемещает элемент в указанную сторону страницы (или другого блока с содержимым). Любой HTML-объект, который указан ниже обтекаемого элемента, изменяет свое положение на странице и обтекает его.

Свойство `float` принимает одно из трех значений: `left`, `right` или `none`. Чтобы выровнять изображение по правому краю страницы, вы можете создать класс и применить с его помощью стиль к элементу `img`:

```
.floatRight { float: right; }
```

То же самое свойство, примененное к элементу `div` с содержимым, позволяет создать боковую панель:

```
.sidebar {  
  float: left;  
  width: 25%;  
}
```

На рис. 13.1 показаны эти два стиля в действии.

ПРИМЕЧАНИЕ

Значение `none` отменяет любое выравнивание и определяет элемент как обычный (необтекаемый). Это полезно только для отмены выравнивания, которое уже применено к элементу. Допустим, у вас есть элемент, к которому применен специфический класс, такой как `.sidebar`, и этот элемент смещен вправо. На одной из страниц вы, возможно, захотите, чтобы элемент с этим классом не смещался, а был определен в общем потоке страницы, допустим, в качестве примечания. Создавая более специфичный селектор (см. раздел «Управление каскадностью» главы 5) с кодом `float: none`, вы можете предотвратить смещение этого элемента.

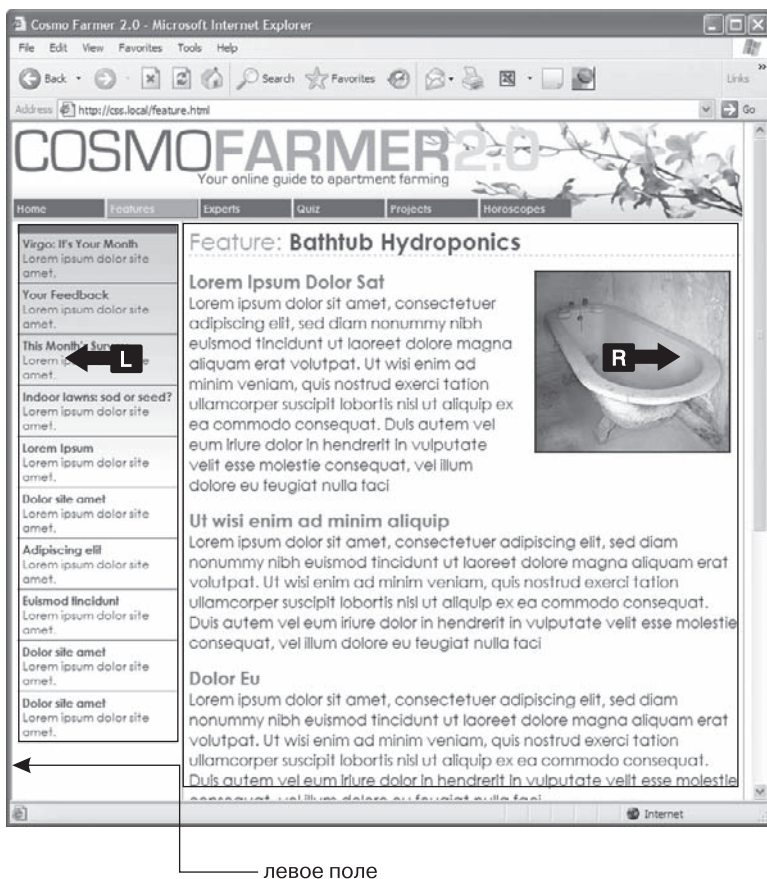


Рис. 13.1. Используйте свойство `float` для компоновки многоколоночной веб-страницы. Блок новостей выровнен по левому краю. У него есть фиксированная ширина, однако у основного контента ее нет, что делает этот дизайн резиновым. Основной раздел страницы расширяется, заполняя окно браузера. Вверху справа фотография с ванной выровнена по правому краю

Простой дизайн с двумя колонками наподобие показанного на рис. 13.1 требует выполнения всего нескольких действий.

1. Оберните каждую колонку элементом `div` с атрибутом класса или идентификатора.

На рис. 13.1 заголовки новостей, перечисленные в левой части, обернуты в один контейнер (`<div class = "news">`), а основной контент страницы — в другой (`<div class = "main">`).

2. Выровняйте элемент `div` с боковой панелью по правому или левому краю.

Когда вы работаете с обтекаемыми элементами, важен порядок исходного кода (порядок, в котором вы добавляете HTML-код в файл). HTML-код обтекаемого элемента должен быть указан перед HTML-кодом элемента, который оборачивает его.

На рис. 13.2 показаны три варианта двухколоночного дизайна. Схемы в левой части демонстрируют порядок HTML-кода страницы: элемент `div` баннера, за которым следует `div` боковой панели, и, наконец, элемент `div` основного контента. Справа вы видите фактический дизайн страницы. Боковая панель указана *ранее* основного контента в HTML-коде, так что она может переместиться или влево (*вверху и внизу*), или вправо (*посередине*).

3. Установите ширину для обтекаемой боковой панели.

Всегда ограничивайте ширину обтекаемых элементов. Таким образом вы позволите браузеру создать пространство для другого контента в результате обтекания.

В качестве значения ширины может быть задан фиксированный размер, такой как 170 пикселей или 10 em. Вы также можете использовать проценты для резинового дизайна, основанного на ширине окна браузера (см. подраздел «Ключевые слова, проценты и единица измерения em» раздела «Изменение размера шрифта» главы 6, чтобы узнать обо всех «за» и «против» различных единиц измерения). Если боковой панели задана ширина 20 %, а окно браузера — 700 пикселей в ширину, то ширина боковой панели составит 140 пикселей. Если же посетитель сайта изменит размер окна до 1000 пикселей, то боковая панель увеличится до 200 пикселей. Боковые панели с фиксированной шириной легче проектировать, так как не нужно просматривать различные значения ширины, до которых боковая панель могла бы растянуться.

Однако проценты позволяют вам поддерживать одинаковые пропорции между двумя колонками, что может быть визуально привлекательнее. Кроме того, проценты делают ваши конструкции гибкими, поскольку общие пропорции страницы могут подстраиваться под размер экрана, что играет важную роль при создании адаптивных веб-конструкций, речь о которых пойдет в следующей главе.

ПРИМЕЧАНИЕ

Если дизайн всей страницы указан с фиксированной шириной, значения ширины для боковой панели в процентах основаны на элементе с фиксированной шириной. Ширина не зависит от размера окна и остается постоянной.

4. Добавьте поле слева к основному контенту.

Если боковая панель короче другого контента на странице, то текст из основной колонки обтекает панель снизу. Добавление поля слева, больше ширины боковой панели или равного ей, выравнивает основной контент страницы, создавая иллюзию второй колонки:

```
.main { margin-left: 27%; }
```

Рекомендуется сделать поле слева чуть больше по ширине, чем боковая панель: так вы добавите промежуток между двумя элементами. Если в качестве значения ширины боковой панели вы используете проценты, то задавайте немного большее значение для поля слева.

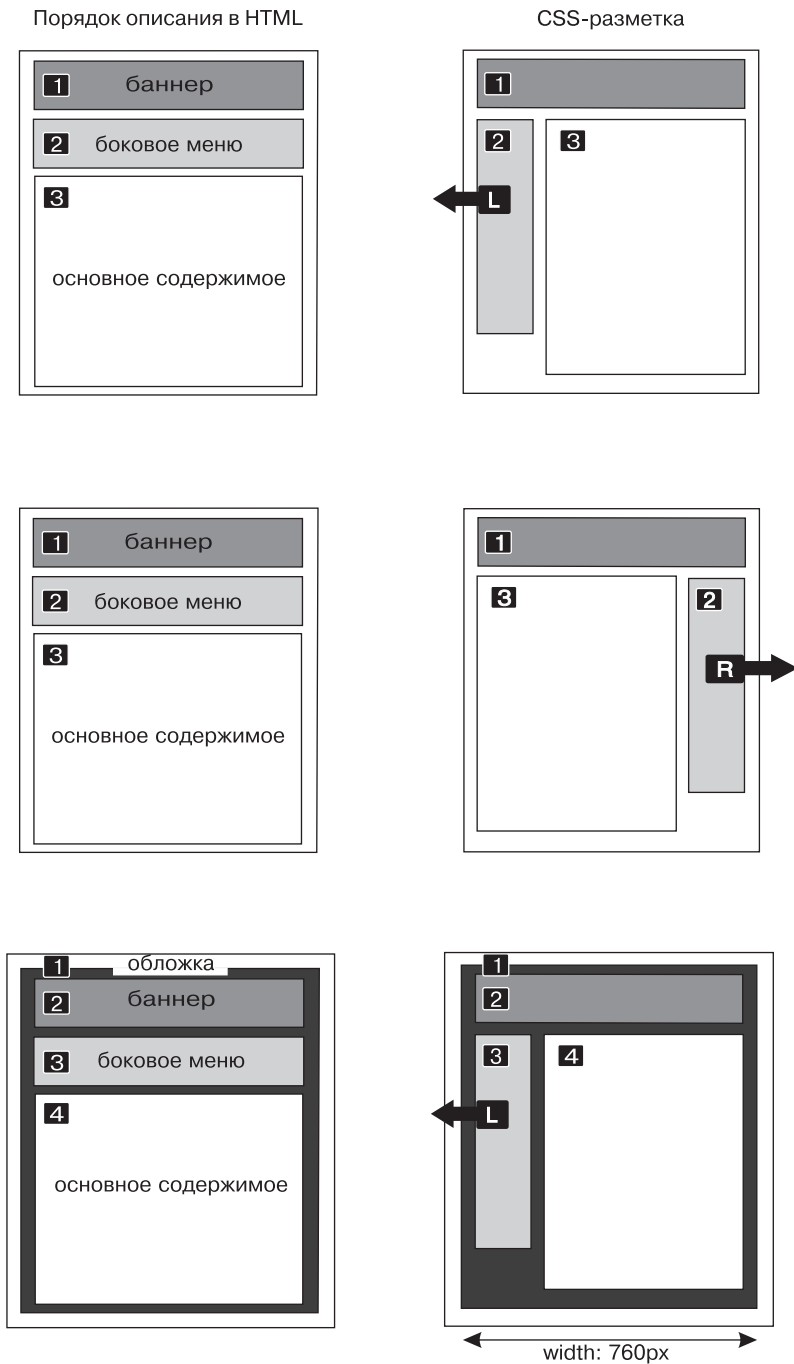


Рис. 13.2. Создание двухколоночного макета лишь вопрос выравнивания элемента div по левому краю (*вверху*). Чтобы переместить боковую панель в правую часть страницы (*посередине*), измените стиль боковой панели на `float: right`

Избегайте ограничения ширины контейнера `div` с основным контентом — браузеры расширяют его, чтобы он занимал доступное пространство. Даже если вы хотите создать фиксированный дизайн, вам не нужно ограничивать ширину основного контента, как вы узнаете в следующем разделе.

Использование обтекаемых элементов при верстке

Теперь, когда вы изучили простую двухколоночную резиновую верстку, можете применять ее бесчисленным множеством способов. Преобразовать ее в верстку с фиксированной шириной — просто. Надо обернуть все элементы внутри тела страницы *другим* элементом `div` (например, `<div class = "wrap">`), а затем создать стиль для этого нового элемента-контейнера, для которого определена ширина, скажем, 960 пикселей (см. рис. 13.2, *внизу*). Подобное ограничение ширины удерживает контент внутри контейнера.

ПРИМЕЧАНИЕ

Существует также вариант создания страницы с фиксированной шириной без применения дополнительного элемента `div`, оборачивающего все элементы: ограничьте ширину элемента `body`. Вы уже видели пример использования этого метода в практикуме главы 2.

Развернуть контент страницы на три колонки также не представляет сложности (рис. 13.3). Сначала добавьте другой элемент `div` между этими двумя колонками и выровняйте его по правому краю. Затем добавьте поле справа к средней колонке так, чтобы, если текст в ней окажется длиннее новой правой боковой панели, он не обтекал боковую панель.

В остальной части этого раздела рассматриваются многочисленные методы CSS-верстки с применением обтекаемых элементов.

Выравнивание всех колонок

В полной мере можно выравнивать *каждую* колонку, а не только левую и правую боковые панели. Вы можете выровнять первую боковую панель и среднюю колонку по левому краю, а правую боковую панель — по правому краю, как показано на рис. 13.2, *вверху*. Этот подход позволяет размещать более трех колонок на странице. Вы можете выравнивать четыре колонки и более, пока есть пространство для всех обтекаемых элементов, чтобы они находились рядом друг с другом.

Если вы выравниваете все колонки в макете, то должны обратить пристальное внимание на ширину каждой из них. Если совокупная ширина всех колонок меньше доступного пространства, например, если окно браузера меньше или колонки обернуты другим элементом `div` с определенной шириной, то последняя колонка опускается вниз (как решать проблему выпадения обтекаемых элементов, вы можете прочитать в разделе «Решение проблем с обтекаемыми элементами» этой главы).

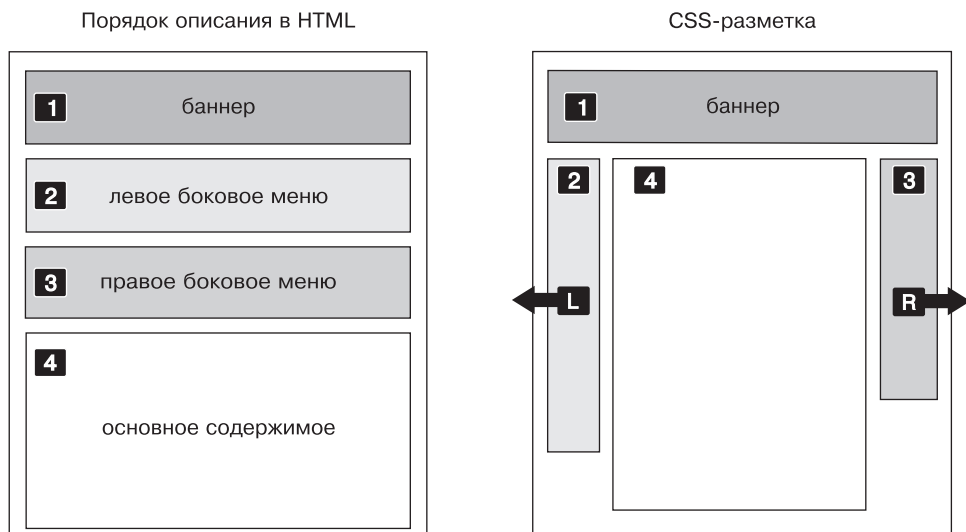


Рис. 13.3. Для трехколоночных макетов используется та же концепция, что и для двухколоночных. При трехколоночном дизайне вы выравниваете левую и правую панель и добавляете поля слева и справа к центральной колонке. Схема в левой части демонстрирует порядок следования HTML-кода, а справа вы можете увидеть, как выглядит веб-страница

Вдобавок выравнивая не только боковые панели, вы сможете изменить порядок разделов `div` в HTML-коде. Используйте, к примеру, левую схему на рис. 13.3, которая демонстрирует порядок следования элементов `div` для страницы. По принципу своей работы обтекаемые элементы должны появляться перед любым контентом, который окружает их, так что в этом примере область основного контента должна следовать *после* боковой панели.

В КУРС ДЕЛА!

Нужно ли снова изобретать колесо?

Если такие термины, как *резиновый макет* и *контейнерный элемент*, кажутся немного пугающими, не поддавайтесь панике. Прежде всего практикум в конце этой главы шаг за шагом проведет вас через весь процесс верстки веб-страниц с помощью каскадных таблиц стилей. Однако нет такого правила, которое бы информировало о том, как вы должны создавать свои собственные CSS-макеты с нуля. Во Всемирной паутине вы найдете множество шаблонных и протестированных макетов, которые можно применять в своих проектах. Сайт Layout Gala содержит 40 различных CSS-дизайнов,

которые функционируют в большинстве популярных браузеров (tinyurl.com/nfesllb). Представленные макеты — каркасы, состоящие из элементов `div` и CSS-кода, который позиционирует их. Все, что вам нужно сделать, — заполнить их собственными стилями форматирования, такими как свойства шрифта и изображения.

Посетите также сайт Templated (templated.co), который содержит свыше 850 бесплатных CSS- и HTML-шаблонов. Эти современные шаблоны включают изображе-

ния, фоновые цвета и свойства типографики, которые принесут вашему сайту популярность.

Существует также немало *генераторов макетов* — эти онлайн-инструменты позволяют настраивать такие основные параметры, как количество колонок, вид макета (резиновый или фиксированный) и т. д. Сайт Layout

Generator (pagecolumn.com) содержит простой инструмент, с помощью которого можно создать многоколоночный дизайн страницы, после чего сгенерированные HTML- и CSS-файлы можно будет скачать. В главе 16 вы узнаете о том, как использовать *модульные сетки* — CSS-файлы с колонками, организованными определенным способом, — для создания страниц со сложным дизайном.

Соблюдение порядка элементов `div` в HTML-коде может показаться чем-то очень важным, пока вы не попытаетесь просмотреть веб-страницу без каскадных таблиц стилей, что имеет место для многих альтернативных браузеров, включая программы экранного доступа, которые произносят контент страницы вслух для слабовидящих посетителей. Без каскадных таблиц стилей весь контент боковой панели (он часто включает навигационные элементы, рекламу и другую информацию, не относящуюся к основному контенту страницы) появится перед содержимым, ради которого зашел посетитель. Неудобство, заключающееся в необходимости прокручивать одно и то же содержимое боковой панели на каждой странице, может отпугнуть многих посетителей. Кроме того, ваша страница будет неудобна пользователям с недостатками зрения, которым придется выслушивать, как их экранный диктор читает длинный список ссылок и рекламы, прежде чем получить реально необходимую информацию.

Если же и это никак не затрагивает ваш проект, то стоит поволноваться насчет поисковых систем. Многие из них ограничивают количество HTML-кода, считываемое при поиске сайта. На особенно длинной веб-странице они просто останавливаются на определенном месте, возможно, упустив важное содержимое, которое *должно* быть проиндексировано. Кроме того, большинство поисковых систем придает большее значение тому HTML-коду, который находится в начале файла. Таким образом, если вас волнует релевантность вашего сайта при выдаче результатов поиска такими системами, то имеет смысл убедиться в том, что важный контент расположен как можно ближе к началу HTML-кода страницы.

На левой верхней схеме на рис. 13.4 HTML-код с основным контентом находится между левой и правой боковыми панелями, что лучше, чем помещать его после этих блоков. Вы можете даже поместить основной контент перед HTML-кодом *обеих* боковых панелей, оборачивая его и левую боковую панель одним элементом `div` и выравнивая его по левому краю. Затем *внутри* этого элемента `div` нужно выровнять основной контент по правому краю, а левую боковую панель — по левому краю (см. рис. 13.4, *внизу*). Теперь HTML-код основной колонки расположен перед остальными элементами `div`.

Вложенные обтекаемые элементы

Нижняя схема на рис. 13.4 иллюстрирует другую полезную методику — выравнивание элементов *внутри* обтекаемых элементов. Предположим, что `div`-обертка основного контента (3) и левой боковой панели (4) не существует, а были оставлены только

обертка колонки (2) и правая боковая панель (5). У вас получится базовый двухколоночный дизайн, где одна колонка выровнена по левому краю, а другая — по правому краю. На самом деле это все еще двухколоночный дизайн, хотя две div-обертки (3 и 4) помещены внутри обертки колонки. Различие в том, что левая колонка сама разделена на две колонки.

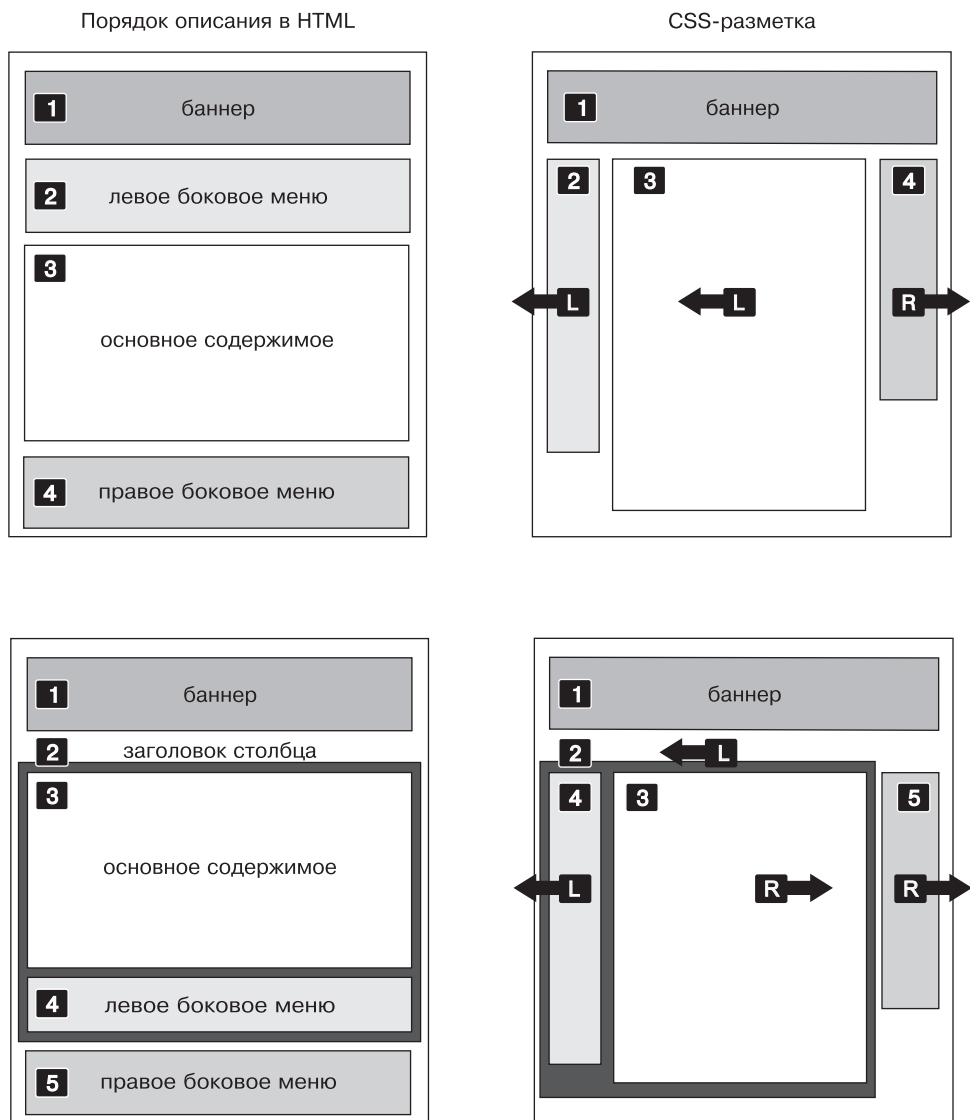


Рис. 13.4. Существует несколько способов сверстать страницу с обтекаемыми элементами. Гибкость каскадных таблиц стилей предоставляет множество вариантов создать многоколоночный макет

Хотя подобное выравнивание слегка смущает, оно бывает полезным во многих случаях. Во-первых, оно позволяет добавлять колонки внутри других колонок. В трехколоночном макете, показанном на рис. 13.5, *вверху*, используется маленький блок для советов в средней колонке, внутри которого, в свою очередь, также созданы две колонки. Вкладывая одни выравниваемые элементы внутри других, вы можете создавать достаточно сложные дизайны.

Решение проблем с обтекаемыми элементами

Когда вы начнете активно работать с каскадными таблицами стилей, то, как и многие веб-разработчики, вероятно, столкнетесь с некоторыми сложностями при работе с выравниваемыми (обтекаемыми) элементами. В этом разделе описаны распространенные проблемы и пути их решения.

Запрет обтекания и элементы-контейнеры

Выравниваемые элементы — мощные средства проектирования, поскольку позволяют содержимому обтекать их по краям. Выравнивание фотографии позволяет тексту, находящемуся ниже, подняться и обернуть изображение (см. рис. 13.1). Когда вы разрабатываете дизайны, основанные на выравниваемых колонках, иногда не требуется, чтобы содержимое передвигалось и оказывалось рядом с выровненным элементом. Например, вы хотите указать записи об авторском праве, контактную информацию или другие сведения в нижней части веб-страницы, ниже остального контента.

В двух и трехколоночных макетах, которые мы уже рассматривали, если основная колонка короче любой колонки с обтекаемой боковой панелью, нижний колонтитул может сместиться вверх, оказавшись рядом с левой выровненной колонкой (рис. 13.6, *слева*). Чтобы оставить нижний колонтитул под боковой панелью, вы можете использовать свойство `clear` (см. раздел «Управление обтеканием контента с помощью плавающих элементов» главы 7). Оно устанавливает, с какой стороны элемента запрещено его обтекание другими элементами. Вы можете отменить обтекание с левого края элемента. При этом все другие элементы на этой стороне опустятся вниз и будут располагаться под текущим (`clear: left;`). Аналогично свойство `clear: right;` отменяет обтекание с правой стороны элемента. Для нижнего колонтитула и других элементов, которые должны находиться в нижней части страницы, вы должны устранить обтекание *как* слева, *так и* справа:

```
footer { clear: both; }
```

Другая проблема возникает, если вы выравниваете один или несколько элементов внутри необтекаемого элемента-контейнера, такого как `div`. Если обтекаемый элемент выше, чем остальной контент в контейнере, он придерживается основания содержащего его объекта. Эта путаница особенно видна, если у элемента есть фоновый цвет или граница. В верхней части веб-страницы, показанной на рис. 13.7, используется элемент `div`, в котором определены заголовок `h1` и две колонки, созданные двумя обтекаемыми контейнерами `div`. Фон и граница, которые появляются только по краям

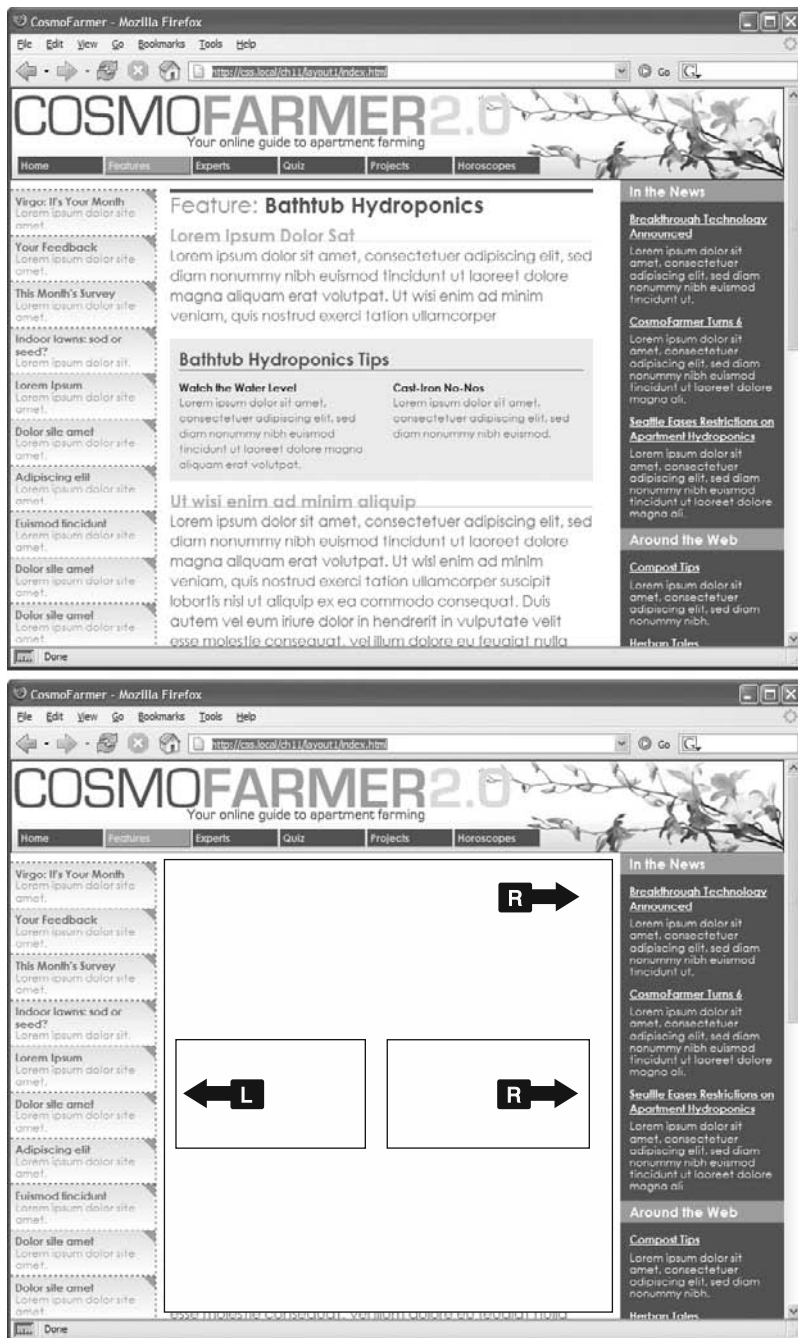


Рис. 13.5. Вверху: создавайте вложенные колонки выравниванием элементов внутри других обтекаемых элементов. Внизу: не важно, в каком направлении выравнивается контейнер (в этом случае — по правому краю), — вы лишь выравниваете две дополнительные колонки по левому и правому краю

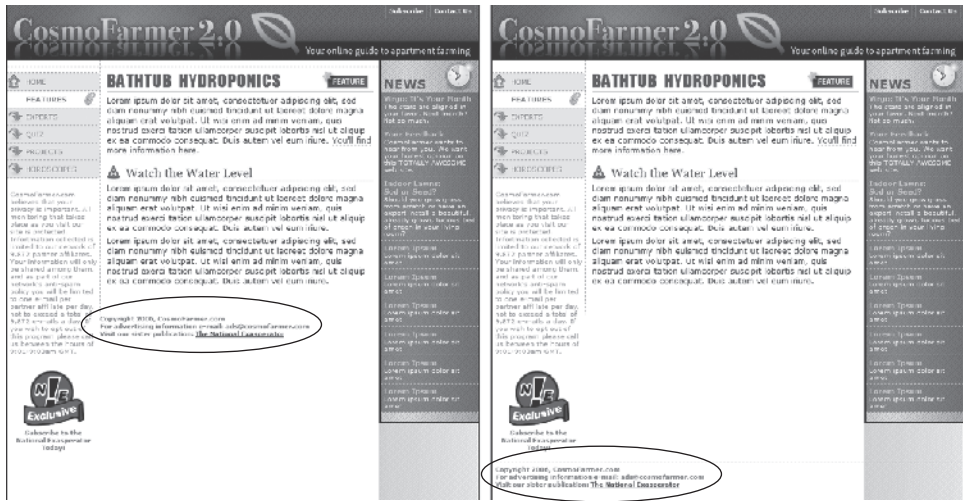


Рис. 13.6. Объект не всегда должен обтекать по краям выравниваемого элемента (слева). Сведения об авторских правах и другие материалы, расположенные в нижней части страницы, всегда должны располагаться отдельно от обтекаемых элементов, находящихся рядом. Чтобы добиться этого, используйте свойство clear

заголовка, в действительности применяются ко всему элементу div, включая область с двумя колонками. Однако, поскольку колонки выравниваются, они выходят за пределы блока вместо того, чтобы расширить границы области.

ПРИМЕЧАНИЕ

Чтобы узнать, почему обтекаемые элементы могут выходить за пределы содержащих их блоков, посетите сайт tinyurl.com/369n3.

Пример подобной проблемы показан на рис. 13.7. В этом случае каждое изображение выровнено по левому краю внутри содержащего их элемента div, для которого задана граница. Поскольку изображения выше, чем их блоки, они выходят за пределы блоков. К сожалению, этот пример еще хуже, чем предыдущий, потому что каждый рисунок смещает нижнее изображение вправо, создавая ступенчатый эффект.

Избавиться от проблем, возникающих с обтекаемыми элементами, можно несколькими способами. Перечислим наиболее популярные из них.

- **Добавьте в конце div-контейнера запрещающий обтекание элемент.** Это решение — наиболее простое. Нужно лишь добавить элемент, например разрыв строки или горизонтальную линию, в конце контейнера div, содержащего обтекаемый элемент (то есть прямо перед закрывающим тегом </div>). Затем используйте свойство clear, чтобы установить этот дополнительный элемент под обтекаемыми элементами. Такой метод расширяет контейнер, выявляя его фон и границу. Вы можете указать разрыв строки —
 (HTML) или
 (XHTML) — перед закрывающим тегом </div> и добавить к нему класс: <br class = "clear"/>. Затем создайте для него следующий стиль:

```
br.clear { clear: both; }
```

Недостаток метода заключается в добавлении дополнительного HTML-кода.

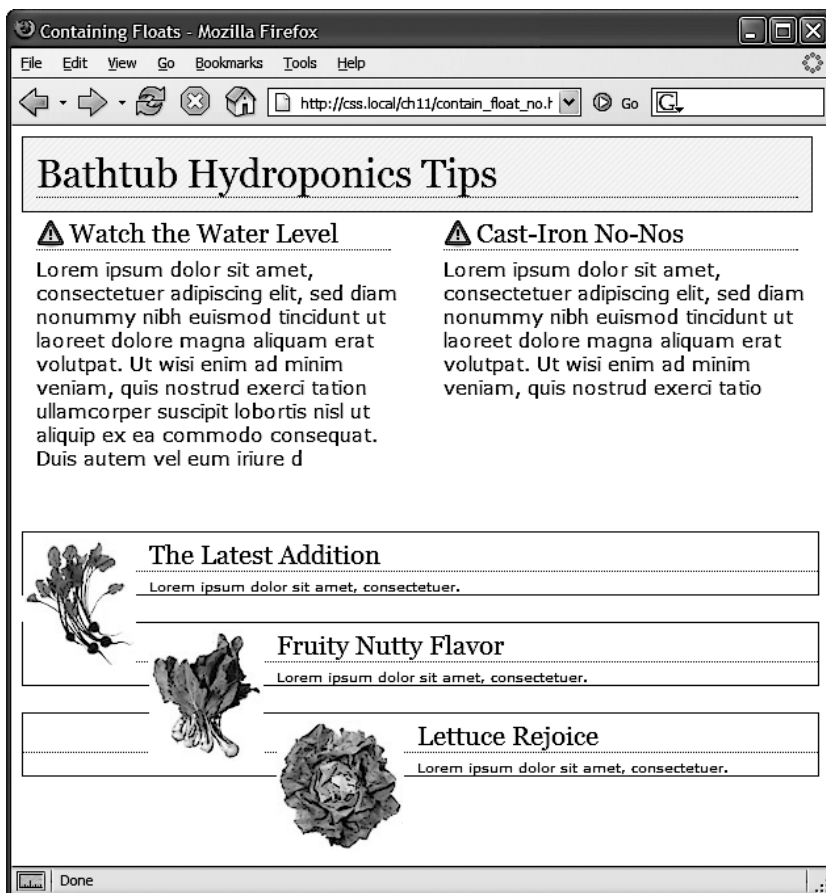


Рис. 13.7. Обтекаемый элемент может выйти за пределы содержащего его блока, если он выше его. Если для блока определены фоновый цвет или граница, то выходящие элементы могут выглядеть так, как будто даже не являются частью блока (*вверху*)

- **Выровняйте элемент-контейнер.** Более легкий путь состоит в том, чтобы выровнять элемент `div`, который содержит обтекаемые элементы. Выровненный контейнер `div` расширяется так, чтобы полностью вмещать любые обтекаемые элементы. На рис. 13.8, *вверху*, элемент `div`, содержащий заголовок и две обтекаемые колонки, выровнен по левому краю страницы. При необходимости вся его область — фон и границы — расширяется, чтобы соответствовать контенту внутри, включая обтекаемые элементы. Это странно, но это так.

Если вы выбрали такой метод, убедитесь, что добавили свойство `clear` к любому элементу, который расположен после обтекаемого контейнера. Так вы гарантируете, что следующий элемент будет находиться под контейнером.

- **Используйте свойство `overflow: hidden`.** Другой распространенный метод состоит в добавлении следующего свойства в стиль блока-контейнера:

```
overflow: hidden;
```

- **Свойство `overflow: hidden`** — одно из странностей каскадных таблиц стилей. Оно провоцирует контейнер расширяться и вмещать обтекаемые элементы. В целом этот метод работает очень хорошо. Тем не менее, если у вас есть абсолютно позиционированные элементы внутри контейнера, они могут не отобразиться. Вы можете попасть в такую ситуацию, если у вас есть раскрывающийся список внутри другого элемента и, когда он раскрывается, кажется, что список находится за пределами элемента-контейнера. Если это так, используйте какой-либо другой способ из описанных на этих страницах.
- **Воспользуйтесь методом очистки потока (`clearfix`)**. Эта технология, созданная Николасом Галлахером, заключается в добавлении к элементу, содержащему обтекаемый элемент, всего нескольких стилей и имени класса. Такой способ наиболее свеж в длинной эволюции методов, использующих псевдокласс `:after`. Чтобы воспользоваться им, нужно добавить в таблицу стилей следующий код:

```
.clearfix:after {
  content: " ";
  display: table;
  clear: both;
}
```

После добавления этих стилей в таблицу к `div`-контейнеру, *содержащему* выпадающие обтекаемые элементы, нужно лишь добавить класс: `<div class="clearfix">`. Если используются семантические элементы HTML5, например `article` или `footer`, класс нужно добавить и к ним: `<article class="clearfix">`. Взгляните на нижнее изображение на рис. 13.8. Эта технология работает очень надежно, но, в отличие от предыдущих двух технологий, вам потребуется наличие на странице дополнительного HTML-кода (класса).

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Простой способ создания нескольких колонок

В CSS представлен модуль *многоколоночной верстки*: он позволяет разделить единый элемент (например, заполненный текст `div`-контейнер) на три, четыре колонки или более. Модуль предоставляет свойства каскадных таблиц стилей для определения количества колонок, пусто пространства между ними и добавления линий между колонками.

Такой способ верстки предназначен для имитации газетных или журнальных страниц, в которых одна длинная публикация простирается от верхней до нижней части одной колонки, продолжается во второй колонке и т. д. То есть он не предназначен для колонок не связанного между собой контента (например, боковой панели со ссылками и колонки, содержащей основной контент).

Кроме того, для каждой отдельной колонки нельзя указать ширину, отличную от ширины других колонок, все они будут идентичны.

Верстка в несколько колонок хорошо смотрится на страницах журналов, у которых страница имеет определенный размер и вы можете видеть ее всю, что упрощает чтение текста в колонке до конца страницы и переход к верхней части следующей колонки в верхней части страницы. Но в случае с веб-страницей размер экрана по высоте устройства посетителя вам неизвестен, поэтому длинная колонка текста может заставить посетителя прокручивать страницу вниз, чтобы добраться до нижней части, затем прокручивать ее вверх, чтобы прочитать следующую

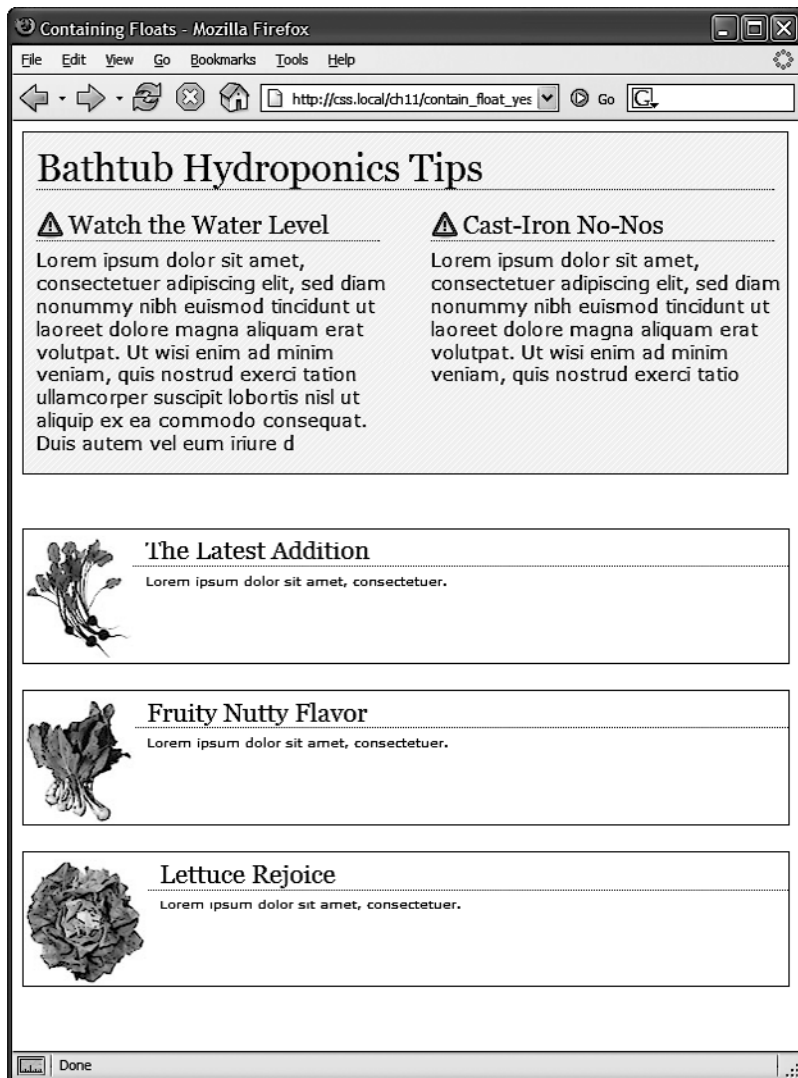


Рис. 13.8. Не позволяйте обтекаемым элементам выпадать из нужных позиций на странице. Есть несколько способов сохранить обтекаемые элементы внутри содержащего их контейнера

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

колонку. Без учета размера колонок вы заставите людей прокручивать страницу вниз-вверх, чтобы прочитать ее контент.

Тем не менее технология создания нескольких колонок хорошо подойдет для длинных маркированных списков с небольшими фрагментами текста в пунк-

тах. Вместо одного длинного маркированного списка можно будет распределить пункты по странице в нескольких колонках, экономя драгоценное вертикальное пространство.

Для задания количества колонок можно воспользоваться свойством `column-count`, для настройки про-

межутков между ними — свойством `column-gap`, а для отображения линий между колонками — свойством `column-rule`. Чтобы установить стиль, размер и цвет линий между колонками, используется точно такой же синтаксис, как и для границ (см. раздел «Добавление границ» главы 7).

Эти свойства применяются к элементу, содержащему контент, который нужно разбить на колонки. Предположим, что имеется несколько абзацев текста в контейнере `div`. При применении этих свойств к такому `div`-контейнеру абзацы будут заливать в несколько колонок. Если, к примеру, этому контейнеру присвоен класс с именем `multicol`, то для получения трехколоночной конструкции с промежутками между колонками `1em` и пунктирными линиями можно создать следующий стиль:

```
.multicol {
  column-count: 3;
  column-gap: 1em;
```

```
  column-rule: 1px dotted black;
}
```

Следует напомнить, что Internet Explorer 9 и более ранние версии этого браузера не поддерживают многоколоночные свойства. Кроме того, нужно использовать вендорные префиксы для браузеров Chrome и Safari (`-webkit-column-count`, `-webkit-column-gap` и `-webkit-column-rule`) и Firefox (`-moz-column-count`, `-moz-column-gap` и `-moz-column-rule`). А браузеры Internet Explorer 10 (и версии выше), Edge и Opera поддерживают эти свойства без префиксов.

Существуют и другие свойства для задания нескольких колонок, о которых можно прочитать на официальной странице Консорциума W3C: tinyurl.com/cjf2ad. Кроме того, простое руководство по созданию нескольких колонок доступно по адресу tinyurl.com/pg96bv3, а интерактивное средство для генерации многоколоночной верстки — по адресу tinyurl.com/ok33y7b.

Заполнение колонок по высоте

HTML-таблицы не лучшее средство для верстки веб-страниц. Они требуют большого объема кода, их трудно обновлять и они не работают так же хорошо в альтернативных браузерах, например тех, что используются в смартфонах. Но, что касается верстки, у таблиц есть один плюс — возможность создавать колонки одинаковой высоты. Их применение позволяет добавлять фоновый цвет или изображение к отдельной колонке и заполнять колонками всю высоту страницы. Фоны двух боковых панелей, показанных на рис. 13.9, *вверху*, заполняют всю высоту, создавая сплошные, широкие полосы с обеих сторон страницы.

Обтекаемые элементы не так великолепно справляются с задачей. Ячейки таблицы в строке всегда одной и той же высоты, чего не скажешь о `div`-контейнерах. Высота обтекаемого элемента обычно определяется его контентом. Когда его мало, сам элемент тоже мал. Поскольку фоновое изображение или фоновый цвет заполняют только обтекаемый элемент, у вас могут получиться колонки другого цвета, не достигающие основания страницы, как выделено на рис. 13.9, *внизу*.

ПРИМЕЧАНИЕ

Гибкая блочная модель решает проблему разности высоты ячеек в строке. О ней вы узнаете в главе 15.

Но, как и у большинства проблем, связанных с каскадными таблицами стилей, доступно несколько обходных путей. Наиболее действенным из них является метод *псевдоколонок*. Секрет состоит в том, чтобы добавить фоновые изображения

к элементу, который *оборачивает* низкую боковую панель и другие колонки на странице. Скажем, в вашем HTML-коде есть два элемента `div`, в которых содержится контент левой боковой панели и основной контент страницы:

```
<div class="sidebar">Контент боковой панели</div>
<div class="main">Основной контент страницы, в этой колонке много текста и она на-
много выше боковой панели.</div>
```

Элемент `div` боковой панели выравнивается по левому краю страницы, и ширина его равна 170 пикселям. Поскольку на боковой панели немного текста, она короче, чем основной контент. Предположим, что вы оборачиваете данный HTML-код в `div`-контейнер следующим образом:

```
<div class="wrapper">
<div class="sidebar">Контент боковой панели</div>
<div class="main">Основной контент страницы, в этой колонке много текста и она на-
много выше боковой панели.</div>
</div>
```

Внешний контейнер увеличится и станет таким же высоким, как самый высокий элемент внутри него, поэтому, если контент `div`-контейнера с классом `main` очень велик, обертывающий блок `div` с классом `wrapper` будет такой же высоты. В этом все волшебство: создайте стиль для обертки `div` с фоновым изображением, ширина которой равна ширине боковой панели, подобрав нужный фоновый цвет. Таким образом, если фоновое изображение повторяется по вертикали, оно сформирует сплошную полосу размером, равным высоте `div`-контейнера (рис. 13.9, *вверху*).

```
.wrapper { background: url(images/col_bg.gif) repeat-y left top; }
```

Браузеры отображают это фоновое изображение прямо *позади боковой панели*, создавая иллюзию того, что у панели есть фоновый цвет.

ПРИМЕЧАНИЕ

Вы не ограничены сплошным цветом. Поскольку допускается использовать изображения, можно создать декоративный узор, который будет чередоваться до конца страницы.

Применение этого метода к двум колонкам немного сложнее. Во-первых, нужно добавить два контейнера-обертки:

```
<div class="wrapper1">
<div class="wrapper2">
<div class="sidebar1">Контент первой боковой панели</div>
<div class="sidebar2">Контент второй боковой панели</div>
<div class="main">Основной контент страницы, в этой колонке много текста и она на-
много выше боковых панелей.</div>
</div>
</div>
```

ПРИМЕЧАНИЕ

Если обертка и каждая колонка имеют фиксированную ширину, вы можете создать вид псевдоколонки для левой и правой панели с одним изображением и `div`-оберткой. Для этого сделайте рисунок таким же широким, как и `div`-обертка, и с левой стороны залейте цвет по ширине левой боковой панели, а с правой — цвет по ширине правой панели. Центральная часть должна быть цвета центральной колонки.

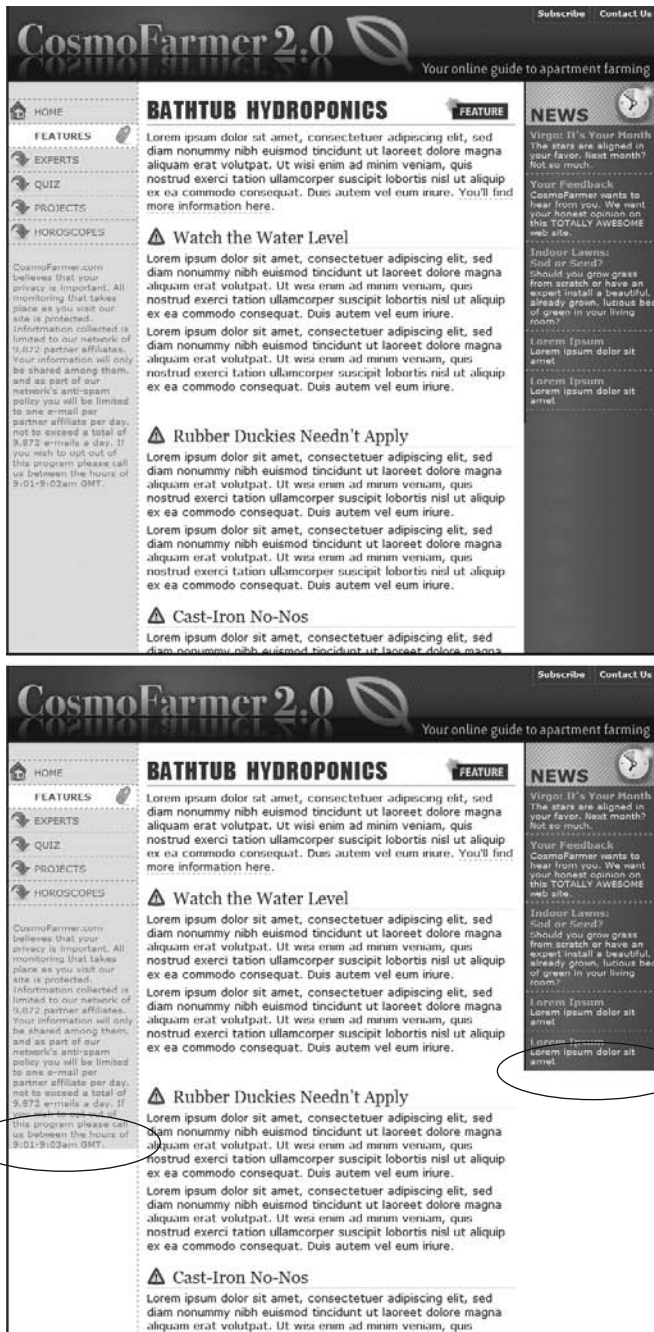


Рис. 13.9. Колонки во всю высоту страницы с фоновым цветом — распространённый прием в веб-дизайне. Боковые панели слева и справа (*вверху*) демонстрируют, как сплошной фоновый цвет помогает визуально разделить области страницы. Когда фон боковой панели резко прерывается (*внизу*), появляется пустое пространство, которое выглядит непривлекательно

Если первая боковая панель отображается в левой части страницы, а вторая — в правой, то создается два стиля. Примените один стиль к первому контейнеру `div`, чтобы добавить фон к левой боковой панели; другой же стиль примените ко второму контейнеру `div`, добавив фон к правой боковой панели (рис. 13.10, *внизу*).

```
.wrapper1 { background: url(images/col1_bg.gif) repeat-y left top; }
.wrapper2 { background: url(images/col2_bg.gif) repeat-y right top; }
```

При добавлении фонового изображения к правой колонке убедитесь, что вы помещаете изображение вверху справа во второй обертке так, что оно простирается позади второй боковой панелью в правой части страницы.

Основная проблема псевдоколонок связана с тем, что их очень трудно заставить работать, если для всех колонок ширина задана в процентах. Если ширина боковых панелей задана в процентах от ширины окна браузера, они могут быть уже или шире в зависимости от характеристик монитора посетителя. Техника псевдо-колонок требует помещения изображения в элемент, являющийся контейнером. Это изображение имеет определенную ширину и не будет масштабироваться при изменении ширины окна браузера, а соответственно, и ширины колонок.

Один из удачных обходных маневров — использование линейных градиентов (см. раздел «Использование градиентных фонов» главы 8) в элементах-контейнерах. Градиент становится для колонок цветом фона внутри элемента-контейнера. С его помощью можно также создать эффект сплошного цвета.

Как уже говорилось, линейные градиенты позволяют устанавливать цветовые узлы, то есть позиции, в которых появляется новый цвет. Если первый и второй цветовой узлы имеют одинаковый цвет, к примеру белый, вместо градиента с переходами цвета вы получите сплошной белый фон. Кроме того, можно установить второй цветовой узел в позиции первого, чтобы второй цвет отображался сразу после первого без какого-либо градиентного эффекта.

Предположим, что есть трехколоночный дизайн. Первая колонка имеет ширину 25 %, вторая — 50 %, третья — 25 %. Нужно, чтобы у первой колонки фоновым цветом был красный, у второго — белый, а у третьего — синий.

1. Заключите все три колонки в элемент-контейнер:

```
<div class="wrapper">
  <div class="sidebar1">...сюда помещается содержимое...</div>
  <div class="main">...сюда помещается содержимое...</div>
  <div class="sidebar2">...сюда помещается содержимое...</div>
</div>
```

Контейнер является тем самым элементом, к которому добавляется градиент. Кроме того, если все три колонки внутри контейнера обтекаемы, то для содержания всех этих обтекаемых элементов нужно будет воспользоваться одним из методов, рассмотренных ранее в разделе «Решение проблем с обтекаемыми элементами».

2. Добавьте линейный градиент с цветовыми узлами в соответствии с шириной колонок:

```
.wrapper {
  background-image: linear-gradient(left,
```

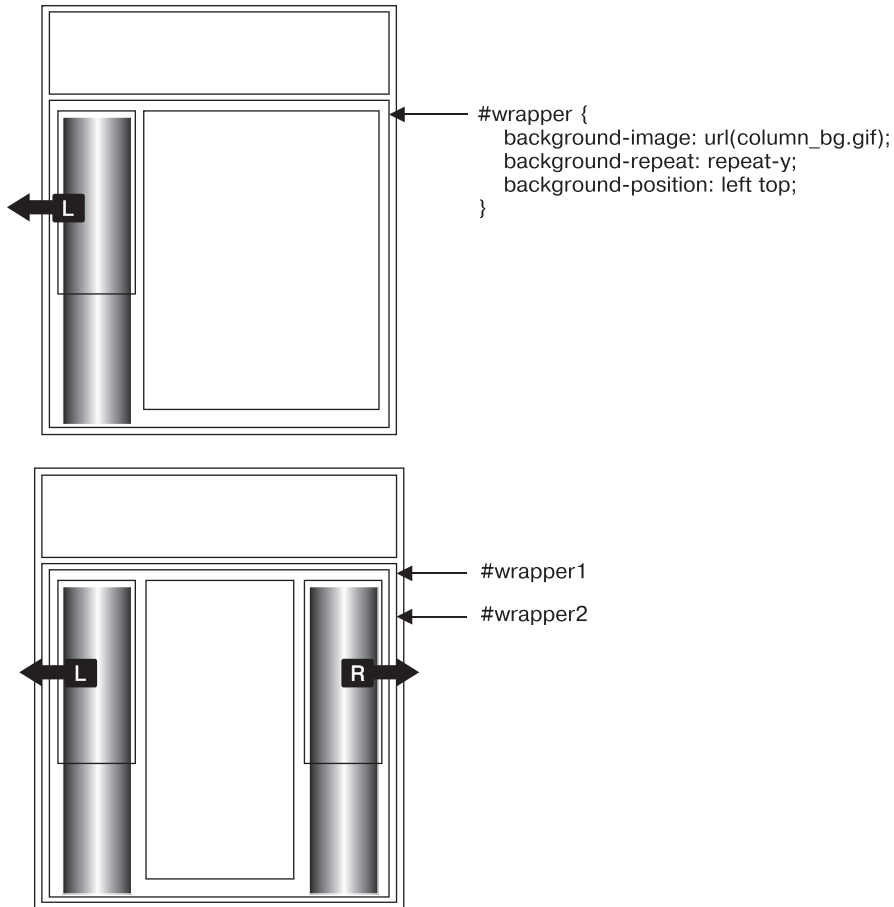



Рис. 13.10. Для креативного решения проблем каскадных таблиц стилей иногда необходимо выходить за пределы элемента

```
red 0%,
red 25%,
white 25%,
white 75%,
blue 75%,
blue 100%);
}
```

Красный цвет займет пространство от 0 % (от левого края контейнера) до 25 %. Поскольку в обоих узлах используется один и тот же цвет, перехода не будет. Затем на отметке 25 %, там же, где закончился красный цвет, начинается белый цвет, поэтому градиента опять не будет. Белый сплошной цвет займет пространство до 75 % отметки. Затем в позиции, где заканчивается белый цвет, начнется синий цвет, который займет пространство до 100 % (до правого края контейнера). То есть получатся три колонки со сплошным фоновым цветом, ширина которых будет изменяться по мере изменения ширины окна браузера.

Недостаток использования линейного градиента заключается в том, что он поддерживает только сплошные цвета (и, разумеется, градиенты, если вы того желаете), поэтому вы не можете использовать в качестве фона изображения или границы по краям каждой колонки. Кроме того, Internet Explorer 9 и ранние версии этого браузера не поддерживают градиенты.

ПРИМЕЧАНИЕ

Есть еще несколько способов отобразить колонки с одинаковой высотой. Обзор различных технологий можно найти в блоге эксперта по каскадным таблицам стилей Криса Койера по адресу tinyurl.com/p5swv4s.

ПРИМЕЧАНИЕ

Существует техника для создания элемента, находящегося позади обтекаемой колонки, которая заключается в использовании псевдокласса `:before`. Подробнее о ней можно прочитать по адресу tinyurl.com/q7lglhu.

Предотвращение выпадений обтекаемых элементов

Может случиться так, что внезапно одна из колонок на вашем сайте опустится ниже других (рис. 13.11, *вверху*). Как видно из рисунка, на странице достаточно пространства для размещения всех колонок рядом друг с другом, но этого не получилось. В данном случае произошло *выпадение обтекаемого элемента*. Если ширина обтекаемых элементов хотя бы на пиксел шире содержащего их блока (например, контейнера `div` или даже окна браузера), то последний элемент опускается ниже других (см. рис. 13.11, *вверху*). Фактическая высота элемента — это комбинация множества свойств каскадных таблиц стилей. На среднем изображении видно, что контур по краям области основного контента слишком широк (выделено на рисунке), чтобы позволить правой боковой панели занять желаемое расположение. Скорректировав значения ширины, отступов или полей всех элементов, вы можете решить проблему (см. рис. 13.11, *внизу*).

Обтекаемая колонка опускается вниз из-за недостатка пространства. Будьте осторожны, если вы ограничиваете ширину *каждой* колонки. Если ширина доступного пространства в окне браузера (или контейнера в фиксированном макете) меньше общей ширины колонок, то обтекаемый элемент может опуститься вниз. Кроме того, не забывайте о блочной модели CSS: как обсуждалось в разделе «Изменение высоты и ширины» главы 7, ширина элемента, отображаемого в окне браузера, не определяется лишь значением присвоенного свойства `width`. Отображаемая ширина любого элемента — это комбинация его размеров контента, а также границ, отступов и полей слева и справа. Для соответствия колонкам окно браузера (или контейнер) должно подстроить совокупность данных размеров.

Возьмем, например, простой трехколоночный дизайн, представленный на рис. 13.11. Как вы можете видеть на верхнем изображении, эти три колонки не располагаются рядом друг с другом. Рассмотрим, из-за чего возникла проблема.

○ **div-обертка.** Контейнер `div` с фиксированной шириной заключает в себя весь контент. Его ширина равна 760 пикселей, таким образом, ширина всех колонок не может быть в сумме больше, чем это значение.

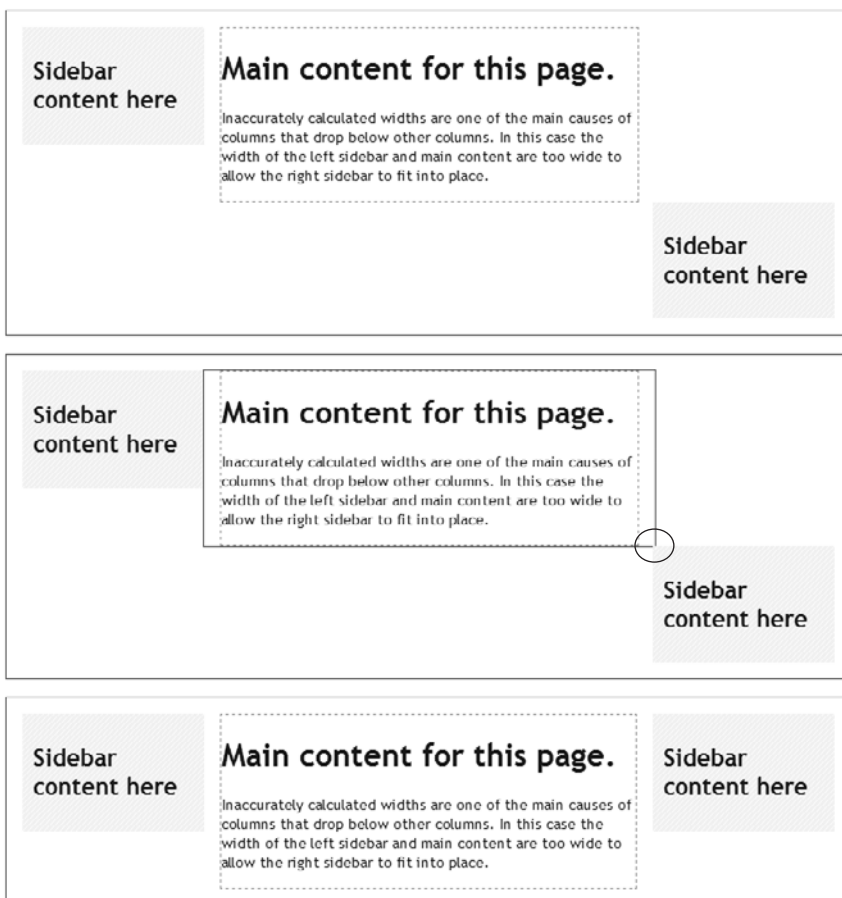


Рис. 13.11. Чтобы разрушить дизайн страницы, достаточно ошибки в 1–2 пиксела

- **Первая боковая панель (в левой части).** Ширина — 150 пикселей, но у панели также есть отступы по 10 пикселей, из-за чего общая ширина составляет 170 пикселей (150 пикселей панели + 10 пикселей отступа слева + 10 пикселей отступа справа).
- **Основной контент.** Имеет ширину 390 пикселей, а также включает по 1 пикселу обеих границ и по 15 пикселей полей слева и справа, из-за чего общая ширина равна 422 пикселям (390 пикселей ширины контента + 1 пиксел левой границы + 1 пиксел правой границы + 15 пикселей левого поля + 15 пикселей правого поля).
- **Вторая боковая панель (в правой части).** Ширина этого элемента равна 150 пикселям. Он также включает в себя по 10 пикселей левого и правого отступов; в результате получается 170 пикселей, как и в случае первой боковой панели.
- **Фактическая ширина всех добавленных элементов составляет 762 пиксела.** Это на 2 пиксела больше ширины div-обертки. На рис. 13.11, *посередине*, показана

рамка по краям контейнера `div` для основного контента, которая представляет его полную ширину *вместе* с полями. Всего двух лишних пикселей ширины (выделено на рисунке) достаточно для того, чтобы заставить колонку опуститься вниз. Решение заключается в удалении 2 пикселей из значения ширины любого элемента. К примеру, измените ширину левого и правого поля основного контента с 15 до 14 пикселей, что добавит дополнительное пространство, необходимое для размещения всех трех колонок, которые теперь будут находиться рядом.

Подведем итог: выпадение обтекаемых элементов вызывается недостатком пространства для вмещения всех колонок.

Способ упрощения вычислений заключается в удалении границ или отступов `div`-контейнеров или элементов, содержащих колонки. Тогда при установке трем колонкам значения ширины 20, 60 и 20 % соответственно вы будете знать, что они поместятся рядом друг с другом, поскольку вместе составляют 100 % и нарушающие это положение вещей отступы или границы отсутствуют. Если нужны дополнительные отступы, их можно добавить к элементам, находящимся внутри колонок, например присвоить одинаковые отступы слева и справа заголовкам, абзацам и другим элементам внутри `div`-контейнера. Это потребует дополнительной работы, но предотвратит потенциальное выпадение обтекаемых элементов, вызванное превышением суммарной ширины колонок в 100 %.

Если нужны еще и границы, можно применить прием вложения `div`-контейнеров:

```
<div class="column1">
  <div class="innerColumn">
    ... здесь размещен контент ...
  </div>
</div>
```

Затем устанавливается ширина внешнего `div`-контейнера, в данном случае имеющего класс `column1`, и добавляются отступы и границы к внутреннему `div`-контейнеру с классом `innerColumn`. Устанавливать ширину внутреннего `div`-контейнера не нужно, он заполнит всю ширину внешней колонки автоматически.

Предотвращение выпадения обтекаемого элемента с помощью свойства `box-sizing`

Основной причиной выпадения обтекаемых элементов является странный способ, который используется браузерами для вычисления фактической экранной ширины элемента. Например, вы устанавливаете ширину 100 пикселей, но браузер визуализирует элемент шириной 122 пиксела, потому что вы добавили также 10-пиксельный левый и правый отступы и 1-пиксельную границу по краям элемента. К счастью, есть CSS-свойство, которое позволяет избавиться от столь неприятного способа расчета.

Свойство `box-sizing` позволяет применить другую модель расчета фактической экранной ширины элемента. Ему можно присвоить одно из трех значений.

- `content-box` приводит к обычной работе браузера: ширина элемента учитывает только размер контента — значение свойства `width`.

```
box-sizing: content-box;
```

- `padding-box` позволяет включить левый и правый отступ в расчет ширины элемента. То есть экранная ширина элемента складывается из значения свойства `width`, а также левого и правого отступов. Любые границы по краям элемента не учитываются.

```
box-sizing: padding-box;
```

- `border-box` включает отступы, границы и значение свойства `width`. В общем, это то, что вам нужно. При задании этого значения расчеты упрощаются, что помогает избежать выпадения обтекаемых элементов, особенно при использовании ширины, заданной в процентном отношении в сочетании со значениями в пикселах ширины границ и отступов:

```
box-sizing: border-box;
```

Такую схему поддерживает большинство браузеров, включая Internet Explorer 8.

Некоторые веб-разработчики предлагают устанавливать для всех элементов значение `border-box`, чтобы они измерялись одинаково. Для этого используется универсальный селектор, который нужно поместить в верхней части таблицы стилей вместе с кодом сброса CSS:

```
* {  
  box-sizing: border-box;
```

Практикум: многоколоночные макеты

В этом практикуме мы рассмотрим, как создавать многоколоночные макеты, основанные на обтекаемых элементах. Вы создадите трехколоночный резиновый и фиксированный дизайны. Кроме того, вы научитесь применять ряд других методов достижения аналогичного результата.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 13.

Структурирование HTML-кода

Первый шаг в верстке макета на основе каскадных таблиц стилей — идентификация различных элементов на странице. Вы делаете это, обертывая фрагменты HTML-кода в контейнеры `div`, каждый из которых представляет отдельный элемент страницы.

1. Откройте файл `index.html` в редакторе HTML-кода и установите курсор на пустой строке после HTML-комментария: `<!-- первая боковая панель -->`.

Как вы можете видеть, часть HTML-разметки уже сделана: на данный момент созданы баннер и нижний колонтитул. Прежде чем создавать какие-либо стили, вы должны добавить структуру и контент на страницу. Далее вы добавите элемент `aside` для левой боковой панели.

2. Добавьте открывающий тег `<aside>` для левой боковой панели: `<aside class = "sidebar1">`. Затем нажмите клавишу **Enter**, чтобы перейти на новую строку.

В примере применяется HTML5-элемент `aside`, но для создания такой колонки можно воспользоваться и элементом `div`.

Если бы вы создавали веб-страницу с нуля, то в этом пункте пришлось бы добавлять HTML-код для боковой панели на странице и, возможно, определять список публикаций сайта, ссылки на родственные сайты и т. д. В данном случае вам не придется этого делать. Код неупорядоченного списка ссылок уже набран в отдельном файле. Осталось только скопировать его и добавить на страницу.

3. Откройте файл `sidebar1.txt`, скопируйте его содержимое, а затем вернитесь к файлу `index.html`. Вставьте скопированный HTML-код после тега `<aside>`, который вы создали в шаге 2 (или тега `<div>`, если вы решили использовать его).

Боковая панель почти готова. Осталось лишь закрыть элемент `aside`.

4. Сразу же после кода, который вы только что добавили, введите код `</aside>`. Вы только добавили на страницу первый элемент макета. Чуть позже мы отформатируем его так, чтобы он был похож на колонку. Но сначала вы должны добавить еще немного кода.

5. Установите курсор на пустую строку после следующего HTML-комментария: `<!-- основной контент -->`, а затем введите код `<article class = "main">`.

Этот раздел будет хранить главное содержимое страницы. Нужный HTML-код вы также возьмете в другом файле.

6. Откройте файл `main.txt`, скопируйте его содержимое, вернитесь к файлу `index.html` и вставьте скопированный код после тега `<article>`, который вы только что создали. Добавьте закрывающий тег `</article>` точно так же, как в шаге 4. Сохраните HTML-файл.

Это весь HTML-код, который нужен для создания дизайна. Теперь пришло время переключиться на создание каскадной таблицы стилей.

CSS-верстка

Если вы просмотрите страницу, то увидите, что для баннера, навигационных кнопок и текста стили уже созданы. Так получилось потому, что к странице присоединена внешняя таблица стилей с базовым форматированием. Далее нужно создать стили для форматирования колонок на странице.

1. Откройте в редакторе HTML-кода файл `styles.css`.

Поскольку веб-страница использует внешнюю таблицу стилей, новые стили будут добавляться в этот CSS-файл. Теперь вы работаете с двумя файлами: HTML и CSS, поэтому перед просмотром страницы в браузере нужно убедиться в том, что сохранены оба файла.

2. Перейдите в конец CSS-файла и найдите комментарий `/* стили из практикума главы 13 */`. Добавьте ниже этого комментария следующий код:

```
.sidebar1 {  
  float: left;  
  width: 20%;  
}
```

Стиль выравнивает боковую панель по левому краю страницы и задает ей значение ширины, равное 20 %. Свойство `width` играет в этом стиле важную роль: если только вы не выравниваете изображение, для которого задана ширина, всегда нужно ограничивать ширину обтекаемого (выравниваемого) элемента. В ином случае браузер установит ширину на основе контента внутри обтекаемого элемента, что приведет к противоречивым результатам. Здесь ширина задана в процентном отношении, стало быть, она определяется шириной данного контейнера. В нашем случае контейнером является элемент `body`, который заполняет всю ширину окна браузера. Поэтому экранная ширина боковой панели будет зависеть от ширины окна браузера, используемой посетителем.

3. Сохраните HTML- и CSS-файлы и просмотрите файл `index.html` в браузере.

Боковая панель теперь представляет собой колонку, выровненную по левому краю. Когда текст основной колонки достигает основания боковой панели, он обтекает основание боковой панели, как показано на рис. 13.12. Хотя это и типично для обтекаемых элементов, это не то, что нам нужно сейчас. Чтобы основной контент отобразился в виде отдельной колонки, следует добавить достаточное по размеру поле слева. Это позволит отделить основной контент от боковой панели.

4. Создайте стиль для второй колонки:

```
.main {  
  margin-left: 22%;  
}
```

Поскольку ширина боковой панели составляет 20 %, поле размером 22 % смещает основной контент на дополнительные 2 %, создавая промежуток между двумя колонками. Дополнительное пустое пространство не только повышает читабельность текста, но и улучшает внешний вид страницы.

Взгляните на страницу, и увидите, что получился двухколоночный дизайн.

Добавление колонки

Как вы могли заметить, двухколоночный дизайн создать несложно. Добавив третью колонку, вы сможете преподнести своим посетителям еще большее количество информации. Подобный дизайн также несложно создавать — действия в этом случае практически такие же, как в предыдущей части этого практикума.

1. Откройте файл `sidebar2.txt`. Скопируйте из него весь HTML-код, а затем вернитесь к файлу `index.html`.

HTML-код для этой колонки помещается после элемента `article` с основным контентом.

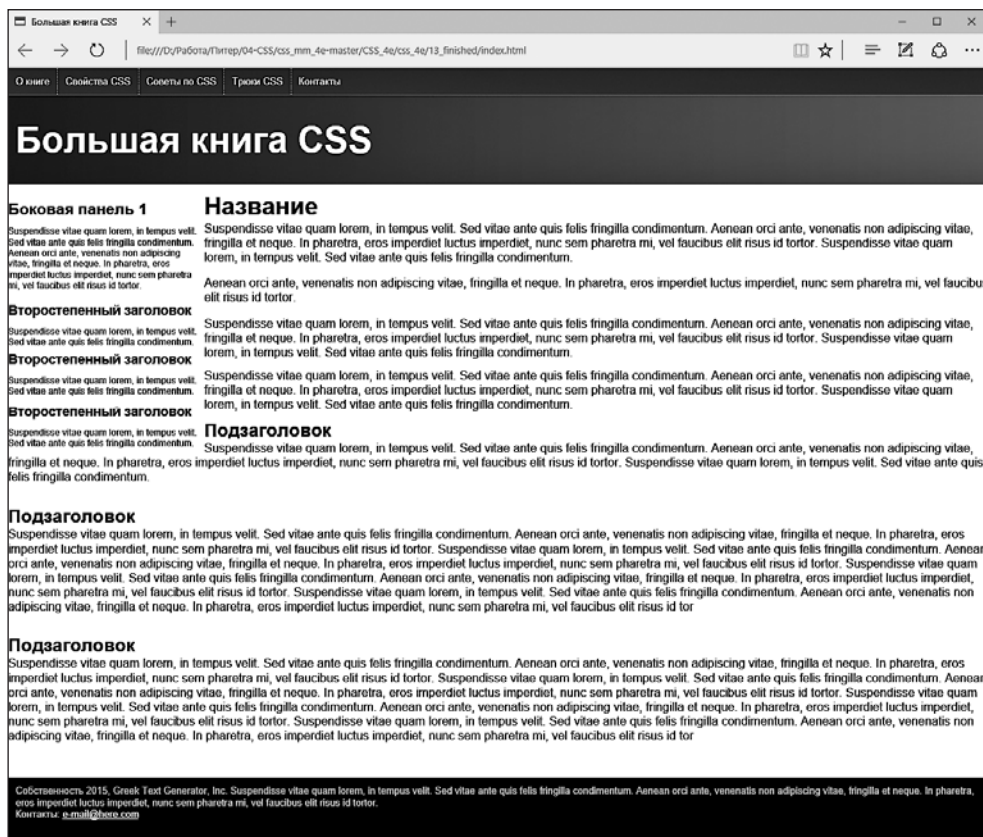


Рис. 13.12. Обтекаемый элемент в действительности не создает колонку на странице. Он лишь смещает любой контент, которое обтекает его, до позиции, где он заканчивается. После этого контент занимает свое место под элементом

- Найдите ближе к концу файла HTML-комментарий `<!--` вторая боковая панель `-->`. Щелкните кнопкой мыши на пустой строке ниже этого комментария. Зачастую, когда для структурирования контента страницы используется много `div`-контейнеров, найти нужный закрывающий тег `</div>` бывает нелегко. Вот почему HTML-комментарии, такие как этот, помогают определить HTML-код на странице.
- Введите код `<aside class="sidebar2">`, нажмите клавишу **Enter** и вставьте HTML-код, который вы скопировали в шаге 1. Вновь нажмите клавишу **Enter** и добавьте закрывающий тег `</aside>`. Сохраните HTML-файл. Закрыв элемент `div`, вы завершили HTML-код третьей колонки на странице. Теперь приступим к форматированию этой колонки.
- Вернитесь в редактор HTML-кода к файлу `styles.css`. Под стилем `.main`, который вы создали в шаге 4 в предыдущем подразделе, добавьте следующий код:


```
.sidebar2 {
  float: right;
  width: 20%;
}
```

Указав данный стиль, вы выровняете колонку по правому краю страницы, чтобы по обе стороны основного контента располагались боковые панели.

5. Сохраните все файлы и просмотрите файл `index.html` в браузере.

Теперь вы должны увидеть нечто странное. Вторая боковая панель отображается ниже основного контента и даже накладывается на нижний колонтитул. Проблема связана с порядком следования HTML-кода. Когда выравнивается элемент, то его обтекает и появляется после него только тот HTML-код, который *следует* в коде за этим обтекаемым элементом. То есть, пока HTML-код второй боковой панели следует за основным контентом, он появляется не рядом с ним, а после него.

Справиться со сложившейся ситуацией можно двумя способами. Можно переместить HTML-код второй боковой панели (второй элемент `aside`), указав его перед HTML-кодом основного контента (перед элементом `article`). Тогда первая боковая панель переместится влево, вторая — вправо, а основной контент поднимется и разместится между ними.

Можно выровнять основной контент. Если установить его ширину таким образом, чтобы ширина всех трех колонок не превышала 100 %, все они будут располагаться рядом. В текущем примере будет применен именно этот вариант.

6. Отредактируйте стиль `.main` следующим образом:

```
.main {
  float: left;
  width: 60%;
}
```

Если сохранить CSS-файл и просмотреть страницу `index.html` в браузере, проявится еще одна проблема — внезапно все снова будет испорчено! Спокойствие: это нижний колонтитул попытался обернуть все обтекаемые элементы и создал тем самым хаос. Как уже ранее говорилось, когда элемент выравнивается и его обтекает другой элемент, фоновый цвет и границы этого элемента фактически расширяются под обтекаемый элемент. Странно, все вроде правильно, а результат разочаровывает. Вполне очевидно, что для решения проблемы нужно, чтобы нижний колонтитул опустился ниже обтекаемых элементов.

7. После стиля `.sidebar2` добавьте следующий код:

```
footer {
  clear: both;
}
```

Как уже упоминалось, свойство `clear` может опустить элемент ниже колонок. В данном случае оно выталкивает нижний колонтитул ниже колонок (рис. 13.13).

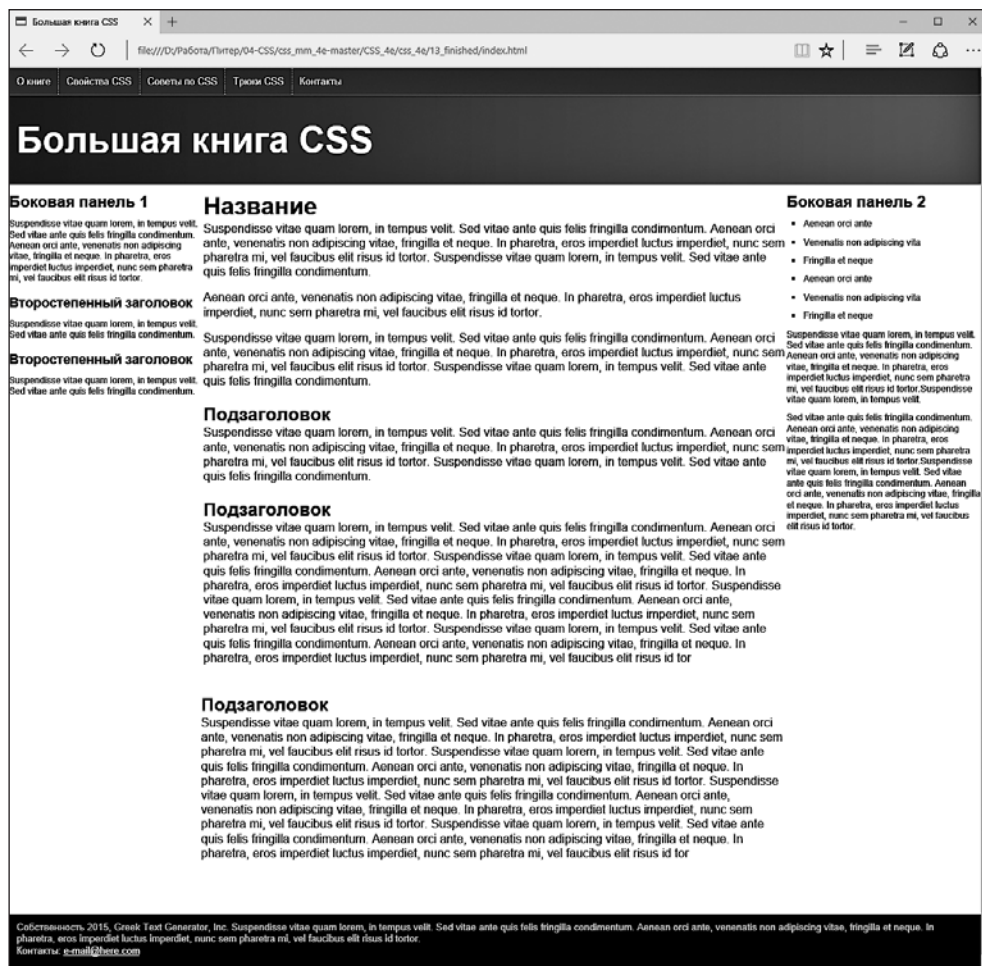


Рис. 13.13. Используя обтекаемые элементы, можно выстроить все три элемента рядом в три колонки

Добавление разрядки

Три колонки у вас уже есть, но текст выглядит слишком скученно. Эти три колонки практически касаются друг друга, и текст на правой боковой панели слишком близко прижимается к краю окна браузера. Исправить ситуацию помогут небольшие отступы.

1. Добавьте отступы в стилях `.sidebar1`, `.main` и `.sidebar2`, придав их коду следующий вид:

```
.sidebar1 {
  float: left;
  width: 20%;
  padding: 0 20px 0 10px;
```

```

}

.main {
  float: left;
  width: 60%;
  padding: 0 20px 0 20px;
}

.sidebar2 {
  float: right;
  width: 20%;
  padding: 0 10px 0 10px;
}

```

Здесь используется сокращенная запись свойства `padding`. Числа представляют верхний, правый, левый и нижний отступы соответственно. То есть в стиль `.sidebar1` добавлен верхний нулевой отступ, 20 пикселей отступа справа, нижний нулевой отступ и 10 пикселей отступа слева.

Если сохранить файл `styles.css` и просмотреть в браузере страницу `index.html`, станет заметна еще одна проблема — выпадение обтекаемого элемента. В результате добавления отступов каждая колонка стала шире, а поскольку общая ширина колонок $(20 + 60 + 20)$ % уже составила в сумме 100 %, дополнительные отступы опустили третью колонку под первые две. Это нужно срочно исправить!

Проблему можно решить несколькими способами. Во-первых, можно убрать отступы из этих стилей и добавить их ко всем внутренним элементам. То есть добавить 10 пикселей правого и левого отступов к элементам `h2`, `h3`, `p` и `ul`. Но это довольно трудоемкий процесс.

Во-вторых, можно убрать отступы из стилей в CSS-файле, а затем в документе `index.html` добавить в каждую колонку `div`-контейнер следующего вида:

```

<aside class="sidebar1">
  <div class="innerColumn">
    ... Сюда помещается контент ...
  </div>
</aside>

```

Затем в файле `styles.css` нужно создать стиль для добавления отступов:

```

.innerColumn {
  padding: 0 20px 0 10px;
}

```

Поскольку в стиле `.innerColumn` ширина не задается, контейнер просто разрастается, чтобы занять колонку, а отступы смещают все, что находится внутри него (заголовки, абзацы и т. д.), вовнутрь на 10 пикселей. Недостаток такого подхода состоит в необходимости добавления дополнительного кода.

Существует еще один, более простой и широко поддерживаемый браузерами способ.

2. Добавьте в верхней части таблицы стилей еще один стиль (сразу после комментария `/* сброс стилей браузера */`):

```
* {  
  box-sizing: border-box;  
}
```

В этом стиле используется универсальный селектор. Благодаря ему свойство `box-sizing` применяется к каждому элементу страницы. Присвоение этому свойству значения `border-box` инструктирует браузеры учитывать размер отступов и границ вместе со значением свойства `width`. То есть дополнительные отступы не добавляются к установленному ранее значению ширины. Тем самым предотвращаются любые выпадения обтекаемых элементов, поскольку колонки составляют не более 100 % ширины окна браузера.

И наконец, добавьте границы, чтобы отделить колонки друг от друга.

3. Отредактируйте стиль `.main`, добавив в него свойства левой и правой границ:

```
.main {  
  float: left;  
  width: 60%;  
  padding: 0 20px 0 20px;  
  border-left: dashed 1px rgb(153,153,153);  
  border-right: dashed 1px rgb(153,153,153);  
}
```

Эти свойства приведут к добавлению прямых линий с каждой стороны раздела основного контента. Если теперь просмотреть страницу в браузере, она должна выглядеть, как показано на рис. 13.14.

Ограничение ширины

В настоящее время у страницы резиновый дизайн, означающий, что контент расширяется, чтобы заполнить всю ширину окна браузера. Но, допустим, вам нужно, чтобы страница все время оставалась одной и той же ширины, так как вас не устраивает, как она выглядит на широкоформатных мониторах или при очень маленьком размере окна браузера. Изменить резиновый дизайн на фиксированный легко. Начните с добавления HTML-кода.

1. Вернитесь в редактор HTML-кода к файлу `index.html`. Сразу же за открывающим тегом `<body>` добавьте новый элемент `div`:

```
<div class="pageWrapper">
```

Таким образом вы заключите всю страницу в блок, который будет использоваться для управления шириной страницы. Вы должны убедиться, что этот контейнер закрыт.

2. Добавьте закрывающий тег `</div>` перед `</body>`:

```
</div>  
</body>
```

Теперь, когда созданный контейнер включает в себя все содержимое страницы, вы можете управлять ее шириной, изменяя ширину данного контейнера.

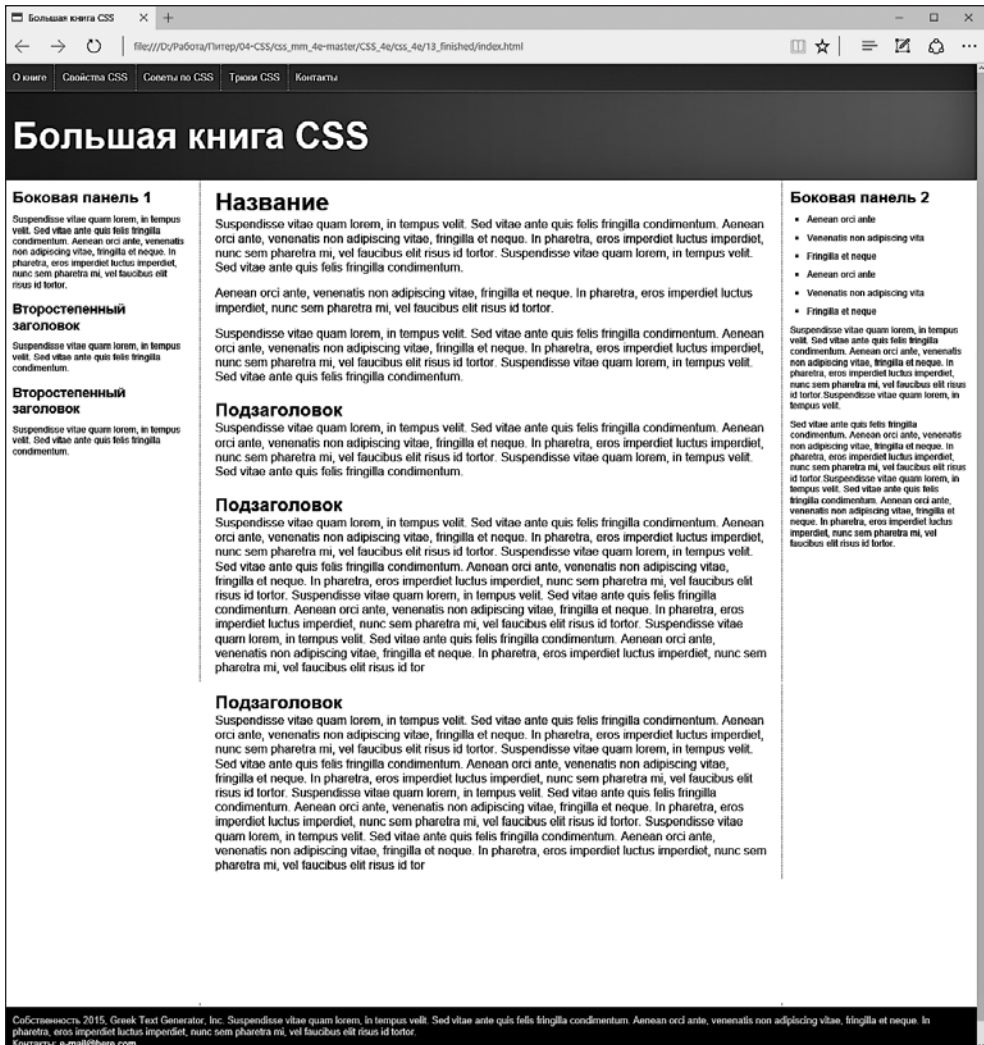


Рис. 13.14. Добавление отступов и границ визуально отделяют колонки друг от друга

- Сохраните HTML-файл и перейдите к редактированию файла `styles.css`. Добавьте еще один стиль:

```
.pageWrapper {
    width: 960px;
}
```

Если сохранить CSS- и HTML-файлы и просмотреть страницу `index.html` в браузере, вы увидите, что ее контент действительно заключен в пространство шириной 960 пикселей. Если сделать ширину окна браузера меньше 960 пикселей, появятся полосы прокрутки.

Но вам, разумеется, не нужно устанавливать точную ширину. Если хотите, чтобы страница поместилась в более узком (скажем, имеющем ширину 760 пикселей) окне браузера, лучше избегать ограничения ширины. Настоящая проблема заключается в том, что страницу становится трудно читать в слишком широком окне браузера. Другой подход заключается в использовании свойства `max-width`, которое не позволяет `div`-контейнеру увеличиваться свыше определенной величины, но не препятствует более узкому значению его ширины для помещения на небольших экранах. Раз уж вы за это взялись, нужно также центрировать `div`-контейнер в окне браузера.

4. Измените только что созданный в файле `styles.css` стиль `.pageWrapper`, чтобы он приобрел следующий вид:

```
.pageWrapper {
  width: 960px;
  margin: 0 auto;
}
```

Свойство `max-width` присуще резиновому макету, но только до конкретного предела. В данном случае, когда окно браузера шире 1200 пикселей, `div`-контейнер не будет увеличиваться. Свойству `margin` присвоено значение `0 auto`, обеспечивающее нулевые поля сверху и снизу и автоматические поля слева и справа. Последняя настройка (`auto`) позволяет браузеру автоматически определять размер полей путем равномерного деления пространства между левой и правой сторонами и центрируя таким образом `div`-контейнер в окне браузера. Теперь страница должна приобрести вид, показанный на рис. 13.15.



Рис. 13.15. Использование свойства `max-width` вместо `width` позволяет получить резиновый дизайн, помещающийся в окнах браузеров различной ширины, но при этом не приобретающий излишнюю ширину, затрудняющую чтение на широкоформатных мониторах с высоким разрешением

Резиновый/фиксированный макет

Страница выглядит вполне привлекательно, но могла бы выглядеть лучше, если бы черный фон для навигационной панели вверху и для колонтитула внизу, а также темно-фиолетовый градиент в баннере распространились на ширину всей страницы (рис. 13.16). Поскольку навигационная панель, баннер и нижний колонтитул находятся внутри `div`-контейнера `pageWrapper`, эти фоновые цвета прекращают отображаться, когда окно браузера становится шире 1200 пикселей. Вместо этого нужно оставить часть страницы — элементы с фоновым цветом — резиновой, при этом ограничив ширину элементов с основным контентом.

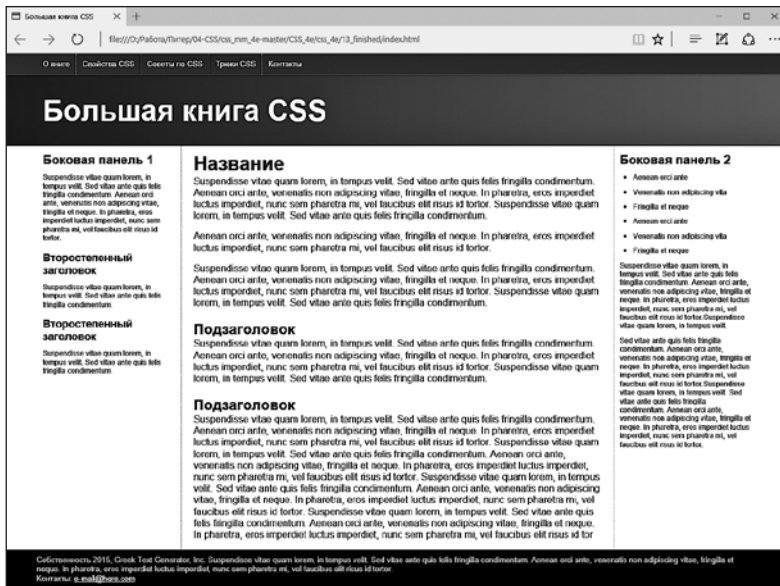


Рис. 13.16. Верстка многоколоночных макетов ничуть не сложнее создания обтекаемых HTML-элементов: три колонки, три обтекаемых элемента. Дополнительные колонки можно добавлять с помощью дополнительных HTML-контейнеров, выравнивая их по левому или по правому краю

Фоновый цвет для навигационной панели применяется к элементу `nav`, фиолетовый градиент — к элементу `header`, черный фон в нижнем колонтитуле — к элементу `footer`. Чтобы фон каждого из этих элементов распространился на всю ширину страницы, они не должны быть ограничены `div`-контейнером `pageWrapper`. Поэтому сначала нужно удалить этот стиль.

1. Удалите в файле `styles.css` недавно созданный стиль `.pageWrapper`.

Контейнер `pageWrapper` в HTML-коде можно оставить. Этот дополнительный HTML-код не станет обузой и может понадобиться вам позже для применения других стилей.

Страница вернется к своему полностью резиновому дизайну. Теперь нужно ограничить только навигационную панель, текст баннера, текст нижнего колонтитула

и основной контент, чтобы они не превышали по ширине 1200 пикселей. Для этого нужно немного углубиться в HTML-код и посмотреть, с какими элементами следует поработать.

Взгляните на HTML-код файла `index.html` и найдите элемент `nav`. Внутри этого элемента вы увидите кнопки навигации, созданные с помощью простого неупорядоченного списка. Именно это нам и нужно. Для списка можно установить максимальную ширину 1200 пикселей и центрировать его на странице, позволив элементу `nav` (и его черному фону) распространиться на всю ширину окна браузера. То же самое касается текста баннера внутри элемента `h1`. Для него также можно установить максимальную ширину и поля. В нижнем колонтитуле можно взять под контроль элемент `p`.

2. Добавьте в файл `styles.css` следующий код:

```
nav ul, header h1, footer p {
    max-width: 1200px;
    margin: 0 auto;
}
```

Этот групповой селектор нацелен на панель навигации, баннер и нижний колонтитул, но не на те элементы, в которых они содержатся. Теперь, если сохранить CSS-файл и просмотреть в браузере страницу `index.html`, вы увидите что элементы навигации, название и нижний колонтитул не становятся шире 1200 пикселей. Но основной контент по-прежнему заполняет все пространство окна браузера. Чтобы исправить ситуацию, нужно заключить три колонки в еще один `div`-контейнер для создания группы, размером и выравниванием которой можно управлять.

ПРИМЕЧАНИЕ

Другой способ заключается в том, чтобы добавить контейнер `div` в элемент `header` и обернуть им элементы `nav` и `h1`, а второй контейнер `div` вставить в элемент `footer`. Можно не ограничивать ширину элементов `header` и `footer`, ограничивая при этом ширину вложенных `div`-контейнеров.

3. Откройте в редакторе HTML-кода файл `index.html`. Непосредственно перед комментарием `<!-- первая боковая панель -->` добавьте код `<div class="contentWrapper">`:

```
<div class="contentWrapper">
<!-- первая боковая панель -->
```

Теперь этот `div`-контейнер нужно закрыть.

4. Перейдите в конец HTML-файла. Между закрывающим тегом `</aside>` и открывающим тегом `<footer>` добавьте закрывающий тег `</div>`, чтобы HTML-код приобрел следующий вид:

```
</aside>
</div>
<footer>
```

И наконец, можно добавить это новое имя класса к стилю, созданному на шаге 2.

5. Добавьте новый класс к групповому селектору в файле `styles.css`:

```
nav ul, header h1, footer p, .contentWrapper {  
    max-width: 1200px;  
    margin: 0 auto;  
}
```

Вид завершенной страницы показан на рис. 13.16. Полная версия этого урока находится в папке `13_finished`.

14

Позиционирование элементов на странице

Когда Консорциум W3C представил концепцию *CSS-позиционирования*, некоторые разработчики подумали, что они смогут делать так, чтобы веб-страницы выглядели как печатные документы, созданные в компьютерных текстовых редакторах. Благодаря лишь нескольким свойствам CSS-позиционирование позволяет поместить элемент в определенной позиции на странице, скажем в 100 пикселях от верхнего края страницы и 200 пикселях от левого края. Казалось, что точное пиксельное размещение обещало, что вы наконец-то сможете проектировать страницу, просто помещая фотографию в одной позиции, заголовок — в другой и т. д.

К сожалению, те возможности контроля, которые разработчики ожидали от CSS-позиционирования, так и не были реализованы. Разные браузеры всегда по-разному отображали элементы, позиционированные с помощью каскадных таблиц стилей. Но, что еще более существенно, Всемирная паутина функционирует не так, как печатная брошюра, журнал или книга. Веб-страницы намного более изменчивы, чем печатные. Как только журнал тиражируется в издательстве, читатели не могут изменить размер страницы или шрифта.

Посетители сайтов, с другой стороны, могут переделать вашу работу. Они могут увеличить размер шрифта в своем браузере, что, возможно, приведет к тому, что текст выйдет за пределы точно позиционированных элементов макета, которым заданы определенные размеры. Кроме того, поскольку в наше время для веб-серфинга люди используют смартфоны, планшеты и даже телевизоры, нельзя заранее предсказать размеры экранов, чтобы точно определить позиции *каждого* элемента на странице. Но все не так плохо: пока вы не пытаетесь навязать определенные ширину, высоту и позицию каждого элемента, свойства CSS-позиционирования будут для вас полезны. Вы можете использовать их, чтобы поместить подпись над фотографией, логотип в верхней части страницы и т. д.

Принципы работы свойств позиционирования

Свойство `position` каскадных таблиц стилей позволяет управлять, *где и как* браузер отобразит определенные элементы. Используя его, вы можете, например, поме-

стить боковую панель на странице там, где вы этого желаете, или убедиться, что навигационная панель наверху страницы останется на своем месте, даже если пользователи прокручивают страницу вниз. Каскадные таблицы стилей предлагают четыре типа позиционирования.

- **Абсолютное.** Такое позиционирование позволяет устанавливать расположение элемента, задавая позиции `left`, `right`, `top` или `bottom` в пикселах, единицах `em` или процентах (для получения дополнительной информации о выборе между различными единицами измерения см. главу 6). Вы можете поместить блок на расстоянии 20 пикселей от верхнего и 200 пикселей от левого края страницы, как показано на рис. 14.1, *посередине* (далее вы узнаете, как писать код с этими инструкциями).

Кроме того, абсолютно позиционированные элементы полностью отделены от потока страницы, определенного HTML-кодом. Другими словами, остальные элементы на странице не подозревают, что существует абсолютно позиционированный элемент. По вашей невнимательности они могут даже полностью исчезнуть, попав под него.

ПРИМЕЧАНИЕ

Не пытайтесь применять одновременно свойство `float` и любой тип позиционирования, кроме статичного (рассмотрен ниже). Выравниваемые объекты и позиционирование (абсолютное или фиксированное) не могут применяться к одному и тому же элементу.

- **Относительное.** Элемент с таким позиционированием позиционируется относительно его текущего положения в HTML-потоке. Так, например, присваивая свойству `top` значение 20 пикселей и `left` — значение 200 пикселей для относительно позиционированного заголовка, вы переместите его на 20 пикселей вниз и 200 пикселей влево *от той позиции, где он появился бы на странице*.

В отличие от абсолютного позиционирования, здесь остальные элементы страницы регулируют старое HTML-размещение относительно позиционированного объекта. Соответственно, перемещение объекта с относительным позиционированием оставляет «дыру», на месте которой он должен был находиться. Посмотрите на темную полосу на рис. 14.1, *внизу*. Это то место, где *появился бы* относительно позиционированный блок, если бы ему не было дано указание на перемещение. Основная польза относительного позиционирования не в том, чтобы переместить элемент, а в установке новой точки привязки для абсолютно позиционированных элементов, которые вложены в него (больше об этой концепции вы узнаете в подразделе «Когда абсолютное позиционирование относительно» далее).

- **Фиксированное.** Фиксированный элемент блокируется в определенной позиции на экране. Определение такого позиционирования играет ту же роль, что и присвоение значения `fixed` свойству `background-attachment` (см. раздел «Позиционирование фоновых изображений» главы 8). Когда посетитель прокручивает страницу, фиксированные элементы остаются на экране, например абзацы и заголовки, в то время как фотографии прокручиваются вместе со страницей. Использование фиксированных элементов — отличный способ создать неподвижную панель навигации в верхней или нижней части окна браузера. Вы скоро узнаете, как создается такой эффект.

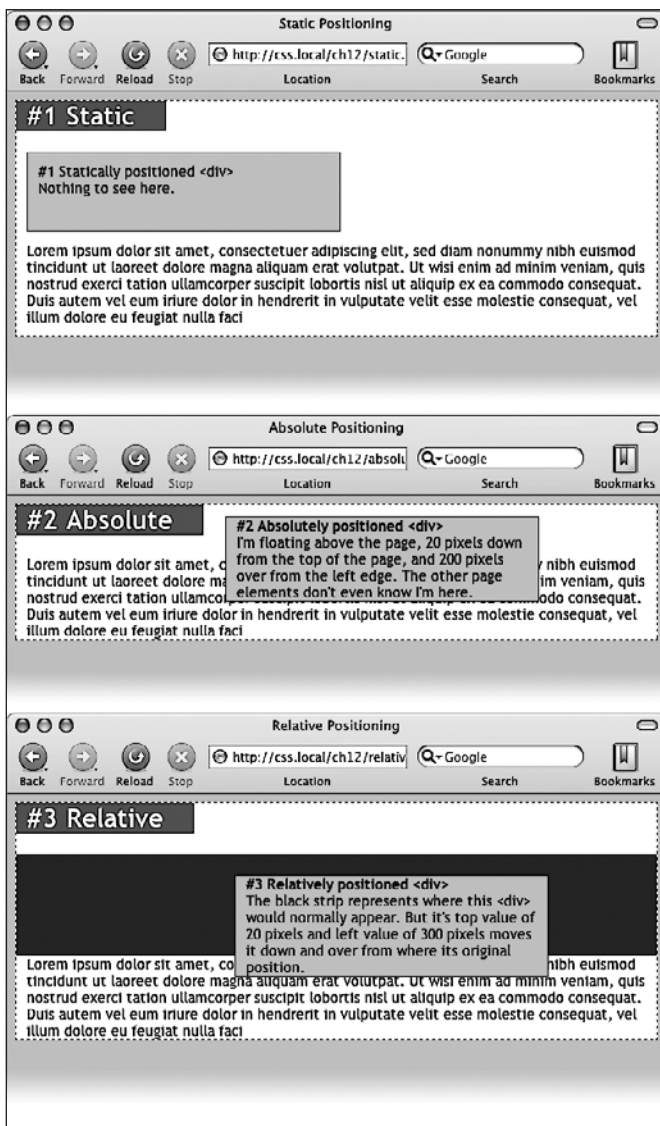


Рис. 14.1. Каскадные таблицы стилей предлагают несколько путей позиционирования на веб-странице

- **Статичное позиционирование** просто означает, что контент следует нормально нисходящему потоку HTML-элементов (см. рис. 14.1, *вверху*). Но зачем тогда назначать элементу статичное позиционирование? Скорее всего, вы никогда не будете делать этого.

Итак, можно сделать выводы. Статичное позиционирование — это то, как браузеры представляли содержимое с начала существования Всемирной паутины. Они просто отображали HTML-элементы в нисходящем порядке (сверху вниз). Абсо-

лютное позиционирование удаляет элемент из потока страницы, перемещая его в верхнюю часть страницы, иногда перекрывая какой-нибудь другой контент. Относительное позиционирование помещает элемент относительно той позиции, где он появился бы на странице, и оставляет «дыру» там, где этот элемент находился бы без относительного позиционирования.

Чтобы изменить позиционирование любого элемента, используйте свойство `position`, которому присваивается одно из этих четырех значений: `static`, `absolute`, `relative` или `fixed`. Чтобы создать абсолютно позиционированный элемент, добавьте следующее свойство в стиль:

```
position: absolute;
```

Статичное позиционирование — это обычный метод расположения элементов, так что, если только вы не отменяете ранее созданный стиль, которым уже назначается абсолютное, относительное или фиксированное позиционирование, вам не нужно указывать значение `static`. Кроме того, статичные элементы не подчиняются ни одному из значений позиционирования, как обсуждается далее.

Присвоение значений позиционирования — это обычно только часть битвы. Чтобы на самом деле расположить элемент где-нибудь на странице, вы должны разобраться с различными свойствами позиционирования.

Настройка позиций

Область отображения браузера, также называемая областью просмотра, имеет верхний, нижний, правый и левый края. Для каждого из этих четырех краев существует соответствующее CSS-свойство: `top`, `bottom`, `left` и `right`. Вам не нужно задавать значения для всех четырех сторон. Обычно достаточно двух, чтобы расположить элемент на странице. Вы можете, к примеру, поместить элемент на расстоянии 10 em от левого края страницы и 20 em от верхнего.

Чтобы указать расстояние от края страницы до соответствующей стороны элемента, используйте любую из действующих в CSS единиц измерений: пиксели, единицы em, проценты и т. д. При позиционировании можно также использовать отрицательные значения, например `left: -10px;`, чтобы поместить элемент частично за пределы страницы (или другого элемента). Это приведет к появлению особого визуального эффекта, который будет описан в разделе «Эффективные стратегии позиционирования» этой главы.

После свойства `position` вы указываете одно свойство или более для сторон (`top`, `bottom`, `left` или `right`). Если вы хотите, чтобы элемент принял ширину меньше допустимой (например, чтобы сделать тонкую боковую панель), то можете также использовать свойство `width`. Чтобы поместить баннер страницы в определенной позиции относительно верхнего и левого краев окна, создайте следующий стиль:

```
.banner {  
  position: absolute;  
  left: 100px;  
  top: 50px;  
  width: 760px;  
}
```

Этот стиль расположит баннер так, как показано на рис. 14.2, *вверху*.

ПРИМЕЧАНИЕ

Если не указать значение позиции по вертикали (top или bottom), браузер помещает элемент в ту же позицию на странице по вертикали, где бы он располагался при отсутствии позиционирования. То же самое справедливо для настройки размещения по горизонтали (left или right). То есть, если установить для элемента абсолютное позиционирование, но не предоставить значений позиционирования top, right, bottom или left, браузер оставит элемент в той же позиции, но поверх другого контента.



Рис. 14.2. Польза абсолютного позиционирования состоит в возможности поместить элемент относительно правого края окна браузера (*посередине*). Если ширина окна изменится, расстояние между правым краем окна и элемента останется неизменным (*внизу*)

Рассмотрим другой пример: поместим элемент таким образом, чтобы он всегда оставался на определенном расстоянии от правой стороны браузера. Если вы ис-

пользуете свойство `right`, браузер измеряет расстояние от правого края окна до правого края элемента (см. рис. 14.2, *посередине*). Чтобы поместить баннер на расстоянии 100 пикселей от правого края окна, вы создадите тот же самый стиль, что и ранее, но только используя свойство `right` вместо `left`:

```
.banner {  
  position: absolute;  
  right: 100px;  
  top: 50px;  
  width: 760px;  
}
```

Поскольку рассчитываемая позиция основывается на правом крае окна браузера, регулирование его размера автоматически меняет расположение баннера, что вы можете видеть на рис. 14.2, *внизу*. Хотя баннер и смещается, расстояние между правым краем элемента и правым краем окна браузера остается неизменным.

Можно даже указать свойства для левой и правой позиций, а также для верхней и нижней и позволить браузеру определить ширину и высоту элемента. Скажем, вы хотите, чтобы центральный блок текста размещался на расстоянии 10 % от верхнего края окна браузера и 10 пикселей от левого и правого края. Чтобы позиционировать блок, вы можете использовать стиль с абсолютным позиционированием, с присвоением свойствам `top`, `left` и `right` значения 10 %. В окне браузера левый край блока начинается от левого края окна на расстоянии 10 % от ширины окна, а правый край находится на расстоянии 10 % от правого края (рис. 14.3, *вверху*). Точная ширина блока при этом зависит от ширины окна браузера. Более широкое окно увеличит блок; чем уже окно, тем меньше будет блок. Левая и правая позиции, однако, по-прежнему составляют 10 % от ширины окна браузера.

Свойства `width` и `height`, о которых вы узнали в главе 7, работают аналогично и для позиционированных элементов. Чтобы поместить серый блок размером 50 × 50 пикселей в верхний правый угол окна браузера, создайте следующий стиль:

```
.box {  
  position: absolute;  
  right: 0;  
  top: 0;  
  width: 50px;  
  height: 50px;  
  background-color: #333;  
}
```

Здесь следует вспомнить предостережение, приведенное в разделе «Изменение высоты и ширины» главы 7: будьте внимательны с указанием высоты элементов. Если вы не создаете какие-либо изображения с уже заданной высотой, то не можете знать, насколько высоким будет на странице конкретный элемент. Вы могли бы определить для боковой панели высоту 200 пикселей, но если вы добавите текст или изображения, которые увеличат боковую панель свыше 200 пикселей в высоту, то в конечном счете содержимое панели выйдет за ее пределы. Даже если вы уверены, что контент поместится, посетитель в любой момент может увеличить размер шрифта в своем браузере, сделав текст достаточно крупным, чтобы он мог «выпасть» из блока. К тому же, если вы ограничиваете ширину и высоту в стиле,

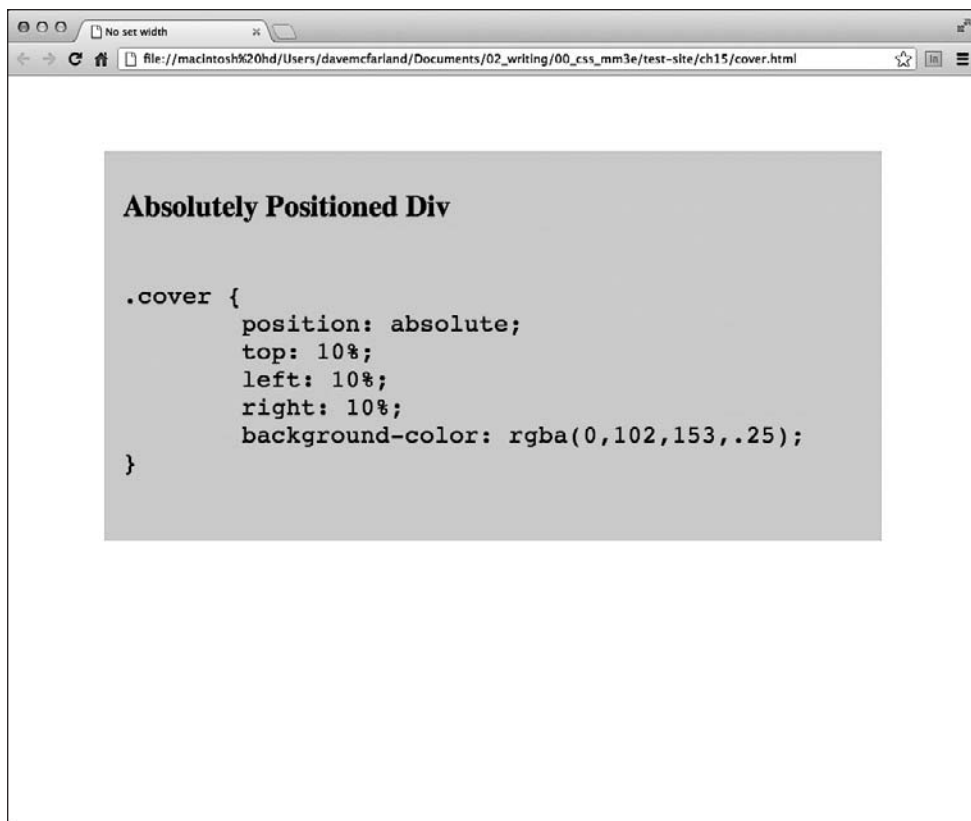


Рис. 14.3. Можно позволить браузеру определить ширину элемента, указав для элемента с абсолютным позиционированием значения свойств как `left`, так и `right`

а контент внутри разрабатываемого элемента оказывается шире или выше, могут быть достигнуты непредсказуемые результаты.

Когда абсолютное позиционирование относительно

Ранее в этой главе мы говорили о позиционировании элемента в конкретной позиции в окне браузера. Однако абсолютное позиционирование не всегда работает таким образом.

На самом деле абсолютно позиционированный элемент помещается *относительно* пределов его ближайшего позиционированного предка. Проще говоря, если вы уже создали элемент с абсолютным позиционированием (скажем, контейнер `div`, который появится на расстоянии 100 пикселей от верхнего края браузера), то любые абсолютно позиционированные HTML-элементы *внутри* этого контейнера `div` размещаются относительно верхнего, нижнего, левого и правого краев контейнера.

ПРИМЕЧАНИЕ

Если вы не помните, что такое родительские элементы и предки, см. раздел «Форматирование вложенных элементов» главы 3.

На верхнем изображении на рис. 14.4 светло-серый блок абсолютно позиционирован на расстоянии 5 em от верхнего и левого краев окна браузера.

Есть также элемент div, вложенный в этот блок. Абсолютное позиционирование этого элемента позволит поместить его *относительно абсолютно позиционированного родительского элемента*. Присвоение свойству bottom значения 0 переместит блок не к основанию экрана, а к основанию его предка. Аналогично свойство right этого вложенного элемента div относится к правому краю его родителя (см. рис. 14.4, *внизу*).

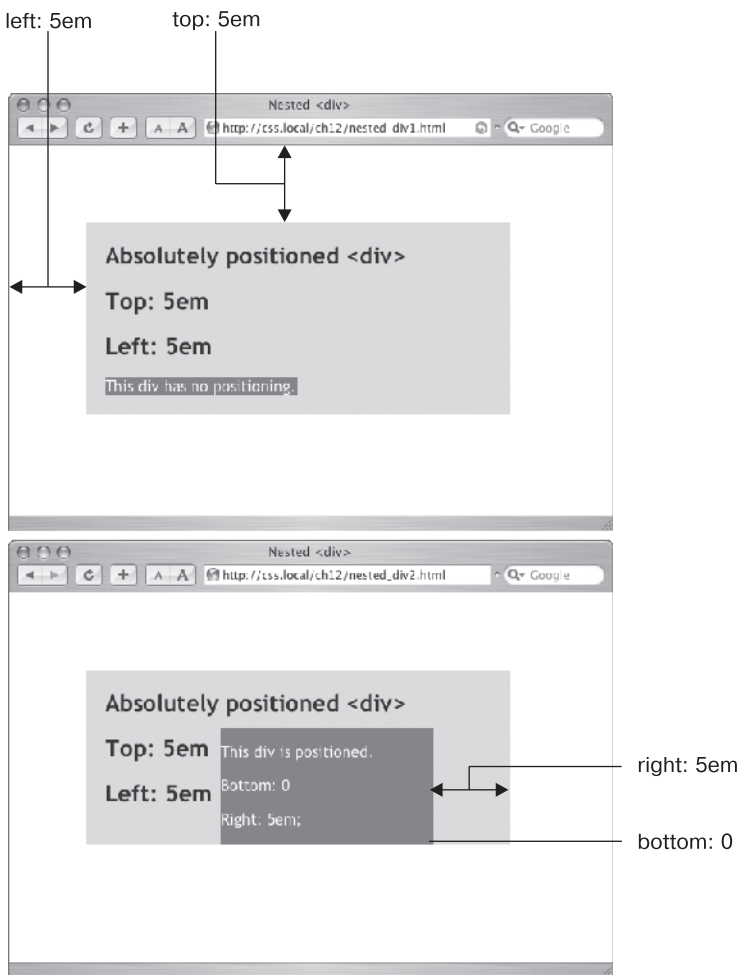


Рис. 14.4. Вы можете расположить элемент относительно окна браузера (вверху) или относительно позиционированного предка (внизу)

Каждый раз, когда вы используете абсолютное позиционирование, чтобы расположить элемент на странице, его точная позиция зависит от местонахождения любых других элементов, в которые вложен формируемый элемент. Позиционирование подчиняется следующим правилам.

- **Элемент позиционирован относительно окна браузера**, если у него абсолютное позиционирование и он не находится внутри любого другого элемента, к которому применено абсолютное, относительное или фиксированное позиционирование.
- **Элемент позиционирован относительно сторон другого элемента**, если он находится внутри другого элемента с абсолютным, относительным или фиксированным позиционированием.

Когда (и где) использовать относительное позиционирование

Вы получаете существенную выгоду, помещая элемент относительно другого элемента: если он смещается, позиционированный элемент перемещается вместе с ним. Скажем, вы поместили изображение внутрь элемента `h1` и хотите, чтобы оно появилось с правого края заголовка. Если вы просто поместите изображение в конкретной позиции в окне браузера, то рискуете потерять его из виду. Если заголовок будет смещен, абсолютно позиционированное изображение останется «приклеенным» к назначенной ему позиции. Вместо этого лучше определить изображение относительно элемента `h1` так, что, если заголовок будет перемещен, изображение переместится вместе с ним (два нижних изображения в нижней части рис. 14.5).

ПРИМЕЧАНИЕ

Используйте свойство `background-image` (см. раздел «Добавление фоновых изображений» главы 8), чтобы поместить изображение в качестве фона элемента `h1`. Однако если изображение будет выше элемента `h1` или вы хотите, чтобы оно появилось за пределами заголовка (см. третье изображение на рис. 14.5),стройте изображение с помощью элемента `img` и воспользуйтесь относительным позиционированием, рассматриваемым в данном разделе.

Чтобы поместить изображение на страницу, вы могли использовать значение `relative` свойства `position`, но здесь также есть недостатки. Если вы устанавливаете относительное позиционирование для элемента, а затем размещаете его (к примеру, используя свойства `left` и `top`), элемент перемещается на определенное расстояние от той позиции, где он появился бы в нормальном потоке HTML. Другими словами, он перемещается относительно своего текущего положения. В результате на его исходном месте остается пустое пространство, которое элемент занимал бы без настройки позиционирования (см. рис. 14.1, *внизу*).

Лучший способ использовать относительное позиционирование состоит в том, чтобы создать новую связь позиционирования для вложенных элементов. Например, элемент `h1` в примере в начале этого раздела является предком элемента `img`, который находится внутри него. При установке относительного позиционирования элемента `h1` любое абсолютное позиционирование, которое вы примените к элементу `img`, будет определено относительно четырех сторон заголовка `h1`, а не окна браузера. CSS-код выглядит следующим образом:

```
h1 { position: relative; }
h1 img {
  position: absolute;
  top: 0;
  right: 0;
}
```

Присвоение свойствам `top` и `right` изображения значения `0` перемещает его в верхний правый угол заголовка, а не окна браузера.

В CSS термин *относительный* означает не совсем то же самое, что в реальном мире. Все же, если вы хотите поместить элемент `img` относительно элемента `h1`, вашей первой мыслью может быть определение для изображения относительной позиции. На самом деле элемент, который вы хотите поместить, — изображение — получает абсолютное позиционирование, в то время как элемент, *относительно которого* вы хотите определить изображение, — заголовок — получает значение `relative`. Расшифруйте его как «относительно меня». Когда вы применяете относительное позиционирование к элементу, это означает, что «все элементы, заданные внутри него, должны размещаться относительно его местоположения».

ПРИМЕЧАНИЕ

Поскольку вы будете часто применять относительное позиционирование для задания новой связи расположения вложенных элементов, вам даже не нужно использовать значения `top`, `bottom`, `left` и `right`. У элемента `h1` есть свойство `position: relative`, но нет значений `top`, `bottom`, `left` и `right`.

Наложение элементов

Как вы можете видеть на рис. 14.6, абсолютно позиционированные элементы расположены на переднем плане веб-страницы и могут даже находиться поверх (или позади) других позиционированных элементов. Такое наложение определяется индексом позиционного уровня (`z-index`). Если вы знакомы с понятием стопок слоев в программе Photoshop, Sketch или Adobe Illustrator, то знаете, как работает индекс позиционного уровня: он представляет порядок, в котором элементы накладываются друг поверх друга на странице.

Чтобы сказать это другими словами, представьте веб-страницу как лист бумаги, а абсолютно позиционируемый элемент — как стикер, приклеиваемый поверх. Каждый раз, когда вы добавляете на страницу абсолютно позиционированный элемент, вы «приклеиваете» на нее стикер. Конечно, если вы делаете это, то рискуете перекрыть какой-либо контент на странице.

Обычно порядок наложения элементов следует их порядку в HTML-коде страницы. На странице с двумя абсолютно позиционированными элементами `div` второй HTML-элемент появится *поверх* другого элемента `div`. Однако вы можете управлять порядком, в котором накладываются элементы, используя свойство `z-index` каскадных таблиц стилей. В свойстве указывается числовое значение:

```
z-index: 3;
```

Чем больше значение, тем выше в стопке появится элемент. Скажем, у вас есть три абсолютно позиционированных изображения и все они частично перекрываются. Изображение, имеющее больший индекс позиционного уровня, появится

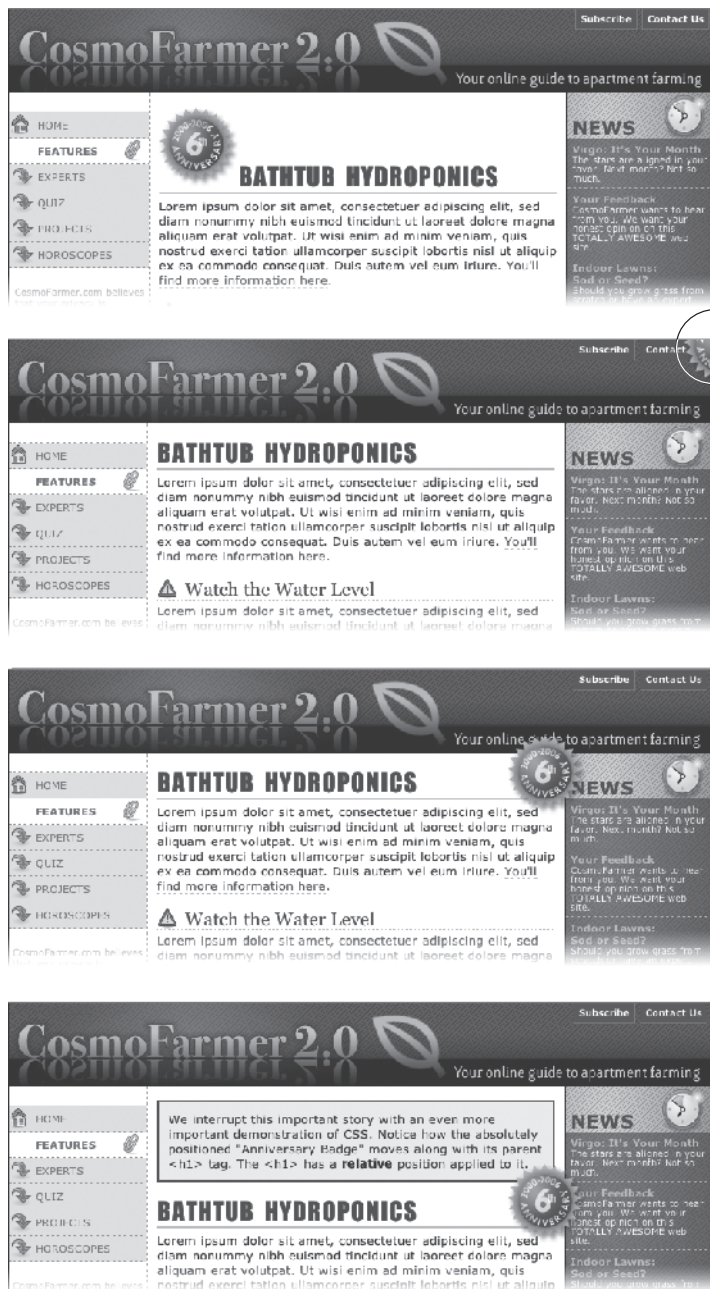


Рис. 14.5. *Вверху:* круглая графическая кнопка помещена внутрь элемента h1. *Второе сверху:* на втором изображении страницы к кнопке применено абсолютное позиционирование, которое смещает ее за пределы области элемента h1 и устанавливает в верхнем правом углу окна браузера. *Третье сверху:* в коде третьей веб-страницы к элементу h1 добавлено свойство position: relative, которое создает новую связь позиционирования для элемента img. *Внизу:* когда вы смещаете заголовков ниже, изображение также перемещается

поверх других (см. рис. 14.6, *вверху*). Когда вы меняете индекс позиционного уровня одного или нескольких изображений, вы изменяете порядок их наложения (см. рис. 14.6, *посередине*).

Свойству `z-index` можно присваивать и отрицательные значения, которые могут пригодиться, если нужно позиционировать элемент позади его родительского элемента или любого из его предков. Например, на рис. 14.6, *вверху*, элемент `div` имеет отрицательное позиционирование. Если нужно поместить одно из изображений позади `div`-контейнера, можно воспользоваться отрицательным значением свойства `z-index`: `z-index: -1`;

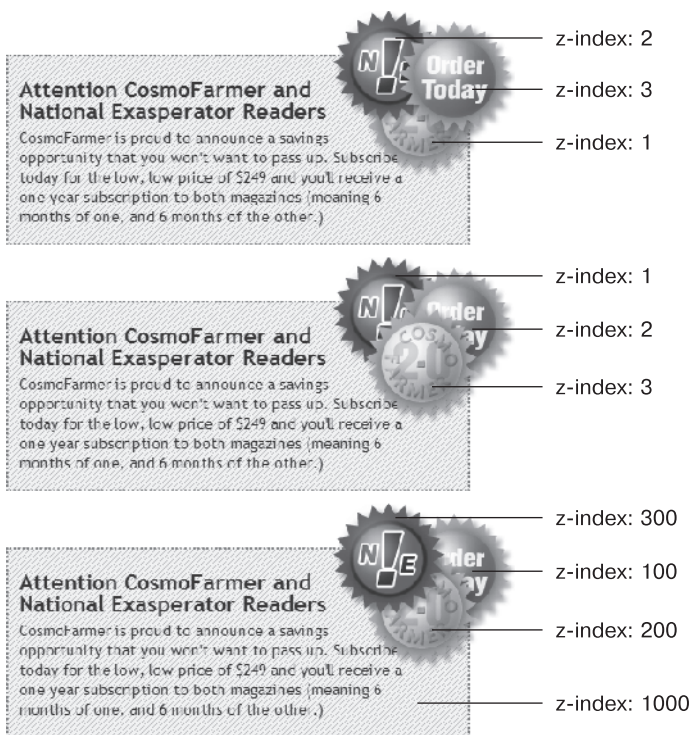


Рис. 14.6. Если вы используете индекс позиционного уровня, чтобы управлять порядком наложения элементов, родительские элементы определяют порядок наложения для своих детей

На рис. 14.6 в двух верхних примерах текстовый блок не позиционирован: он разделяет индекс позиционного уровня самой страницы непосредственно, для которой значение свойства `z-index` равно 0. Таким образом, кнопки в этих двух текстовых блоках накладываются поверх страницы. Однако для текстового блока в нижнем изображении задано абсолютное позиционирование со значением 1000 свойства `z-index`. Элемент `div`, содержащий текстовый блок, становится отправной точкой для укладки в него изображений. Поэтому, хотя свойству `z-index` контейнера `div` присвоено значение 1000, его вложенные дети (с более низкими значениями свойства `z-index`) появятся поверх, в то время как сам текстовый блок находится позади.

СОВЕТ

Промежутки в значениях индекса позиционного уровня вполне допустимы. Другими словами, значения 10, 20, 30 определяют то же самое, что и 1, 2, 3. В действительности, промежуток в числовых значениях допускает в дальнейшем добавление новых элементов. Если нужно обеспечить, чтобы поверх того или иного элемента больше ничего не появлялось, задайте ему очень большой индекс позиционного уровня, например `z-index: 10000`; но не переусердствуйте, поскольку максимальное значение, которое обрабатывают некоторые браузеры, равно 2147483647.

Соккрытие фрагментов страницы

Другое свойство каскадных таблиц стилей, часто используемое с абсолютно позиционированными элементами, — `visibility`. Оно позволяет скрыть часть страницы (или отобразить ее). Скажем, вы хотите, чтобы поверх изображения внезапно появлялось всплывающее сообщение, когда пользователь помещает поверх него указатель мыши. Или сделать подрисовочную подпись невидимой, когда страница загружается (`visibility: hidden`), и переходящей в режим видимости (`visibility: visible`), когда указатель находится поверх изображения.

Значение `hidden` свойства `visibility` подобно значению `none` свойства `display`, но между ними есть существенное различие. Если вы присваиваете свойству `display` элемента значение `none`, он буквально бесследно исчезает со страницы. А присвоение свойству `visibility` значения `hidden` предотвращает отображение браузером содержимого элемента, но оставляет пустое пространство в той позиции, где должен был быть элемент. При использовании с абсолютно позиционированными элементами, которые уже удаляются из потока страницы, свойства `visibility: hidden` и `display: none` ведут себя одинаково. Большинство разработчиков используют метод `display: none` и вообще не обращаются к свойству `visibility`.

Существует и другой способ сокращения элемента — присвоение его свойству непрозрачности `opacity` нулевого значения:

```
opacity: 0;
```

Чтобы элемент снова появился на экране, его свойству `opacity` можно вернуть значение 1:

```
opacity: 1;
```

Преимущество использования свойства `opacity` заключается в том, что браузер его может анимировать. Это означает, что вы можете использовать CSS-переходы, рассмотренные в главе 10, для анимации изменений уровня непрозрачности. Например, элемент можно сделать постепенно появляющимся, изменяя уровень его непрозрачности от 0 до 1 и добавляя переход.

Самый распространенный способ переключать режим элемента от скрытого к видимому и наоборот — через JavaScript-сценарий. Однако вам не нужно изучать программирование на JavaScript, чтобы использовать свойство `visibility` (или `display`). Вы можете добавить псевдокласс `:hover` (см. раздел «Псевдоклассы и псевдоэлементы» главы 3), чтобы сделать невидимый элемент видимым. Предположим, вам требуется поместить подпись поверх изображения, но важно, чтобы она появлялась только при нахождении указателя мыши над изображением (рис. 14.7). Для достижения результата выполните следующие действия.

1. Создайте HTML-код для изображения и подписи.

Один из способов предполагает использование HTML5-элементов `figure` и `figcaption`:

```
<figure class="hat">

  <figcaption>Изображение шляпы</figcaption>
</figure>
```

Здесь к элементу `figure` применяется класс `hat`, поэтому стиль можно применить только к данному изображению.

2. Позиционируйте подпись.

Для помещения подписи поверх изображения используется абсолютное позиционирование. Чтобы поместить подпись относительно элемента `img`, нужно установить для его родительского элемента `figure` относительное позиционирование, а также задать параметры ширины и высоты, соответствующие размерам фотографии.

```
.hat {
  position: relative;
  width: 100px;
  height: 100px;
}
.hat figcaption {
  position: absolute;
  bottom: 0;
  left: 0;
  right: 0;
  background-color: white;
}
```

Подпись помещается в нижнюю часть изображения (`bottom: 0`). За счет присвоения нулевых значений его свойствам `left` и `right` подпись займет всю ширину рисунка.

3. Скройте подпись.

При использовании кода выше браузер отобразит подпись поверх изображения, но вам нужно, чтобы это происходило только при нахождении указателя мыши над изображением. Чтобы скрыть подпись, добавьте к стилю `.hat figcaption` строку кода либо `visibility: hidden`, либо `display: none`.

```
.hat figcaption {
  display: none;
  position: absolute;
  bottom: 0;
  left: 0;
  right: 0;
  background-color: white;
}
```

4. Сделайте так, чтобы подпись появлялась, когда посетитель помещает указатель мыши поверх изображения.

Для этого воспользуйтесь псевдоклассом `:hover`. Нужно сделать подпись видимой при нахождении поверх изображения указателя мыши. К сожалению, такого стиля не существует, но, так как подпись находится внутри элемента `figure`, можно создать селектор потомков, влияющий на подпись при расположении указателя мыши над изображением:

```
.hat:hover figcaption {  
  display: block;  
}
```

Этот селектор потомков предписывает следующее: «Нужно нацелиться на любой элемент `figcaption`, который находится внутри элемента с классом `hat`, но только когда указатель мыши находится поверх элемента». Он работает только потому, что элемент `figcaption` является потомком элемента, поверх которого находится указатель мыши.

Такую идею можно использовать для создания основанных на CSS-коде всплывающих подсказок — дополнительной информации, появляющейся при помещении указателя мыши поверх ссылки. Для подробностей обратитесь к сайту tinyurl.com/nnmcjlo. Существует также множество JavaScript-сценариев: например, полноценным и простым в использовании JavaScript-сценарием для создания всплывающих подсказок является jQuery-модуль `aTip2` (tinyurl.com/pf2myco).

Как уже упоминалось, для сокрытия/отображения элемента можно также воспользоваться свойством `opacity`, а для анимации эффекта — свойством `transition`. Для этого можно изменить код ранее созданных стилей следующим образом:

```
.hat figcaption {  
  opacity: 0;  
  transition: opacity .5s ease-in;  
  position: absolute;  
  bottom: 0  
  left: 0;  
  right: 0;  
  background-color: white;  
}  
.hat:hover figcaption {  
  opacity: 1;  
}
```

Вы рассмотрите этот пример в действии в практикуме этой главы. Но следует помнить, что свойство `transition` не поддерживается браузером Internet Explorer 9 и более ранними версиями. Однако в данном случае ничего страшного не произойдет. Ваши посетители все равно увидят подпись, но только без ее плавного появления и исчезновения.

Эффективные стратегии позиционирования

Как говорилось в начале этой главы, если вы попытаетесь использовать CSS-позиционирование для размещения *каждого* элемента страницы, то можете столкнуться с проблемой. Поскольку просто невозможно предсказать все мыслимые комби-



Рис. 14.7. Подпись то видна, то не видна. Элемент с абсолютным позиционированием можно поместить поверх другого элемента, но скрыть его (*слева*), пока посетитель не наведет указатель мыши на родительский элемент (*справа*)

нации браузеров и параметров настройки, используемые вашими посетителями, позиционирование под управлением CSS работает лучше всего в качестве «тактического оружия». Применяйте его аккуратно, чтобы обеспечить точное размещение для определенных элементов.

В этом разделе вы узнаете, как использовать абсолютное позиционирование, чтобы добавлять маленькие, но важные детали к дизайну своей страницы, как абсолютно позиционировать определенные компоненты макета и закреплять важные элементы страницы в одном месте, в то время как остальной контент будет прокручиваться.

Позиционирование внутри элемента

Один из самых эффективных способов использования позиционирования состоит в том, чтобы размещать маленькие элементы относительно других объектов на странице. Абсолютное позиционирование похоже на выравнивание по правому краю, которое вы задаете обтекаемым элементам. На рис. 14.8, 1, дата на верхнем заголовке кажется высоко расположенной, но с помощью каскадных таблиц стилей вы можете переместить ее к правому краю нижнего заголовка.

Чтобы поместить дату отдельно от остальной части заголовка, вам нужно заключить ее в HTML-элемент. Добавление элемента `span` — распространенный способ применения класса к строке текста, формирующий ее отдельно от остальной части абзаца:

```
<h1><span class="date">Nov. 10, 2006</span> CosmoFarmer Bought By Google</h1>
```

Теперь возникает вопрос о создании стилей. Во-первых, вы должны присвоить элементу-контейнеру (здесь — `h1`) значение `relative` (относительное позиционирование). Затем применить абсолютное позиционирование (значение `absolute`) к элементу, который желаете разместить, — дате. Далее приведен CSS-код для нижнего изображения на рис. 14.8, 1:

```
h1 {
  position: relative;
  border-bottom: 1px dashed #999999;
}
h1 .date {
  position: absolute;
  bottom: 0;
  right: 0;
  font-size: .5em;
  background-color: #E9E9E9;
  color: black;
  padding: 2px 7px 0 7px;
}
```

Некоторые из приведенных выше свойств, например `border-bottom`, представлены для наглядности. Ключевые свойства выделены полужирным шрифтом: `position`, `bottom` и `right`. Как только вы задаете заголовку относительное позиционирование, можете поместить элемент `span`, содержащий дату, в нижний правый угол заголовка, присвоив свойствам `bottom` и `right` значение 0.

Исключение элемента за пределы блока

Вы можете также использовать позиционирование для размещения элемента таким образом, чтобы он «выглядывал» из-под другого элемента. На рис. 14.8, 2, верхнее изображение демонстрирует заголовок с графикой. Так, элемент `img` помещен внутрь элемента `h1` в качестве части заголовка. Использование абсолютного позиционирования и отрицательных значений свойств `top` и `left` позволяет переместить изображение к левому краю заголовка и вытеснить его за верхний и левый края. Рассмотрим CSS-код, который применяется в этом примере:

```
h1 {
  position: relative;
  margin-top: 35px;
  padding-left: 55px;
  border-bottom: 1px dashed #999999;
}
h1 img {
  position: absolute;
  top: -30px;
  left: -30px;
}
```

Основная концепция этого кода такая же, что и в предыдущем примере, но с некоторыми дополнениями. Во-первых, значения свойств `top` и `left` изображения отрицательные, поэтому изображение фактически появляется на расстоянии 30 пикселей над верхним краем заголовка и 30 пикселей левее от левого края заголовка. Будьте внимательны, когда используете отрицательные значения. В результате может получиться так, что элемент будет частично (или полностью) выпадать за пределы страницы или будет перекрывать содержимое другого элемента. Для предотвращения того, чтобы «отрицательно» позиционированный элемент выпадал за

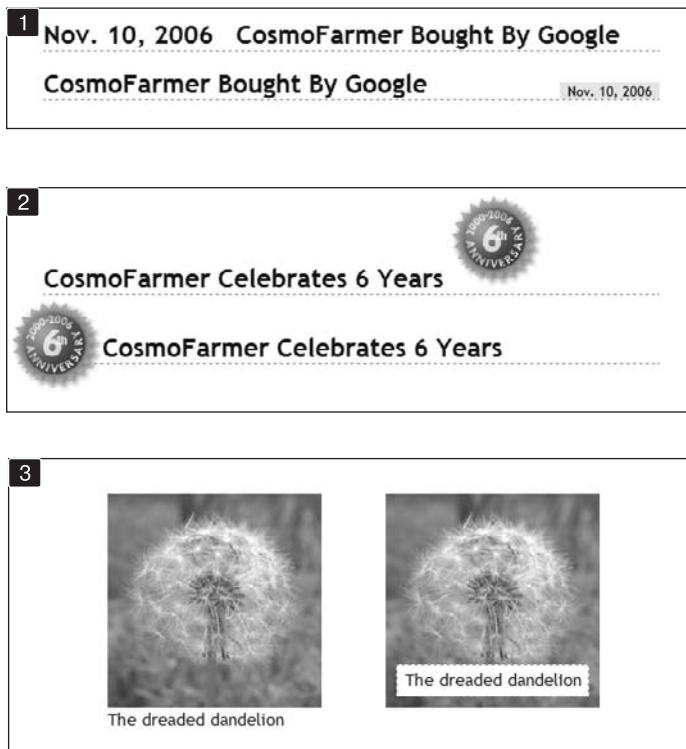


Рис. 14.8. Абсолютное позиционирование хорошо подходит для простых элементов дизайна, например для размещения даты в нижнем правом углу заголовка (*вверху*), вытеснения изображения из содержащего его блока (*посередине*) или наложения подписи поверх фотографии (*внизу*)

пределы окна браузера, добавьте поля или отступы необходимого размера либо к самому элементу, либо к содержащему его относительно позиционированному элементу — в данном примере `h1`. Дополнительные поля обеспечат достаточно пространства для размещения выпадающего изображения. В данном случае, чтобы предотвратить перекрытие изображением любого контента над заголовком, добавьте верхнее поле. Отступ слева шириной 55 пикселей также сместит текст заголовка из-под абсолютно позиционированного изображения.

Создание CSS-фреймов с фиксированным позиционированием

Поскольку многие веб-страницы длиннее одного экрана (прокрутки), может возникнуть необходимость сохранения на виду некоторых элементов страницы, например навигационной панели, поля поиска или логотипа сайта. HTML-фреймы были когда-то единственным способом «удержать» важные элементы, в то время как другое содержимое прокручивалось и исчезало из области видимости. Однако у HTML-фреймов есть существенные недостатки. Поскольку каждый фрейм

описывается в отдельном файле веб-страницы, приходилось создавать несколько HTML-файлов, чтобы сделать одну полноценную веб-страницу (называемую страницей с *набором фреймов*). Это не только отнимало много времени у разработчиков, но и усложняло поиск по сайту для поисковых машин. HTML-страницы, содержащие набор фреймов, также были неудобны посетителям с недостатками зрения и людям, использующим программы экранного доступа и желающим распечатать страницы сайта.

Тем не менее концепция фреймов все еще жизнеспособна, поэтому каскадные таблицы стилей предлагают вариант позиционирования, который позволит получить вид фреймов с меньшим количеством работы. Страница, созданная с использованием *фиксированного* позиционирования, показана на рис. 14.9. Присваивая значение `fixed` свойству `position`, можно имитировать HTML-фреймы, фиксируя некоторые элементы в желаемых позициях, допуская возможность прокрутки контента длинной веб-страницы. Полоса прокрутки (выделена на рисунке) позволяет прокручивать только большую текстовую область; верхний и нижний баннеры и боковая панель остаются неподвижными.

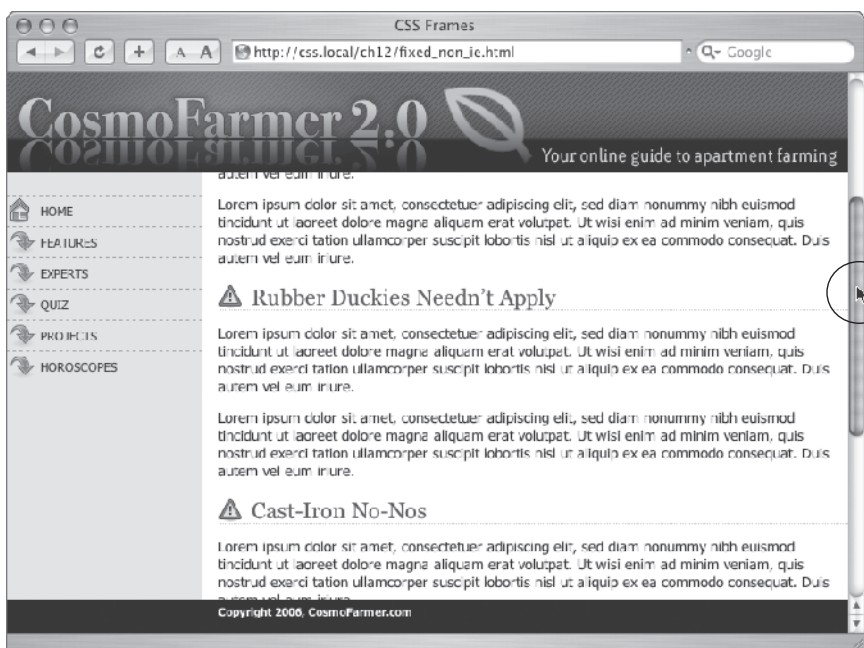


Рис. 14.9. Эффект старых страниц с фреймами, но с намного меньшим объемом кода

Фиксированное позиционирование действует во многом подобно абсолютному — вы точно так же можете использовать свойства `top`, `bottom`, `left` или `right` для размещения элемента. Как и абсолютное, фиксированное позиционирование удаляет элемент из потока HTML. Он «плавает» поверх прочего контента страницы, который игнорирует его.

Рассмотрим, как создать страницу, похожую на изображенную на рис. 14.9, у которой есть фиксированный баннер, боковая панель и нижний колонтитул, а также прокручиваемая область с основным контентом.

1. Добавьте элементы `div` с классами (или идентификаторами) к каждому разделу страницы.

У вас может быть четыре основных элемента `div` с такими классами (или идентификаторами), как `banner`, `sidebar`, `main` и `footer` (рис. 14.10). Порядок, в котором вы указываете эти элементы в HTML-коде, не имеет значения. Как и абсолютное, фиксированное позиционирование позволяет размещать элементы на странице независимо от их расположения в HTML-коде.

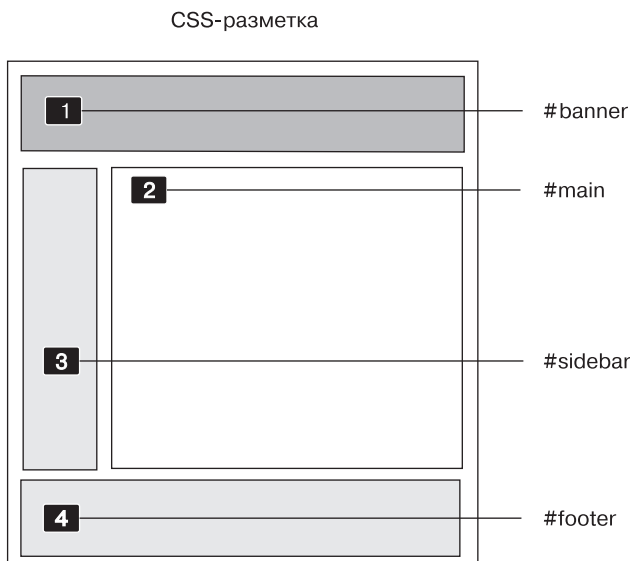


Рис. 14.10. С фиксированным позиционированием можно закрепить все элементы страницы, чтобы они всегда были в поле зрения при прокрутке. В этом примере заголовок (1), боковая панель (3) и нижний колонтитул (4) зафиксированы, а область с основным контентом (2) прокручивается

2. Добавьте контент к каждому контейнеру `div`.

В основном используйте фиксированные `div`-контейнеры для контента с необходимостью постоянного доступа в областях, которые вы желаете закрепить в конкретных позициях. В этом примере баннер, боковая панель и нижний колонтитул содержат логотип, глобальную навигацию по сайту и сведения об авторском праве. Основной контент помещается в оставшийся элемент `div`.

Важно отметить: не добавляйте слишком много контента к фиксированному элементу `div`. Если фиксированная боковая панель длиннее, чем окно браузера пользователя, посетитель не сможет увидеть боковую панель целиком. Поскольку фиксированные элементы не прокручиваются, у пользователя не будет никакой возможности (за исключением покупки монитора с большей

диагональю) увидеть содержимое боковой панели, которое не соответствует окну его браузера.

3. Создайте стили для всех фиксированных элементов.

Значения `top`, `bottom`, `left` и `right` задаются относительно окна браузера, таким образом, определите, где какие элементы вы хотите видеть на экране, и добавьте значения. Кроме того, определите ширину для элементов.

ПРИМЕЧАНИЕ

В отличие от абсолютного, фиксированное позиционирование всегда задается относительно окна браузера, даже если фиксирующийся элемент помещается внутрь другого элемента с относительным или абсолютным позиционированием.

Код стилей для позиционирования элементов *1*, *3* и *4* (на рис. 14.10) выглядит следующим образом:

```
.banner {
  position: fixed;
  left: 0;
  right: 0;
  top: 0;
}
.sidebar {
  position: fixed;
  left: 0;
  top: 110px;
  width: 175px;
}
.footer {
  position: fixed;
  bottom: 0;
  left: 0;
  right: 0;
}
```

4. Создайте стиль для прокручиваемой области контента.

Поскольку фиксированные элементы удаляются из потока HTML, прочие элементы на странице не учитывают данное обстоятельство. Так, элемент `div` с основным контентом страницы, к примеру, появляется под фиксированными элементами. Основная задача следующего стиля состоит в применении полей для свободного перемещения контента.

```
.main {
  margin-left: 190px;
  margin-top: 110px;
}
```

Фиксированное позиционирование прекрасно поддерживается в Internet Explorer 8 и последующих версиях, а также во всех других основных браузерах (включая самые актуальные версии для мобильных устройств под управлением операционных систем iOS и Android).

Практикум: позиционирование элементов страницы

Этот практикум позволит вам исследовать несколько различных способов применения абсолютного позиционирования, таких как создание трехколоночного макета, позиционирование элементов внутри баннера и добавление подписей поверх фотографий. В отличие от предыдущей главы, где вы заключали фрагменты HTML-кода в элементы `div` и добавляли к ним имена классов, в этих уроках большая часть работы с HTML-кодом уже была выполнена. Вы можете сосредоточиться на оттачивании ваших новых навыков с каскадными таблицами стилей.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 14.

Улучшение баннера страницы

Во-первых, сделаем несколько маленьких, но визуально важных изменений в баннере страницы. Создадим стили для HTML-элементов с примененными к ним классами (классы уже применены).

1. Запустите браузер и откройте файл `index.html` из папки 14.

На этой веб-странице (рис. 14.11) вы измените позиции некоторых частей баннера.

2. Откройте файл `styles.css` в редакторе HTML-кода.

Файл уже содержит основные стили форматирования. Далее вы добавите собственные стили в нижнюю часть файла.

Начните с перемещения небольшого изображения шляпы в левую сторону баннера. Чтобы избавиться от блочного вида, типичного для CSS-верстки, вытесните изображение за пределы баннера, таким образом сделав его похожим на стикер-арт.

3. В нижней части файла `styles.css` добавьте новый стиль:

```
header .badge {  
    position: absolute;  
    top: -20px;  
    left: -90px;  
}
```

Изображение находится внутри HTML5-элемента `header` с классом `badge`. Только что созданный стиль приводит к тому, что левый верхний угол изображения помещается на 90 пикселей влево и 20 пикселей над верхним краем страницы.

Теперь просмотрите страницу, и вы увидите, что на ней есть несколько проблем. Во-первых, изображение «нависает» над краем страницы, а вы хотите, чтобы она располагалась над краем области баннера. Займемся этой проблемой.

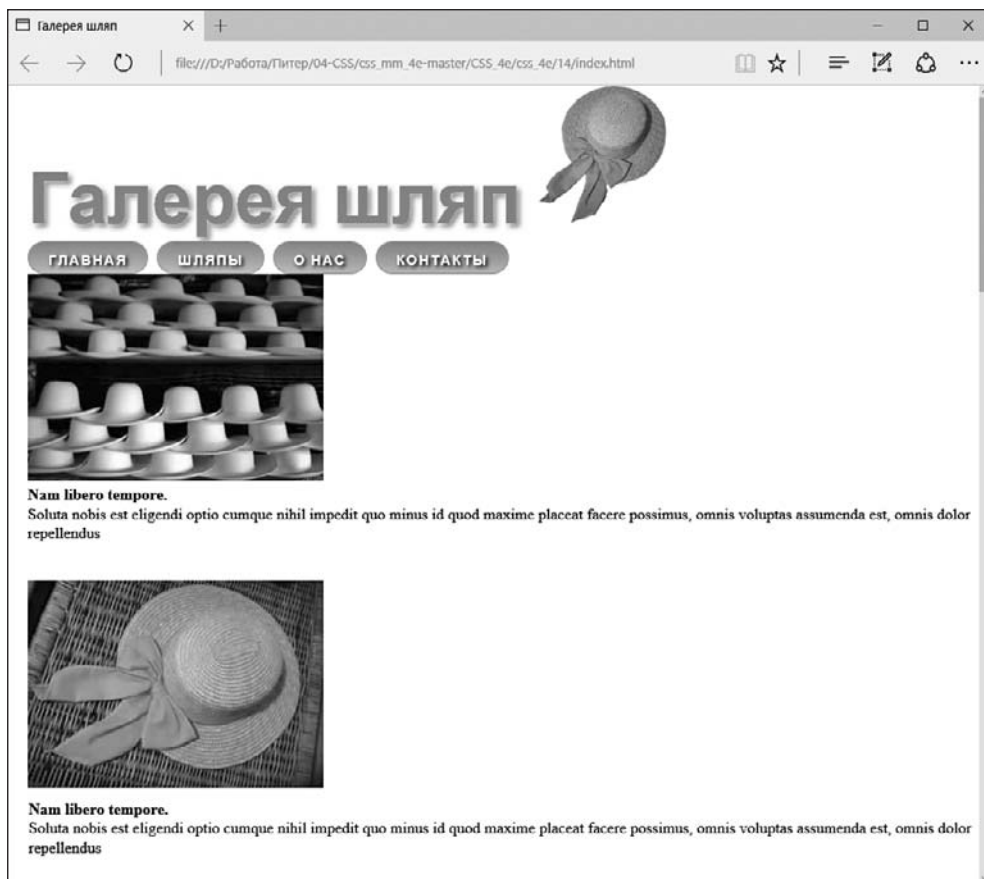


Рис. 14.11. На странице находится много элементов — логотип, баннер, панель навигации и галерея изображений с подписями, — при этом визуальная структура отсутствует. Это обычная статичная HTML-страница с потоком контента сверху вниз. Вы можете сделать ее более удобной для чтения и привлекательной, применив абсолютное позиционирование

4. Добавьте перед только что созданным стилем следующий код:

```
header {
  position: relative;
}
```

Код стилей, которые управляют основным разделом страницы (как этот стиль `header`), принято помещать выше кода стилей, формирующих фрагменты этого раздела (как стиль, который мы создали в шаге 3). Кроме того, группирование стилей для связанных разделов упрощает их поиск, когда нужно проанализировать или отредактировать CSS-код страницы. В этом случае стиль `header` указывается первым во внутренней таблице стилей, потому что применяется к большому фрагменту HTML-кода. Но вы должны указать стиль `header .badge` рядом с ним, так как добавляете дополнительные стили на страницу (подробнее

о способах организации CSS-кода на странице можно прочитать в разделе «Организация стилей» главы 18).

Стиль `header` создает новую связь позиционирования для любых вложенных элементов. Другими словами, значение `relative` функционирует так, что любые другие элементы внутри этого элемента позиционируются относительно краев баннера. Это изменение в позиционировании меняет размещение элемента со стилем, который вы создали в шаге 3. Теперь он смещен на 20 пикселей вверх и на 90 пикселей влево от области баннера. Изображение все еще немного «нависает» над страницей, поэтому нужно добавить небольшие поля для заголовка, чтобы немного его опустить.

5. Отредактируйте стиль `header`, добавив в его нижнюю часть две строки кода, выделенные полужирным шрифтом:

```
header {  
    position: relative;  
    margin-top: 20px;  
    padding: 20px 0 0 10px;  
}
```

Свойство `margin-top` добавляет пространство над элементом `header`, достаточное, чтобы он и изображение переместились вниз. Кроме того, отступ добавляет пространство внутри элемента заголовка, чтобы заголовок (и расположенная рядом навигационная панель) не смотрелись слишком скученно. Но теперь появилась другая проблема — заголовок частично скрыт под значком. Перекрытие элементов — один из недостатков абсолютного позиционирования. В данном случае проблему можно решить путем добавления свойства `z-index` изображения и помещения его позади текста.

6. Добавьте в стиль `header .badge` свойство `z-index: -1;`

```
header .badge {  
    position: absolute;  
    top: -20px;  
    left: -90px;  
    z-index: -1;  
}
```

Значение `-1` приводит к тому, что элемент с абсолютным позиционированием помещается позади своего родительского элемента, в данном случае позади текста (рис. 14.12). Затем абсолютным позиционированием нужно воспользоваться для смещения панели навигации в правую часть элемента `header`.

7. Добавьте после стиля `header .badge` следующий код:

```
header nav {  
    position: absolute;  
    right: 0;  
    top: 45px;  
}
```

Хотя задать позицию панели навигации можно за счет выравнивания элемента `h1`, в данном случае намного проще будет воспользоваться абсолютным

позиционированием. Здесь создается стиль для HTML-элемента `nav`, когда он находится в элементе `header`. Следует помнить, что в шаге 4 вы задали для элемента `header` относительную позицию, следовательно, любые элементы внутри, например `nav`, позиционируются относительно него. Поэтому нулевое значение, присвоенное свойству `right`, в этом стиле приводит к тому, что правый край панели навигации помещается по правому краю баннера (см. рис. 14.12).

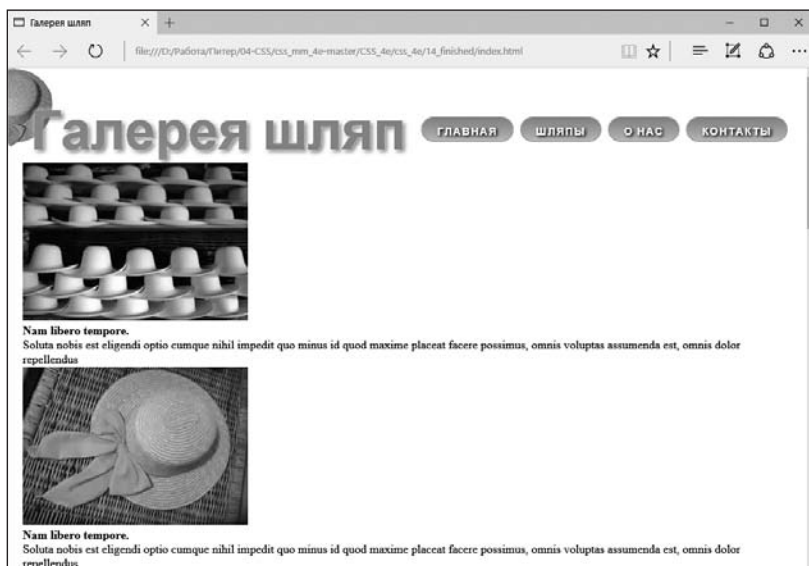


Рис. 14.12. Абсолютное позиционирование — хорошее подспорье в размещении небольших элементов вроде изображения шляпы и панели навигации. В отличие от обтекаемых элементов, точно указанная позиция изображения и навигационной панели в HTML-коде не имеет значения, предоставляя большую степень гибкости

Добавление подписей к иллюстрациям

В главе 8 мы рассмотрели один из способов добавления подписи к фотографии (см. практикум главы 8). В примерах той главы подписи располагались под фотографиями, чего мы и добиваемся в большинстве случаев. Но может понадобиться добавить подпись непосредственно *поверх* фотографии, как, например, субтитры в телевизионных новостях, которые отображаются в нижней части экрана.

1. Откройте файл `index.html` в редакторе HTML-кода.

На этой странице находится галерея фотографий. HTML-код одного из изображений имеет следующий вид:

```
<figure>
  
  <figcaption><strong>Nam libero tempore.</strong> Soluta nobis
    est eligendi optio cumque nihil impedit quo minus id quod
    maxime placeat facere possimus, omnis voluptas assumenda est.
```

```

    omnis dolor repellendus</figcaption>
  </figcaption>
</figure>

```

В этом примере используются HTML-элементы `figure` и `figcaption`. Элемент `figure` по умолчанию блочный, но, поскольку нужно, чтобы изображения располагались друг рядом с другом, начните с преобразования их в строчные элементы.

2. Ниже ранее добавленного стиля `header nav` укажите еще один стиль:

```

.gallery figure {
  display: inline-block;
  width: 300px;
  height: 210px;
  margin: 15px;
  position: relative;
}

```

Код создает селектор потомков, который применяется ко всем элементам `figure`, сгруппированным внутри контейнера `div` с классом `gallery`. Селектор потомков используется потому, что на эту страницу могут добавляться другие элементы `figure`, которые не являются частью галереи и должны быть отформатированы по-другому. Этот селектор потомков предназначен только для тех элементов `figure`, которые нас интересуют в данный момент.

После преобразования блочного элемента `figure` в строчный (`inline-block`) все изображения смогут находиться друг рядом с другом. Значения свойств `width` и `height` соответствуют ширине и высоте изображений. То есть элементы `figure` должны быть достаточного размера для вмещения изображений. Свойство `margin` добавляет небольшое пространство по периметру изображений, чтобы они не сталкивались друг с другом. И наконец, объявление `position: relative` устанавливает новую связь позиционирования, чтобы каждую подпись можно было позиционировать относительно связанного с ней изображения.

Теперь настал черед позиционирования подписей.

3. Ниже только что добавленного стиля разместите следующий код:

```

.gallery figcaption {
  position: absolute;
  top: 15%;
  bottom: 15%;
  left: 0;
  right: 0;
  background-color: rgba(153,153,153,.9);
}

```

Элементы `figcaption` получают абсолютное позиционирование с использованием всех четырех квадрантов позиционирования: `top`, `bottom`, `left` и `right`. По существу, подписи будут распространяться на все изображение, но помещаться немного ниже его верхнего края и немного выше нижнего края (фактически на 15 % ниже и выше). Использование всех четырех настроек означает, что вам не

нужно переживать за настройку ширины или высоты подписей: вместо этого вы оставляете ее на усмотрение браузера.

И наконец, указывается объявление фонового цвета `background-color` с установкой полупрозрачного фона поверх каждого изображения, что означает, что изображения можно видеть сквозь фон подписей.

Теперь займемся форматированием текста.

4. Отредактируйте только что созданный стиль, добавив код, выделенный полужирным шрифтом:

```
.gallery figcaption {  
  position: absolute;  
  top: 15%;  
  bottom: 15%;  
  left: 0;  
  right: 0;  
  background-color: rgba(153,153,153,.9);  
  padding: 20px;  
  font-family: Titillium, Arial, sans-serif;  
  font-weight: 400;  
  font-size: .9em;  
  color: white;  
}
```

Отступы создают небольшую разрядку для текста, а другие свойства задают шрифт, размер и цвет.

Если теперь посмотреть страницу, можно увидеть, что подписи отображаются поверх всех изображений. Далее нужно будет изменить стиль, чтобы подписи появлялись только при нахождении указателя мыши поверх изображения. Начнем с сокрытия подписей.

5. Добавьте в стиль код `opacity: 0;`:

```
.gallery figcaption {  
  position: absolute;  
  top: 15%;  
  bottom: 15%;  
  left: 0;  
  right: 0;  
  background-color: rgba(153,153,153,.9);  
  padding: 20px;  
  font-family: Titillium, Arial, sans-serif;  
  font-weight: 400;  
  font-size: .9em;  
  color: white;  
  opacity: 0;  
}
```

Присвоение свойству `opacity` значения 0 полностью скрывает подпись. Для сокрытия подписей можно также воспользоваться объявлением `display: none;` или `visibility: hidden;`, но выбранный способ позволяет анимировать значение не-

прозрачности с помощью CSS-перехода, и этот эффект будет вскоре добавлен. Но сначала нужно добавить состояние `:hover`, чтобы при помещении указателя мыши над изображением появлялась соответствующая подпись.

6. Добавьте в таблицу стилей следующий код:

```
.gallery figure:hover figcaption {
  opacity: 1;
}
```

Этот хитрый фрагмент CSS-кода можно расшифровать как «при установке указателя мыши поверх элемента `figure` (`figure:hover`) внутри элемента с классом `gallery` (`.gallery`) установить уровень непрозрачности подписи равным 1». То есть при установке указателя мыши поверх элемента `figure` его элемент-потомок `figcaption` становится видимым.

Сохраните страницу и посмотрите, что получилось. Когда указатель мыши помещается поверх изображения, должна появляться подпись. Мы можем анимировать этот эффект, добавив в стиль `.gallery figcaption` переход.

7. Отредактируйте стиль `.gallery figcaption`, чтобы он приобрел следующий вид (изменения выделены полужирным шрифтом):

```
.gallery figcaption {
  position: absolute;
  top: 15%;
  bottom: 15%;
  left: 0;
  right: 0;
  background-color: rgba(153,153,153,.9);
  padding: 20px;
  font-family: Titillium, Arial, sans-serif;
  font-weight: 400;
  font-size: .9em;
  color: white;
  opacity: 0;
  transition: opacity .75s ease-out;
}
```

Вы добавили свойство `transition`. Стоит отметить, что Internet Explorer 9 и более ранние версии этого браузера не поддерживают переходы, но здесь это не вызовет проблем, подписи все равно будут появляться в этом браузере. Они будут моментально возникать и исчезать, а не постепенно становиться видимыми и затухать.

И наконец, нужно привязать сведения об авторских правах к нижней части окна браузера, используя фиксированное позиционирование.

8. Добавьте в таблицу стилей следующий код:

```
footer {
  position: fixed;
  bottom: 0;
  left: 0;
```

```

right: 0;
padding: 10px;
background-color: black;
color: white;
}

```

Как уже говорилось, фиксированное позиционирование позволяет «привязать» элемент к определенной позиции в окне браузера. В данном случае элемент привязывается к нижней части страницы (`bottom: 0`) и расширяется на всю страницу (благодаря объявлениям `left: 0` и `right: 0`). Последние три объявления добавляют пространство по периметру нижнего колонтитула, а также устанавливают ему черный фон и белый текст.

9. Сохраните файл `styles.css` и просмотрите страницу `index.html` в браузере.

В окончательном виде страница должна выглядеть так, как показано на рис. 14.13. Полную версию этого урока можно найти в папке `14_finished`.

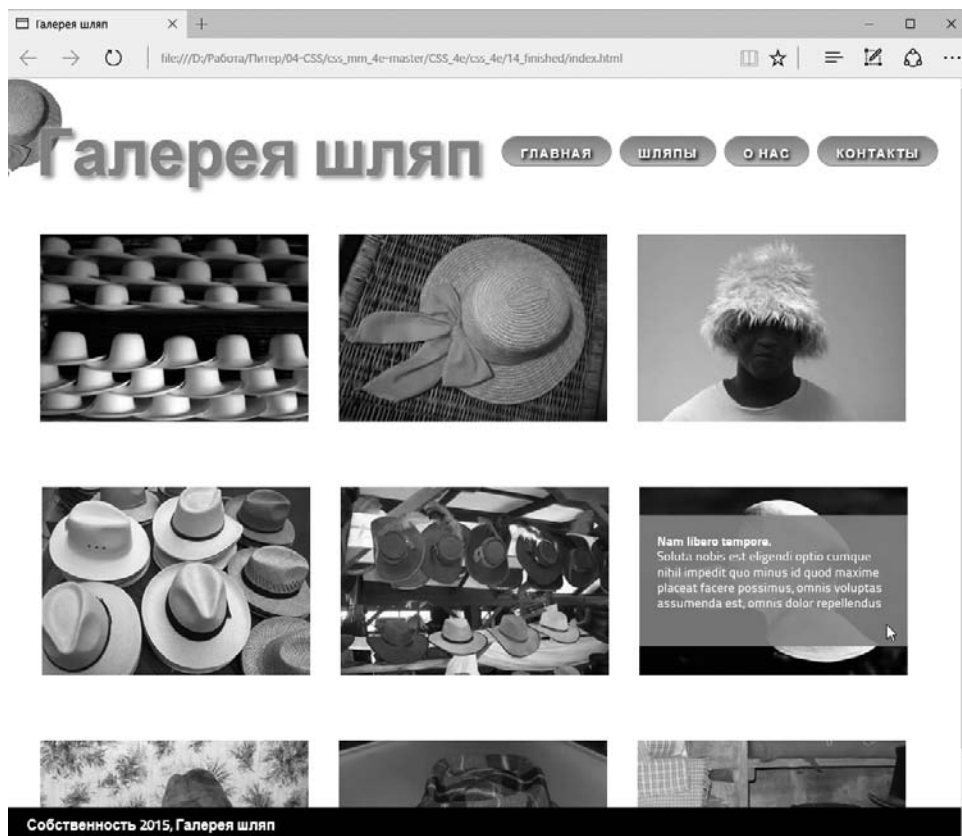


Рис. 14.13. Используя искусный CSS-код, можно без особого труда заставить подписи плавно появляться при наведении указателя на изображение

15 Адаптивный веб-дизайн

Веб-дизайнерам всегда приходилось решать задачу разработки сайтов под различные размеры экрана, от дисплеев ноутбуков с разрешением по горизонтали 760 пикселей до огромных широкоэкранных мониторов. С ростом количества смартфонов и планшетных компьютеров разработка в расчете на широкий диапазон экранов с различным разрешением стала еще актуальнее. Некоторые компании зашли так далеко, что стали создавать отдельные сайты, предназначенные только для мобильных устройств (рис. 15.1, *вверху*). Но при дефиците времени, средств и технических знаний для разработки двух сайтов и программного обеспечения для веб-сервера, предоставляющего нужный сайт тому или иному устройству, сайты исключительно для мобильных устройств вам не по плечу.

К счастью, есть еще один, более простой подход, позволяющий создать один сайт, адаптируемый к устройствам с различным разрешением экрана (см. рис. 15.1, *внизу*). Эта технология, получившая название «адаптивный веб-дизайн», использует ряд различных приемов, заставляющих страницу изменять макет на основе ширины окна браузера. Например, на смартфоне страницу можно скомпоновать в одну удобную для чтения колонку, помещающуюся на узком экране (см. рис. 15.1, *внизу слева*), а на более широких мониторах поддерживать компоновку в несколько колонок (см. рис. 15.1, *внизу справа*).

Основы адаптивного веб-дизайна

Читать веб-страницу, состоящую из четырех колонок на экране смартфона, имеющем разрешение по горизонтали 320 пикселей, очень трудно. Не менее трудно читать одну колонку текста, растянутую на 2560 пикселей огромного компьютерного монитора. *Адаптивный веб-дизайн* (термин введен Итаном Маркоттом) является попыткой решить эту проблему. Адаптивный веб-дизайн позволяет изменять всю компоновку страницы на основе ширины окна браузера (наряду с другими факторами), допуская создание наиболее легко читаемых представлений для каждого устройства, не испытывая необходимости в создании нескольких версий одного и того же сайта. Адаптивный веб-дизайн не является единой технологией или методом. В нем собраны воедино несколько методов CSS- и HTML-верстки для создания веб-страниц, макеты которых адаптируются к различным экранам.



Рис. 15.1. На смартфоне страница может появиться в виде длинной колонки, в окне браузера компьютера та же самая страница будет использовать всю ширину экрана и включать в себя несколько колонок и более крупные иллюстрации

ПРИМЕЧАНИЕ

Свои подходы к адаптивному веб-дизайну Итан Маркотт излагает в книге «Отзывчивый веб-дизайн». Можно также прочитать статью, посвященную адаптивному веб-дизайну, этого же автора по адресу tinyurl.com/c68oc3m.

В адаптивном веб-дизайне объединены три основные концепции: гибкие сетки для компоновки, гибкая среда для изображений и видеоконтента и медиазапросы

CSS, предназначенные для создания различных стилей для экранов с разным разрешением. Гибкие сетки позволяют избавиться от фиксированных дизайнов. Поскольку у смартфонов экраны бывают с самым разным разрешением, создавать страницы с фиксированной шириной нет никакого смысла, тут нужна страница, способная расширяться и сужаться, чтобы поместиться на экране конкретного устройства. Создание гибкой среды для передачи мультимедийного контента позволяет подбирать масштаб изображениям и видеоматериалам, чтобы они поместились на соответствующем экране — большие фотографии на больших мониторах, фотографии поменьше на небольших экранах и т. д.

И наконец, медиазапросы являются технологией каскадных таблиц стилей, позволяющей на основе создавшихся условий отправлять браузеру различные стили. Например, для экрана, имеющего разрешение по горизонтали менее 480 пикселей, можно отправить один набор стилей, а для экрана с разрешением по горизонтали 760 пикселей — другой. Но одной только шириной дело не ограничивается: можно разрабатывать стили, применяемые только к планшетным устройствам при альбомном просмотре или к устройствам с экранами с высокой плотностью пикселей (например, к смартфонам iPhone и планшетам iPad с дисплеем Retina).

Создание адаптивного дизайна веб-страницы

Если у вас есть смартфон под управлением операционной системы iOS или Android, присмотритесь к нему повнимательнее. Откройте браузер и перейдите на страницу nytimes.com. Вы попадете на версию сайта, предназначенную специально для мобильных устройств — mobile.nytimes.com. Тем не менее на ней доступна ссылка на версию сайта, предназначенную для компьютеров. Вы увидите контент, подобный показанному на рис. 15.2 (если только издание *New York Times* с момента написания книги не сделало свой сайт адаптивным). Это крупноформатный, многоколоночный дизайн, втиснутый в небольшое пространство экрана смартфона. Поскольку производители смартфонов понимают, что большинство сайтов созданы для экранов компьютерных мониторов, они заставили свои браузеры вести себя немного непривычно для вас. Мобильные браузеры не отображают страницу на все 100 %; если бы они это сделали, то страница шириной 960 пикселей не поместилась бы на экране и вы бы увидели только часть этой страницы. Затем, чтобы увидеть всю страницу, вам пришлось бы смахивать экран в разных направлениях. Вместо этого, чтобы страница поместилась на экране, браузеры смартфонов уменьшают масштаб. Конкретный коэффициент уменьшения варьируется в зависимости от характеристики той или иной модели смартфона. Например, браузер Safari смартфона iPhone работает так, будто экран действительно имеет разрешение по горизонтали 980 пикселей, и уменьшает страницу, чтобы она уместилась в этих 980 пикселях.

Впрочем, подобное поведение мобильных браузеров позволяет неплохо справиться с большинством сайтов, но с адаптивным веб-дизайном оно сочетается не очень хорошо. Так как адаптивные сайты предназначены для получения хорошего вида на смартфонах, уменьшать масштаб на экране не нужно, поскольку текст станет слишком мелким для чтения. К счастью, существует довольно простой способ

отмены такого поведения в браузерах мобильных устройств. Нужно в раздел заголовка вашей веб-страницы добавить следующий код (самое подходящее место для этого будет непосредственно перед элементом title):

```
<meta name="viewport" content="width=device-width">
```

Метатеги языка HTML предоставляют дополнительную информацию о содержимом страницы и могут передавать браузерам дополнительные инструкции о способах отображения страницы на экране. В данном случае viewport обозначает экран браузера, а для атрибута content устанавливается ширина окна браузера, равная разрешению по горизонтали экрана смартфона. То есть браузерам мобильных устройств, склонным к уменьшению масштаба, предписывается этого не делать, настроив ширину экрана на текущее разрешение по горизонтали экрана смартфона.



Рис. 15.2. Сайты, разработанные для браузеров компьютеров, на смартфонах выглядят как небольшие почтовые марки. Посетителям приходится увеличивать масштаб изображения и, чтобы все прочитать, перетаскивать страницу по экрану

Медиазапросы

Язык CSS включает понятие *медиазапросов*. Медиазапросы позволяют назначать страницам стили на основе размера (ширины и высоты) окна целевого браузера. С помощью этого метода можно создавать пользовательские стили для браузеров смартфонов, планшетов и компьютеров и тем самым настраивать представление вашего сайта таким образом, чтобы он выглядел наилучшим образом на экране каждого типа устройств.

Весь смысл адаптивного дизайна состоит в том, чтобы дать посетителям вашего сайта наиболее читаемое и привлекательное представление. Обычно это означает настройку дизайна под наилучший вид в окнах браузеров различной ширины. Многие разработчики ориентируются на три наиболее распространенных устройства просмотра веб-контента: смартфоны, планшеты и компьютеры. При условии широкого диапазона размеров экранов этих устройств (могут быть смартфоны как с маленьким экраном, так и с большим, 8-дюймовые и 10-дюймовые планшетные компьютеры и т. д.) единого значения ширины области просмотра для всех этих устройств не существует. И целью является нормальный внешний вид страницы при различном разрешении экрана по горизонтали. Вы можете протестировать различные дизайнерские решения в окнах разной ширины, чтобы увидеть, когда четырехколоночный дизайн должен превращаться в одно- или двухколоночный.

Стратегии использования медиазапросов

Хотя для решения о внесении изменений в дизайн, позволяющего придать ему наиболее удачный внешний вид на различных устройствах, нами рекомендуется метод проб и ошибок, существует ряд общих дизайнерских изменений, на которые обычно нацелены медиазапросы.

- **Изменение количества колонок.** Несколько колонок контента неплохо смотрятся на больших мониторах (и даже на планшетных компьютерах в альбомном режиме), но не подходят для смартфонов. Кроме того, четырех колонок, скорее всего, будет многовато для большинства планшетных компьютеров с книжной ориентацией экрана, поэтому сведение дизайна страницы к двум или трем колонкам, по всей видимости, вполне подходит для медиазапросов, нацеленных на планшетные компьютеры. Исключение обтекаемых элементов в стилях медиазапросов, направленных на планшетные компьютеры, позволяет складывать контейнеры с контентом страницы друг на друга. Эта технология будет опробована в практикуме этой главы.
- **Гибкая ширина.** Фиксированный дизайн можно использовать для браузеров компьютеров — именно так годами и поступали дизайнеры. Но на более узких экранах планшетных компьютеров и смартфонов фиксированные элементы не поместятся. Страница шириной 960 пикселей будет слишком велика для смартфонов с экранами с разрешением по горизонтали 320 или 480 пикселей. Для смартфонов и планшетов более удачным подходом будет присвоение свойству `width` контейнеров `div` с контентом значения `auto` или `100%`. Эти установки превратят дизайн вашей страницы из фиксированного в резиновый,

или гибкий. Иными словами, независимо от разрешения экрана смартфона `div`-контейнеры поместятся на нем полностью. Если держать смартфон iPhone в книжной ориентации (при разрешении экрана по горизонтали 320 пикселей), а затем быстро повернуть в альбомную (изменив тем самым разрешение экрана по горизонтали до значения 480 пикселей), `div`-контейнеры со свойством `width: auto` или `width: 100%` просто изменят свой размер, чтобы поместиться в новом пространстве.

- **Сжатие пустых пространств.** Обширные промежутки между заголовками, изображениями и другими элементами страницы позволяют свободно компоновать дизайн для огромных мониторов, но приводят к хаосу и нерациональному использованию пространства на небольших экранах смартфонов, принуждая посетителей прибегать к прокрутке. Сужение полей и отступов позволяет разместить на таких небольших экранах больше полезной информации.
- **Настройка размеров шрифта.** Контраст между крупными, полужирными заголовками и набранным мелким шрифтом основным текстом неплохо выглядит на компьютерных мониторах. Но на портативных устройствах излишне крупные заголовки труднее читаются и совершенно необоснованно занимают полезное пространство. А вот незначительное увеличение шрифта обычного текста на смартфонах зачастую облегчает его чтение. Иначе говоря, создавая стили медиазапросов, обращайте внимание на размеры шрифтов.
- **Изменение навигационных меню.** У вас может быть красиво оформленная горизонтальная панель навигации, занимающая всю верхнюю часть веб-страницы и состоящая из десятка кнопок, направляющих посетителей к разным разделам вашего сайта. К сожалению, по мере сужения окна браузера эти кнопки могут не поместиться на весь экран. Они будут разнесены на две, три и более строки. Пример такого явления вы увидите в практикуме в конце главы. Возможно, в том, что навигационная панель займет не одну, а несколько строк экрана, не будет ничего страшного, но ведь эта панель может занять в верхней части страницы слишком много места, заставляя пользователей задействовать прокрутку, чтобы добраться до первых строк реального контента.

Увы, каскадные таблицы стилей не предлагают простых и понятных решений. На многих сайтах для динамического превращения навигационного меню в раскрывающийся список используется JavaScript, тогда это меню занимает небольшое пространство экрана (чтобы этому научиться, обратитесь по адресу tinyurl.com/q96ertm). Но есть и другие решения. Обзор различных подходов, применяемых на некоторых сайтах, представлен на сайтах tinyurl.com/l9ox9sv и tinyurl.com/oyvmpd6.

- **Скрытие лишнего контента на портативных устройствах.** Многие разработчики скрывают некоторый контент на мобильных версиях сайтов. На компьютерном мониторе просмотр нескольких колонок и сотен строк текста дается довольно легко, а вот большое количество информации на смартфоне может показаться совершенно излишним. Каскадные таблицы стилей можно использовать для того, чтобы скрыть контент, который, на ваш взгляд, не нужно показывать пользователям мобильных устройств, для чего следует присвоить свойству `display` значение `none`.

Но все же нужно иметь в виду, что, скрывая контент, вы отстраняете посетителя от той информации, которая представляется на вашем сайте. Тем, кто ранее посетил ваш сайт с компьютера, а теперь посещает его со смартфона, будет крайне неприятно увидеть, что совсем недавно просматриваемая ими важная информация теперь куда-то исчезла. Кроме того, даже если вы скроете контент с помощью каскадных таблиц стилей, сам HTML-код никуда не денется, заставляя смартфон впустую тратить время и трафик на загрузку неиспользуемого HTML-кода.

- **Использование фоновых изображений.** Если поместить на экран 960-пиксельный баннер, то ни один смартфон не покажет его без уменьшения масштаба. Можно, конечно, представить достаточно небольшое изображение, способное поместиться на экране смартфона, или же воспользоваться вместо него фоновыми изображениями. Можно, например, создать div-контейнер и добавить к нему следующий класс: `<div class="logo">`. Затем в таблице стилей для браузера компьютера установить ширину/высоту div-контейнера, соответствующую размеру большого логотипа, используя свойство `background-image` для вставки изображения в фон. Например:

```
.logo {
  width: 960px;
  height: 120px;
  background-image: url(images/large_logo.png)
}
```

Затем можно добавить еще один стиль, используемый смартфонами, который изменяет размеры div-контейнера и устанавливает другое фоновое изображение:

```
.logo {
  width: 100%;
  height: 60px;
  background-image: url(images/small_logo.png)
}
```

В разделе «Адаптивные изображения» далее в этой главе вы научитесь масштабировать изображения, вставляемые в HTML-код с помощью элемента `img`, чтобы они помещались в окна браузеров различной ширины.

Создание точек останова

Медиазапросы позволяют отправлять браузерам различные стили на основе ширины окон этих браузеров. Например, браузеру можно сообщить следующее: «Если твой экран шире 480 пикселей, применяй те стили» или «Если твой экран шире 480 пикселей, но уже 769 пикселей, применяй эти стили». Различные значения ширины, указываемые вами, — 480, 769 и т. д. — в адаптивном дизайне часто называют *точками останова*. А вообще-то, с каких значений нужно начинать разбивать дизайн на точки останова?

Проще всего это определить, если взять готовый дизайн для компьютера и открыть страницу в браузере. Нажав и удерживая кнопку мыши на краю окна браузера,

перетаскивайте указатель и медленно уменьшайте ширину окна. В определенный момент времени страница приобретет совершенно неприглядный вид. К примеру, станет тесно ее четырем колонкам. Та позиция, в которой дизайн теряет приемлемый внешний вид, становится хорошим кандидатом на точку останова, то есть этот размер вполне подходит для определения нового медиазапроса и для загрузки новых стилей, чтобы удалить одну или две колонки.

Нередко создают три набора медиазапросов для трех различных точек останова — один для смартфонов, другой для планшетов, а третий — для компьютерных мониторов. Конкретные значения точек останова зависят от конкретного дизайна (а также от конкретного устройства), но чаще всего отправной точкой служит экран, имеющий разрешение по горизонтали меньше 480 пикселей, который получает один набор стилей, экран с разрешением по горизонтали между 481 и 768 пикселями получает второй набор стилей, а устройства с разрешением по горизонтали больше 768 пикселей получают дизайн, предназначенный для компьютеров. Но все это в конечном счете остается на ваше усмотрение. Некоторые дизайнеры допускают расширение зоны планшетных компьютеров до 1024 пикселей, а стили для компьютеров начинают отправлять тем браузерам, ширина окна которых превышает 1024 пикселя.

Некоторые дизайнеры даже доходят до определения четырех или пяти точек останова, чтобы их творения хорошо смотрелись в более широком диапазоне экранов. Подробности создания этих точек останова с использованием медиазапросов будут даны в подразделе «Создание медиазапросов» далее.

Приоритет мобильных устройств или компьютеров?

Следует рассмотреть еще один вопрос: с расчетом на какое устройство нужно приступать к разработке дизайна? Вам не нужно создавать три отдельных набора стилей, по одному для каждого из устройств, на которые вы нацелены. Вы можете и должны сначала создать *исходный* дизайн, то есть дизайн, работающий без медиазапросов. Затем можно создать стили медиазапросов для замены исходных стилей и переформатирования страницы под конкретное разрешение экрана по горизонтали. Для этого существует два основных подхода.

- **Стратегия Desktop first:** предпочтение компьютерам. Дизайн сайта можно разрабатывать с прицелом на компьютеры. Создайте все требуемые колонки. Отработайте дизайн до совершенства, чтобы он хорошо выглядел на большом мониторе. Теперь это будет ваш исходный дизайн, и все стили можно собрать во внешнюю таблицу стилей и привязать ее к страницам вашего сайта обычным образом.

Затем добавьте медиазапросы для планшетов и смартфонов. Стили этих медиазапросов будут подстраивать «компьютерный» дизайн под новые условия — удалять колонки, уменьшать шрифт заголовков и т. д.

- **Стратегия Mobile first:** предпочтение мобильным системам. Можно сделать все наоборот и сначала разработать дизайн для мобильных систем. Теперь в обычную внешнюю таблицу помещаются основные стили, предназначенные для

устройств с малыми экранами, а потом в медиазапросах дизайн дорабатывается для планшетов и компьютеров путем добавления колонок и других изменений для больших экранов.

Какой бы метод ни был выбран, нужно использовать обычную внешнюю таблицу стилей, привязанную к веб-странице обычным образом. В эту таблицу включаются все стили, являющиеся общими применительно к *различным* устройствам. Например, для всех версий сайта требуется одинаковая цветовая палитра и одинаковые шрифты. Можно также использовать одинаковые стили для ссылок, изображений и других HTML-элементов. Иными словами, вам не нужно создавать для каждого устройства три абсолютно отдельных набора стилей, начните с одного набора, применяемого ко всем браузерам, как смартфонов, так и планшетов и компьютеров, а затем уточните дизайн для устройств, на которые нацелены медиазапросы.

Создание медиазапросов

Запрос, по сути, представляет собой вопрос, заданный браузеру: «Равна ли ширина окна браузера 480 пикселям?» Если ответ положительный, браузер использует таблицу стилей именно для устройства с данным разрешением экрана по горизонтали (предоставляемая вами таблица стилей, рассмотренная ранее). Код, выполняющий эту задачу, схож с кодом привязки любой другой внешней таблицы стилей:

```
<link href="css/small.css" rel="stylesheet" media="(width: 480px)">
```

К этой стандартной ссылке на таблицу стилей добавился еще один атрибут `media`, где определены условия, при которых браузер использует указанную таблицу. В примере выше браузер загружает внешнюю таблицу стилей `small.css`, когда посетитель просматривает ваш сайт с помощью браузера, ширина окна которого составляет 480 пикселей. Скобки, в которые заключен запрос, — `(width: 480px)` — обязательны. Если их не указать, браузер запрос проигнорирует.

Конечно, 480 пикселей очень точное значение, а как быть, если посетитель пользуется смартфоном с экраном разрешением по горизонтали, скажем, 300 пикселей? Поэтому рекомендуется использовать в медиазапросе диапазон значений. Например, может понадобиться применить конкретный стиль для экранов, разрешение по горизонтали которых *меньше или равно* 480 пикселям. Это можно сделать с помощью следующего кода:

```
<link href="css/small.css" rel="stylesheet" media="(max-width:480px)">
```

Запись `(max-width:480px)` эквивалентна высказыванию «для экранов, имеющих разрешение по горизонтали не более 480 пикселей». Поэтому стили из файла `small.css` будут применены, к примеру, к экранам с разрешением по горизонтали 480, 320 и 200 пикселей.

Существует также вариант `min-width`, определяющий, имеет ли браузер минимальную указанную ширину окна. Этот вариант применяется при нацеливании на устройство, экран которого больше смартфона или планшета. Например, чтобы применить

стили на устройствах с разрешением по горизонтали более 768 пикселей, что превосходит разрешение экрана по горизонтали многих планшетных компьютеров, нужно использовать следующий код:

```
<link href="css/large.css" rel="stylesheet" media="(min-width:769px)">
```

Чтобы эта таблица стилей была применена, окно браузера должно быть шириной не менее 769 пикселей, что на 1 пиксел больше, чем 768 пикселей, составляющих разрешение экрана по горизонтали ряда планшетных устройств.

И наконец, можно установить как максимальное, так и минимальное значение ширины экрана целевых устройств, чтобы в них попадали браузеры устройств *между* смартфонами и компьютерами. Например, чтобы создать набор стилей для планшетного компьютера, имеющего экран с разрешением по горизонтали 768 пикселей, можно воспользоваться следующим CSS-кодом:

```
<link href="css/medium.css" rel="stylesheet" media="(min-width:481px) and (max-width:768px)">
```

Иными словами, окно браузера должен быть шириной не менее 481 и не более 768 пикселей. К примеру, файл `medium.css` не будет применяться на смартфонах с разрешением экрана по горизонтали 320 пикселей, а также на компьютерных мониторах с разрешением экрана по горизонтали 1024 пикселя.

ПРИМЕЧАНИЕ

Медиазапросы каскадных таблиц стилей способны не только на проверку ширины окна браузера. Текущие стандарты медиазапросов позволяют проверять высоту и ориентацию (то есть узнать, в каком положении находится смартфон или планшет посетителя: в горизонтальном или вертикальном) и даже определять типа экрана устройства: цветной или монохромный. Получить дополнительные сведения о медиазапросах можно на сайте Консорциума W3C по адресу tinyurl.com/3xjdbg.

Добавление медиазапросов в таблицу стилей

Показанная выше технология является одним из способов использования медиазапросов с элементом `link` для загрузки различных таблиц стилей на устройствах с экранами с разным разрешением по горизонтали. Но медиазапросы можно также добавлять и внутри единой таблицы стилей. Возможно, вам захочется сделать это, чтобы не пришлось, к примеру, добавлять в HTML-файл несколько элементов `link`, или понадобится хранить все стили медиазапросов в основной таблице стилей. Большинство веб-дизайнеров используют этот подход вместо нескольких отдельных файлов для каждого медиазапроса.

Существует два способа добавления медиазапросов в таблицу стилей.

- **Использование правила `@import`.** Эта технология уже рассматривалась в главе 2. Правило `@import` позволяет загружать дополнительные внешние таблицы стилей во внутреннюю либо во внешнюю таблицу стилей. Правило `@import` можно также использовать с медиазапросом. Предположим, нужно загрузить внешнюю таблицу стилей `small.css`, содержащую стили для экранов с разрешением по горизонтали не более 320 пикселей. Для этого добавьте следующее правило `@import` непосредственно в таблицу стилей:

```
@import url(css/small.css) (max-width:320px);
```


ПРИМЕЧАНИЕ

Правила `@import` должны помещаться в начало таблицы стилей. Их нельзя указывать после каких-либо стилей. В результате могут возникать проблемы с каскадностью, при которых стили, определенные во внешней таблице стилей и загруженные с помощью правила `@import`, будут замещаться более поздними стилями в таблице стилей. Этой проблемы можно избежать созданием одной внешней таблицы стилей, которая содержит только правила `@import`. Первая из них приведет к загрузке основной таблицы стилей, предназначенной для всех устройств, а вторая и третья приведут к загрузке таблиц стилей с использованием медиазапросов:

```
@import url(css/base.css); /* без медиазапроса, для всех */
@import url(css/medium.css) (min-width:481px) and (max-width:768);
@import url(css/small.css) (max-width: 480px);
```

- **Внедрение медиазапроса в таблицу стилей.** Медиазапрос можно также внедрить непосредственно в таблицу стилей:

```
@media (max-width: 480px) {

    body {

        /* сюда помещаются свойства стиля*/

    }

    .style1 {

        /* сюда помещаются свойства стиля */

    }

}
```

Правило `@media` функционирует как своеобразный контейнер для всех стилей, соответствующих запросу. Следовательно, в этом примере стили `body` и `.style1` применяются только к тем устройствам, разрешение экрана по горизонтали которых не превышает 480 пикселей. Использование встраиваемых правил `@media` позволяет организовать все ваши стили в одну таблицу стилей. Рекомендуется начинать внешние таблицы с тех стилей, которые не содержатся в медиазапросах, отдавая сначала предпочтение стилям либо для компьютеров, либо для мобильных устройств, а затем добавляя медиазапросы для всех остальных устройств. Этот подход наиболее общий, используемый большинством веб-дизайнеров.

Основная структура таблицы стилей

Можно сказать, что существует множество различных способов использования медиа-запросов, нацеленных на различные устройства: с приоритетом, отдаваемым компьютерам, с приоритетом, отдаваемым мобильным системам, в отдельных таблицах стилей, в единой таблице стилей и т. д. По мере накопления опыта вы поймете, какой из способов больше подойдет для вашего проекта. Но вначале нужно всегда создавать единую внешнюю таблицу стилей, включая в нее стили для компьютерных

мониторов, а затем добавляя медиазапросы с изменениями этого дизайна основного уровня под планшетные устройства и смартфоны. Схематично структура для такого файла должна иметь следующий вид:

```
/* Сброс стилей браузера */
/* стили для компьютеров и базовые стили для всех устройств */
body {
  /* свойства тела страницы */
}

/* только для устройств со средним размером экрана */
@media (min-width: 481px) and (max-width:768px) {
  body {
    /* только для планшетов */
  }
}

/* только для устройств с малым размером экрана */
@media (max-width:480px) {
  body {
    /* только для смартфонов */
  }
}
}
```

Вы увидите эту структуру в действии в практикуме в конце текущей главы, а базовую таблицу стилей можно найти в файле `desktop_first.css`, который находится в папке 15 заданий практикума по адресу github.com/mrightman/css_4e. Даже притом, что в этот файл включены медиазапросы, он содержит обыкновенную таблицу стилей, которая привязывается к веб-странице обычным образом. Например, если сохранить этот файл в его текущем виде, нужно добавить в раздел заголовка веб-страницы следующий HTML-код:

```
<link href="styles.css" rel="stylesheet">
```

Приоритет мобильных устройств

Если вы решили отдать предпочтение мобильным системам, то сначала нужно создать набор стилей, нацеленный на браузеры мобильных систем, а затем добавить медиазапросы для изменения дизайна под браузеры планшетов и компьютеров. Основная структура таблицы стилей при таком подходе будет иметь следующий вид:

```
/* Конец кода сброса стилей браузера */

/* сброс стилей браузера */
/* стили для мобильных устройств и базовые стили для всех устройств */
body {
  /* свойства тела страницы */
}
}
```

```
/* только для устройств со средним размером экрана */
@media (min-width: 481px) and (max-width:768px) {
  body {
    /* только для планшетов */
  }
}

/* только для устройств с крупным размером экрана */
@media (min-width:769px) {
  body {
    /* только для компьютеров */
  }
}
```

Начальную основную таблицу стилей можно найти в файле `mobile_first.css`, который находится в папке 15 с примерами для этой главы.

Гибкие сетки

Не стоит поддаваться соблазну разработки отдельных фиксированных дизайнов для поддержки конкретных устройств, к примеру шириной 375 пикселей для iPhone 6, шириной 768 пикселей для планшетов iPad в книжной ориентации и шириной 1000 пикселей для компьютерных мониторов. Не делайте этого. Хотя смартфоны iPhone 6 и приобрели широкую популярность, на них свет клином не сошелся. Есть еще и другие модели iPhone, и смартфоны под управлением операционной системы Android, имеющие всевозможные формы и размеры, а стало быть, и разрешение экрана. Не сомневаюсь, что вам попадались и многие другие устройства с необычными размерами, да и планшетные устройства имеют весьма разнообразные разрешения экрана. То есть универсального значения ширины страниц для смартфонов и планшетных устройств попросту нет, поэтому лучше всего создавать страницы с изменяемой шириной.

Основным компонентом адаптивного веб-дизайна являются гибкие сетки. Это не что иное, как резиновый макет, который рассматривался в главе 12 и у которого общая ширина страницы изменяется, чтобы поместиться на экранах с разным разрешением по горизонтали. В большинстве случаев это означает, что свойству ширины присваивается значение 100%. Но с учетом пользователей компьютеров, возможно, потребуется использовать свойство `max-width`, чтобы страница не становилась крайне широкой на мониторах с высоким разрешением.

Кроме того, в процентах должна измеряться ширина и отдельных колонок, вместо пикселей или единиц `em`. Колонки по отдельности также должны становиться шире или уже, чтобы вписываться в изменяющуюся ширину страницы.

СОВЕТ

В следующей главе вы познакомитесь с некоторыми бесплатными системами гибких сеток. Фактически с помощью одной из самых популярных систем под названием Skeleton очень легко создавать многоколоночные веб-страницы на основе гибких сеток.

Предположим, что нужно создать двухколоночный дизайн, в котором первая колонка занимает одну треть ширины страницы, а вторая — две трети. Можно начать с создания следующего HTML-кода:

```
<div class="columns">
  <div class="one-third">
    ...контент...
  </div>
  <div class="two-thirds">
    ...контент...
  </div>
</div>
```

Затем можно добавить несколько стилей для создания резинового дизайна:

```
.columns {
  width: auto; /* same as 100% */
  max-width: 1200px;
}
.columns:after {
  content: "";
  display: table;
  clear: both;
}
.one-third {
  float: left;
  width: 33%;
}
.two-thirds {
  float: left;
  width: 67%;
}
```

Первый стиль — `.columns` — устанавливает ширину `div`-контейнера, содержащего колонки. Присвоение свойству `width` значения `auto` — то же самое, что и установка значения `100%`, поскольку свойство `max-width` будет ограничивать блок от приобретения слишком большой ширины. Второй стиль — `.columns:after` — помогает управлять двумя обтекаемыми колонками (подробности, как это работает, можно найти в разделе «Решение проблем с обтекаемыми элементами» главы 13). И наконец, последние два стиля устанавливают ширину двух `div`-контейнеров равной `33%` (в ширину одной трети его контейнера) и `67%` (две трети), а также выравнивают их по левому краю, чтобы они появлялись рядом.

Значимость HTML-кода

В книжной ориентации смартфона на его экране, при условии сохранения читабельности страницы, попросту не хватит пространства не только для трех, а даже и для двух колонок контента. Поэтому многие разработчики для отображения контента страницы на экране смартфона компонуют его в одну большую колонку. Для этого нужно удалить из созданных колонок все обтекаемые элементы. Например, если

создается трехколоночный дизайн для компьютерных мониторов и в нем используются обтекаемые элементы для позиционирования колонок друг рядом с другом, нужно присвоить свойству `float` таких элементов значение `none`. Тогда они отобразят HTML-контент в обычном порядке — один блочный элемент за другим.

Такое поведение придает порядку следования исходного HTML-кода особую важность. Например, у страницы могут быть две боковые панели, первая с перечнем ссылок на родственные сайты, а вторая — с рекламой товаров, продаваемых вашей компанией. При этом основной контент, ради которого, собственно, и посещают вашу страницу, содержится в средней колонке. Один из способов компоновки всего этого по колонкам заключается в выравнивании первой боковой панели по левому краю, а второй боковой панели — по правому краю, позволяя основной колонке обтекать две прочие колонки и находиться в центре.

В понятиях HTML это будет означать, что сначала отображаются два `div`-контейнера боковых панелей, а затем следует элемент, содержащий основной контент. Если адаптировать эту страницу под мобильные устройства, удалив указание на то, что боковые панели являются обтекаемыми элементами, то получится, что две боковые панели отобразятся *ранее* основного контента. Вашей аудитории придется прокручивать страницу вниз, чтобы пропустить рекламу и ссылки и добраться до нужного ей контента.

Лучшим решением было бы поместить контейнер с основным контентом перед боковыми панелями. Как уже упоминалось в разделе «Использование обтекаемых элементов при верстке» главы 13, этот метод может потребовать добавления дополнительных контейнеров и выравнивания всех элементов, включая `div`-контейнер с основным контентом.

Одним словом, нужно соблюдать соответствующий порядок следования HTML-элементов, прежде чем создавать многоколоночный дизайн для компьютеров. Проще всего понять, что происходит, если просмотреть страницу без применения к ней каскадных таблиц стилей. Тогда вы увидите все `div`-контейнеры и другие блочные элементы в порядке их следования и сможете понять, как будет выглядеть страница в виде одной колонки на экране смартфона.

ПРИМЕЧАНИЕ

Как можно создать трехколоночный дизайн для компьютеров при условии размещения наиболее важного контента в верхней части страницы, показано в практикуме в конце этой главы.

Сброс блочной модели

Как уже объяснялось в подразделе «Предотвращение выпадений обтекаемых элементов» раздела «Решение проблем с обтекаемыми элементами» главы 13, при ограничении ширины в процентном отношении возникает угроза выпадения обтекаемых элементов, если общая ширина колонок в одном ряду превышает 100 %, из-за чего последняя колонка выпадает под другие колонки. Из-за способа, используемого браузерами для вычисления ширины элементов, увеличение толщины границы по периметру `div`-контейнера или добавление к нему внутренних отступов приводит к тому, что ширина `div`-контейнера на экране становится больше указанной в каскадной таблице стилей.

Например, если для одной колонки указана ширина 33 %, а для другой — 67 % (как в примере, приведенном ранее), они должны уместиться рядом друг с другом, поскольку их общая ширина составляет 100 %. Но если к колонке добавлена граница толщиной 1 пиксел, то ее общая ширина станет 100 % + 2 пиксела (учитываются левая и правая границы). Теперь эта колонка будет слишком большой, чтобы поместиться в окне, и вторая колонка выпадет под первую.

Существует несколько способов преодоления этого затруднительного положения, и все они были рассмотрены в подразделе «Предотвращение выпадений обтекаемых элементов» раздела «Решение проблем с обтекаемыми элементами» главы 13. Но наиболее очевидным решением станет предписание браузеру *засчитывать* в общую ширину и высоту элемента границы и отступы в качестве части его вычислений в рамках блочной модели. То есть браузер можно инструктировать включать границы и отступы в состав свойства width, чтобы добавление дополнительных отступов или границ не приводило к увеличению ширины (или высоты) элемента. Поскольку такой подход неплохо бы применить ко всем элементам, лучше воспользоваться универсальным селектором, позволяющим сбросить исходные параметры блочной модели для всех элементов, имеющих на странице:

```
* {  
  box-sizing: border-box;  
}
```

Лучше всего поместить этот стиль в код каскадной таблицы стилей, используемый для сброса исходных установок страницы (см. раздел «Управление каскадностью» главы 5).

Преобразование фиксированного макета в гибкие сетки

При разработке совершенно нового дизайна подбор процентных соотношений не сложен. В конце концов, если нужны четыре колонки одинаковой ширины, для каждой из них устанавливается значение 25%:

```
width: 25%;
```

Но если приходится иметь дело с фиксированным дизайном и нужно преобразовать его в резиновый, ситуация усложняется. Для начала представим, что вы разработали страницу с фиксированной шириной 960 пикселей. Либо контент страницы заключен в элемент div, для которого задана ширина 960 пикселей, либо такая ширина установлена для элемента body. В любом случае теперь нужно, чтобы этот контейнер был полностью резиновым. Для этого достаточно изменить код:

```
width: 960px;
```

```
на
```

```
width: auto;
```

Присвоение свойству width элемента значения auto — практически то же самое, что и использование кода width: 100%;; ширина этого элемента будет идентична ширине его контейнера.

Затем нужно преобразовать значения свойств `width` колонок из пикселей в проценты. Чтобы упростить вычисление, гуру адаптивного веб-дизайна Итан Маркотт придумал замечательную формулу: *цель / контекст = результат*. Проще говоря: «возьмите ширину элемента, подвергаемого преобразованию (в пикселах), и разделите ее на ширину контейнера этого элемента (в пикселах)». В результате получится дробное значение, которое нужно перевести в проценты.

Рассмотрим пример. Предположим, что на этой странице шириной 960 пикселей имеются две колонки: боковая панель шириной 180 пикселей и основная колонка шириной 780 пикселей. В CSS-файле имеется следующий код:

```
.sidebar {  
  float: left;  
  width: 180px;  
}  
.main {  
  float: left;  
  width: 780px;  
}
```

Конечно, в нем присутствует множество другого CSS-кода, задающего границы, фоновые цвета и т. д., но в данном случае нас интересуют только свойства ширины. Приступая к преобразованию боковой панели, нужно взять значение ее ширины, равное 180 пикселям, и разделить на значение ширины контейнера этой боковой панели, то есть 960 пикселей. Получившийся результат — 0,1875 — нужно умножить на 100 и получить процентное значение: 18,75 %. Точно так же для основной колонки: 780 делится на 960 и получается 0,8125. При умножении этого результата на 100 получаем 81,25 %. То есть нужно внести следующие изменения в код:

```
.sidebar {  
  float: left;  
  width: 18.75%;  
}  
.main {  
  float: left;  
  width: 81.25%;  
}
```

Полученные значения округлять не следует. То есть не превращайте 18,75 % в 19 %, поскольку это, скорее всего, приведет к выпадению обтекаемого элемента. Браузеры хорошо справляются с десятичными значениями, и вы спокойно можете использовать любые значения, возвращаемые калькулятором. Вполне допустима, скажем, ширина, заданная значением 25,48488 %.

То же самое применимо и к вложенным колонкам. Предположим, что в показанной выше основной колонке имеется один раздел, состоящий из двух обтекаемых `div`-контейнеров, создающих две дополнительные колонки внутри основной. Оба `div`-контейнера имеют одинаковую ширину, равную 390 пикселям, поэтому для вычисления их ширины в процентном отношении берется значение в пикселах — 390 — и делится на ширину их контейнера, которая в данном примере составляет 780 пикселей. В результате получается значение 0,5, умножение которого на

100 выдаст 50 %. (Но, по сути, эти вычисления не нужны. Ведь у вас имеются две расположенные рядом колонки одинаковых размеров, и вы знаете, что каждая из них занимает половину доступного пространства, то есть 50 %.)

После редизайна и вычисления процентных значений нужно помнить, что общая ширина всех колонок в одном ряду не должна превышать 100 %.

СОВЕТ

Ту же формулу можно применить для преобразования значений размеров из пикселей в единицы `em`. Предположим, что ширина текстового абзаца составляет 18 пикселей (цель). Исходный размер обычного текста (контекст) составляет 16 пикселей. При делении 18 на 16 получаем новый размер в единицах `em`: 1.125`em`.

Гибкие изображения

Наряду с тем, что гибкая компоновка позволит создать дизайн, который будет прекрасно отображаться на экранах с разрешением широкого спектра устройств, при добавлении на страницы изображений возникает проблема. Хотя колонки в гибком дизайне по мере уменьшения окна сжимаются, изображений обычно это не касается. Это может привести к выпадению изображений за обозначенные пределы и к тому, что они перестанут вписываться в ширину колонки (рис. 15.3).

К счастью, можно придать гибкости и изображениям. Для этого нужно выполнить два шага: создать новый стиль CSS и внести ряд изменений в HTML-код.

1. Для начала добавьте в таблицу стилей следующий код:

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

Этот код ограничит максимальный размер любого изображения значением 100 % от ширины контейнера этого изображения. То есть изображение не сможет стать шире колонки, `div`-контейнера или любого HTML-элемента, внутри которого оно находится.

Но этого еще недостаточно для обеспечения гибкости изображений. Обычно при добавлении элемента `img` указываются атрибуты его высоты и ширины. Именно эти размеры используются браузером при визуализации изображения. Если присвоить значение свойству `max-width`, изображение не станет шире колонки, но его высота по-прежнему будет точно соответствовать значению, указанному в HTML-коде. Таким образом, изображение подстроится под ширину колонки, а его высота не изменится, что приведет к искажению этого изображения. Решение вполне очевидно: нужно удалить из HTML-кода атрибуты `width` и `height`.

2. Найдите на странице все элементы `img` и удалите из них атрибуты `height` и `width`. То есть измените HTML-код:

```

```

следующим образом:

```

```

Во многих редакторах HTML-кода есть режим поиска и замены, который может ускорить поиск и удаление этих атрибутов.



Рис. 15.3. Если колонка становится уже, чем находящееся внутри нее изображение, это изображение выходит за пределы колонки, перекрывая другие колонки и контент страницы

Разумеется, этот подход предполагает, что все ваши изображения будут заполнять колонку, внутри которой они находятся.

Во многих случаях вам потребуется, чтобы изображения были меньше этого размера. Например, можно выровнять фотографию по левому краю основной колонки, а текст будет обтекать изображение по правому краю. Для работы с изображениями, размер которых задается по-разному, можно создать различные классы с иными значениями свойства `max-width` и применить эти классы к конкретным элементам `img` в HTML-коде. Итак, требуется, чтобы изображение было выровнено по левому краю колонки, а его размер составлял 40 % от ширины колонки. Сначала создадим стиль класса:

```
.imgSmallLeft {
  float: left;
  max-width: 40%;
}
```

Затем применим этот класс к элементу `img`:

```

```

Более гибким подходом будет разделить ограничение размера и установку выравнивания элемента по разным классам:

```
.imgSmall {
  max-width: 40%;
}
```

```
.imgLeft {  
  float: left;  
}
```

А затем применить к изображению оба класса:

```

```

Путем использования двух классов можно применить класс `imgSmall` к любым изображениям, даже к тем, которые вы решили выровнять по правому краю или вообще не выравнивать, но ограничить их ширину.

ПРИМЕЧАНИЕ

Изменить размер фоновых изображений можно с помощью свойства `background-size`.

Недостатки гибких изображений


У гибких изображений есть одна проблема. При просмотре страницы на смартфоне размеры колонки и изображения могут ужаться до ширины существенно меньшей, чем в окне браузера компьютера. Это означает, что изображения на экране будут сильно уменьшены. Хотя качество изображения от этого не пострадает, пользователи смартфонов будут вынуждены загружать графические файлы размером намного больше необходимого. Тем самым будет неоправданно расходоваться трафик на большие файлы изображений, в которых нет необходимости. К сожалению, пока эта проблема не имеет полноценного решения, и фактически сторонники адаптивного веб-дизайна считают ее одной из наиболее существенных проблем.

В КУРС ДЕЛА!

Тестирование адаптивных макетов

Поскольку адаптивный дизайн предназначен для ответственности экранам различных устройств просмотра, нужно найти способ просматривать ваши страницы на экранах с различным разрешением. Проще всего протестировать ваши медиазапросы путем просмотра страницы на компьютере при изменении окна браузера. Перетащите край окна браузера, сделав его уже, и посмотрите, что получится в каждой точке останова или на каждом установленном вами медиазапросе. Этот прием работает неплохо, но не все браузеры позволяют уменьшать размер окна до ширины 320 пикселей, характерной для некоторых смартфонов.

Кроме того, браузер Google Chrome содержит простой инструмент, который позволяет имитировать просмотр страницы на различных устройствах с изменением высоты и ширины экрана. Чтобы открыть инструмент

разработчика, нажмите кнопку  и выберите команду меню **Дополнительные инструменты** ▶ **Инструменты разработчика** (**Developer** ▶ **Developer Tools**). Затем нажмите кнопку с изображением смартфона, расположенную в верхнем левом углу панели **Developer Tools** (**Инструменты разработчика**), чтобы открыть новое представление текущей страницы. Используйте раскрывающийся список **Device** (**Устройство**) в верхней части страницы, чтобы выбрать новое устройство, например **Apple iPhone 6** или **Nokia Lumia 520**. По окончании закройте панель **Developer Tools** (**Инструменты разработчика**), чтобы вернуться к обычному режиму просмотра веб-страницы.

Существует также несколько веб-инструментов, позволяющих просматривать страницы в окнах разных размеров. Сайт responsivepx.com позволяет вве-

сти URL-адрес страницы во Всемирной паутине, затем указывать различные параметры ширины и высоты экрана. Этот сайт открывает страницу во фрейме установленной ширины. Браузер применяет любые медиа-запросы, поэтому вам предоставляется возможность просматривать результат их выполнения.

Онлайн-инструмент responsinator.com предоставляет ряд предустановок ширины окна, соответствующих таким популярным устройствам, как iPhone, Samsung Galaxy, iPad, Kindle и т. д. Нужно лишь указать URL-адрес тестируемого сайта, и его страница появится внутри эмулируемых экранов. Ресурс работает только со страницами, опубликованными во Всемирной паутине.

Если у вас есть смартфон или планшет, страницу можно просмотреть непосредственно с вашего ком-

пьютера на этих устройствах, используя приложение Adobe Dreamweaver (tinyurl.com/o92twzj). Команда Предварительный просмотр на устройстве (Device Preview) предоставляет замечательную возможность просматривать дизайн страницы в процессе его разработки. Можно также выгрузить страницу на веб-сервер, подключить к Интернету смартфон и посетить эту страницу, чтобы посмотреть ее. Разумеется, пока вы не обзаведетесь десятком разных моделей смартфонов, вы не узнаете, какое впечатление создается от вашей веб-страницы у тех людей, которые пользуются теми или иными мобильными браузерами.

Наконец, вы можете подписаться на службу Browser-Stack (browserstack.com), которая позволяет увидеть, как будет выглядеть веб-страница в различных операционных системах, браузерах и на разных устройствах, включая смартфоны и планшеты.

К числу других людей и организаций, работающих над проблемой гибких изображений, относится Консорциум W3C. Ознакомиться с наиболее популярными в настоящее время решениями можно на сайте tinyurl.com/q43644b. К сожалению, пока не существует решения, которое бы подошло для всех браузеров, поэтому можно проигнорировать все волнения насчет больших файлов изображений для мобильных систем и ждать, пока не появится решение этой проблемы. Или же можно воспользоваться весьма удачным, но непростым решением, которое называется *адаптивные изображения*. Для передачи изображений подходящего к каждому устройству размера используются команды на языке JavaScript и PHP. То есть браузеры мобильных устройств для своих небольших экранов получают изображения, меньшие по объему, а браузеры компьютеров — более крупные изображения. Широко этот вопрос рассмотрен на сайте adaptive-images.com.

Видео и Flash-контент

При использовании HTML-элемента `video` или внедрении Flash-контента для масштабирования таких элементов вместе с их контейнерами также можно воспользоваться приемом со свойством `max-width`. Добавьте в свою таблицу стилей следующий стиль:

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

К сожалению, он не в состоянии справиться с видеоконтентом, внедренным с помощью элементов `iframe` (а это наиболее часто используемый способ добавления на страницу видеоконтента с сайтов видеохостинга типа YouTube или Vimeo).

Чтобы внедрять видеоконтент с сайта YouTube, прочтите статью tinyurl.com/q2y7k94 или обратитесь к онлайн-инструменту embedresponsively.com. Укажите URL-адрес видеоролика на сайте YouTube или Vimeo, и вы сможете сгенерировать HTML-код, который необходимо добавить в качестве адаптивной версии видеоконтента на своей веб-странице.

В КУРС ДЕЛА!

Когда пиксел уже не пиксел?

Многим пиксел представляется в виде отдельной точки на экране или на мониторе. Это так и есть. Но благодаря новым дисплеям с высокой плотностью пикселей, таким как разработанные компанией Apple Retina-дисплеи, теперь о пикселах нужно иметь двойное представление.

Устройства наподобие iPhone 6 Plus имеют разрешение 1920×1080 пикселей и здесь довольно большое количество пикселей помещено в область, меньшую по размеру, чем у большинства других экранов с таким разрешением. Каждый дюйм экрана составляет 401 пиксел. У компьютерных мониторов обычно приходится около 100 пикселей на дюйм. То есть у экрана iPhone в три раза больше пикселей на дюйм, чем у многих компьютерных мониторов. В результате получаются потрясающе четкие и резкие изображения.

Но в результате этого проблема возникает у веб-дизайнеров. Если для шрифта текста задан размер 16 пикселей, позволяющий ему хорошо смотреться на компьютерном мониторе (высота текста составляет около 0,16 дюйма), то он будет абсолютно нечитаем на смартфоне с Retina-дисплеем, поскольку высота текста составит 0,04 дюйма.

К счастью, браузеры в устройствах с Retina-дисплеями так поступать не будут. Они сделают так, чтобы каждый пиксел занимал на экране такого дисплея *несколько* пикселей и высота текста размером 16 пикселей на самом деле отображалась с использованием *более* 16 пикселей. Смартфоны и другие устройства с Retina-дисплеями различают *пиксели экрана устройства*, то есть точки на экране, и *пиксели веб-страниц*.

Пиксел веб-страницы вычисляется на основе плотности пикселей на экране и расстояния от глаз пользователя до экрана. Поскольку смартфон держат ближе к лицу, чем монитор, элементы на его экране крупнее элементов с таким же размером на мониторе с диагональю, к примеру, 28 дюймов.

На экране смартфона iPhone 1 пиксел веб-страницы фактически представлен 2 пикселями устройства. Поэтому текст высотой 16 пикселей на устройстве фактически занимает высоту 32 пикселя. У разных устройств, например у смартфонов под управлением операционной системы Android, разная плотность пикселей, поэтому для определения количества пикселей экрана, приходящихся на 1 пиксел веб-страницы, используются разные вычисления.

Практикум: адаптивный веб-дизайн

В этом практикуме мы возьмем дизайн, созданный в главе 13 (с добавлением нескольких изображений), и наделим его адаптивными свойствами. Будет продемонстрирован подход, при котором предпочтение отдано дизайну для компьютеров, то есть исходные стили лучше всего будут работать в браузерах компьютеров, а затем к ним будут добавлены медиазапросы, адаптирующие внешний вид под экраны среднего размера (экраны планшетов) и под небольшие экраны (экраны смартфонов).

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого загрузите файлы для выполнения заданий практикума,

расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 15.

Изменение HTML-кода

При адаптации макета под небольшой экран вы превращаете свой трехколоночный дизайн в одноколоночный, где все блоки с контентом следуют друг за другом. Проблема HTML-кода страницы, которая была создана ранее в этой книге, заключается в том, что сначала в нем следует левая боковая панель, поэтому при преобразовании дизайна в одноколоночный эта боковая панель появится ранее основного контента.

Будет лучше, если посетители сайта сначала увидят основной контент, а затем, прокручивая страницу, — дополнительную информацию, находящуюся на бывших боковых панелях. Чтобы получить такой результат, нужно добавить некоторый HTML-код и переместить часть уже существующего кода. Для этого будет использоваться технология, рассмотренная в подразделе «Предотвращение выпадений обтекаемых элементов» раздела «Решение проблем с обтекаемыми элементами» главы 13 (см. рис. 13.4): установка контейнера с основным контентом ранее первой боковой панели с последующей оберткой основного контента и боковой панели в новый `div`-контейнер. Этот новый `div`-контейнер нужно выровнять по левому краю, основной контент внутри него — по правому краю контейнера, а боковую панель — по левому. Такое выравнивание позволит поддерживать единую визуальную компоновку в стиле компьютерных мониторов (слева боковая панель, в центре основной контент, а справа — вторая боковая панель), а для дизайна под смартфоны основной контент будет следовать ранее двух бывших боковых панелей.

1. Откройте файл `index.html`, который находится в папке 15.

Его код в точности соответствует финальной странице из урока главы 13. Сначала нужно переместить основной контент вверх.

2. Найдите в HTML-коде комментарий `<!-- основной контент -->` и выделите его и весь HTML-код ниже, включая закрывающий `ter </article>`.

То есть выделите этот комментарий и весь код ниже до комментария `<!-- вторая боковая панель -->`.

3. Вырежьте текст и поместите его в буфер обмена с помощью меню **Редактировать** ▶ **Вырезать** (**Edit** ▶ **Cut**) (или воспользуйтесь аналогичным способом, предоставляемым вашим редактором HTML-кода).

Далее этот код будет вставлен перед кодом первой боковой панели.

4. Найдите ближе к началу файла элемент `<div class="contentwrapper">`. Добавьте под ним перед комментарием `<!-- первая боковая панель -->` пустую строку и воспользуйтесь командой меню **Редактировать** ▶ **Вставить** (**Edit** ▶ **Paste**), чтобы вставить основной контент ранее кода боковой панели.

Теперь нужно будет добавить `div`-контейнер, чтобы обернуть им основной контент и первую боковую панель.

5. После элемента `<div class="contentWrapper">` добавьте элемент `<div class="columnWrapper">`. HTML-код должен приобрести следующий вид:

```
<div class="contentWrapper">
<div class="columnWrapper">
<!-- основной контент -->
```

Затем нужно закрыть этот div-контейнер.

6. Найдите закрывающий тег `</aside>`, принадлежащий коду первой боковой панели. Он находится непосредственно перед комментарием `<!-- вторая боковая панель -->`. Добавьте тег `</div>` после тега `</aside>`, чтобы данный фрагмент HTML-кода приобрел следующий вид:

```
</aside>
</div>
<!-- вторая боковая панель -->
```

Пока это будут все изменения, вносимые в HTML-код. Если вы запутались или хотите проверить свою работу, файл с именем `new-source-order.html`, содержащий все эти изменения HTML-кода, можно найти в папке 15.

Если просмотреть эту страницу в браузере, можно увидеть трехколоночный дизайн, но с основным контентом слева и первой боковой панелью в центре (рис. 15.4). Это проблему можно решить с помощью дополнительного CSS-кода.

7. Откройте файл `styles.css`, который находится в папке 15.

В нем содержится код каскадной таблицы стилей, созданный во время изучения предыдущей главы. Сначала нужно добавить новый стиль для контейнера колонок и выровнять его по левому краю, чтобы он располагался рядом с правой боковой панелью.

8. Добавьте ближе к концу файла перед стилем `.sidebar1` следующий код:

```
.columnWrapper {
  float: left;
  width: 80%;
}
```

Значение свойства `width` здесь формируется из ширины левой боковой панели (20 %) и основной колонки (60 %). На самом деле здесь создается двухколоночный дизайн: первую колонку представляет ранее созданный элемент `div`, а вторую — правая боковая панель. Контейнер основного контента и левая боковая панель, по сути, являются двумя колонками внутри `div`-контейнера, в которую заключена эта колонка, то есть вложенными колонками, как уже демонстрировалось на рис. 13.4.

Далее нужно будет настроить обтекаемые элементы и изменить ширину основного контента и первой боковой панели.

9. Измените значение свойства `width` в стиле `.sidebar` на 25%. Код стиля должен приобрести такой вид:

```
.sidebar1 {
  float: left;
```

```
width: 25%;
padding: 0 20px 0 10px;
}
```

Изначально эта боковая панель занимала 20 % всей ширины страницы, но теперь, когда она находится внутри div-контейнера, оборачивающего колонки, нужно подогнать ширину колонок боковой панели и основного контента, чтобы они умещались в 80 % от ширины страницы, выделенных контейнеру колонок. То есть указанное процентное отношение относится не ко всей странице, а к контейнеру колонок, который занимает только 80 % ширины страницы.

Чтобы вычислить новое процентное отношение, нужно взять предыдущее значение — 20 %, разделить его на ширину контейнера — 80 %, а затем умножить результат на 100. Результат деления 20 на 80 равен 0,25. Умножение этого результата на 100 позволяет получить значение 25 %. Та же технология должна быть применена и к изменению ширины основного контента.



Рис. 15.4. Страница по-прежнему имеет трехколоночный дизайн, но теперь основной контент находится в левой части окна, а левая боковая панель находится в центре. Вернуть эти колонки на их исходные места можно с помощью пары простых стилей

10. Найдите стиль `.main` и присвойте свойству `float` значение `right`, а свойству `width` — значение 75%, придав стилю следующий вид:

```
.main {
float: right;
width: 75%;
padding: 0 20px;
```



```
border-left: dashed 1px rgb(153,153,153);
border-right: dashed 1px rgb(153,153,153);
}
```

Теперь элемент выровнен по правому краю, поэтому он появляется справа от первой боковой панели, в результате чего основной контент снова помещается в центр. Ширина вычисляется путем деления старого значения ширины элемента на значение ширины его контейнера: $60 / 80 = 0,75$, или 75 %. Если сохранить файлы и просмотреть файл `index.html` в браузере, он должен выглядеть так, как показано на рис. 15.5.

ПРИМЕЧАНИЕ

Дизайн страницы в этом примере не станет шире 1200 пикселей благодаря свойству `max-width`, которому присвоено значение `1200px` в групповом селекторе `nav ul, header h1, footer p, .contentWrapper`. Если нужно, чтобы страница заполняла окно браузера независимо от его ширины, удалите объявления `max-width: 1200px`; из этого группового селектора.

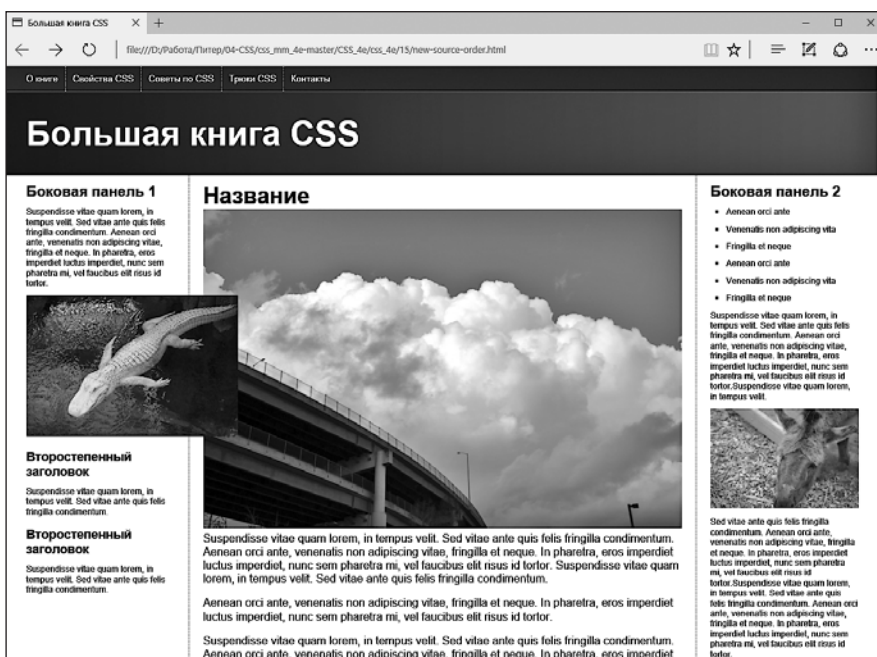


Рис. 15.5. Возвращаясь к исходному состоянию: к трехколоночному дизайну. Но теперь порядок следования исходного HTML-кода страницы больше подходит для создания одноклоночного дизайна для смартфонов. Некоторые изображения не помещаются в отведенное для них пространство и выпадают за пределы колонок. Далее вы займетесь устранением этого недостатка

Адаптивные изображения

Вы успешно справились с созданием резинового макета. Если теперь просмотреть страницу и изменить размеры окна, можно увидеть, что по мере уменьшения раз-

мера окна браузера колонки становятся меньше. К сожалению, некоторые изображения шире колонок и выпадают за их пределы. Как уже ранее упоминалось, составной частью адаптивного веб-дизайна является еще и придание изображениям возможности адаптации к изменениям ширины окна браузера. Для реализации такой возможности нужно немного дополнить CSS-код и убрать часть HTML-кода.

1. Перейдите к файлу `styles.css` в редакторе HTML-кода. В нижней части таблицы стилей добавьте стиль для элемента `img`:

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

Этот стиль ограничивает максимальную ширину изображения 100 % от ширины контейнера. Следовательно, для колонки, имеющей на экране ширину 200 пикселей, изображение будет шириной 200 пикселей. Если посетитель изменит размеры окна браузера и колонки станут меньше, то изображение также уменьшится, чтобы поместиться в колонке.

Но все это будет функционировать при условии удаления свойств изображения `width` и `height` из HTML-кода.

2. Удалите для каждого из четырех элементов `img`, имеющихся в файле `index.html`, атрибуты `height` и `width`.

У нас имеются четыре изображения — `clouds.jpg`, `jellyfishy.jpg`, `gator.jpg` и `mule.jpg`. Удалите из соответствующих элементов `img` атрибуты `width` и `height`. Например, код:

```

```

должен принять вид:

```

```

Если сохранить CSS- и HTML-файлы и просмотреть страницу в браузере, можно будет увидеть, что по мере того, как окно браузера уменьшается, изображения меняют свой размер. Но два изображения в основной колонке все же слишком большие. Использование объявления `max-width: 100%` для элемента `img` приводит к созданию гибких изображений, но во многих случаях вам не нужно, чтобы изображения становились такими же по ширине, как и сама колонка. В случае с двумя изображениями в основной колонке они будут в целом лучше смотреться, если займут только половину размера основной колонки. Этого можно добиться путем применения к этим элементам определенных классов и добавления стилей.

3. Добавьте в конце файла `styles.css` следующие стили:

```
img.half {  
    max-width: 50%;  
}  
img.left {  
    float: left;  
    margin: 0 10px 10px 0;
```

```
}  
img.right {  
  float: right;  
  margin: 0 0 10px 10px;  
}
```

Первый стиль присваивает свойству `max-width` значение 50%, что соответствует половине ширины колонки. Другие два стиля выравнивают элемент изображения по левому или правому краю колонки, позволяя тексту обтекать изображения, и создают небольшие поля. Чтобы воспользоваться этими стилями, к элементам изображений нужно добавить имена классов.

4. Найдите в файле `index.html` элемент `img` изображения `clouds.jpg` — `` — и добавьте код `class="half right"`, чтобы получился следующий код:

```

```

Добавление к элементу более одного класса — довольно устоявшаяся и полезная практика. Таким образом можно объединять простые модульные стили для создания более сложных конструкций дизайна. В данном случае класс `half` изменяет размер изображения, а класс `right` выравнивает его по правому краю. Простым изменением значения `right` на `left` можно выровнять изображение по левому краю колонки, сохраняя ограничение размера, заданное в стиле `half`.

5. Добавьте в элемент изображения `jellyfish.jpg` атрибут `class="half left"`, придав HTML-коду следующий вид:

```

```

Теперь изображения получили нужные размеры, и страница должна приобрести вид, показанный на рис. 15.6.

По желанию можно создать дополнительные стили для других размеров и даже указать размеры в пикселах, позволяя нужным изображениям изменять размеры до их полной ширины, но не более.

Добавление стилей для планшетов

На данный момент дизайн страницы не меняется на всех устройствах просмотра: компьютерных мониторах, планшетах и смартфонах. Это резиновый дизайн, который меняет размер контента, чтобы уместить его в окне браузера. Но в определенный момент боковые панели становятся настолько узкими, что текст на них уже не прочитать. Первый добавляемый вами медиазапрос будет нацелен на устройства с экранами, разрешение по горизонтали которых находится в диапазоне 480–768 пикселей с перемещением правой боковой панели в нижнюю часть страницы, превращая трехколоночный дизайн в более удобочитаемый двухколоночный.

1. Откройте в редакторе HTML-кода файл `styles.css`. Перейдите в конец файла и добавьте следующий код:

```
@media (min-width: 481px) and (max-width:768px) {  
}
```

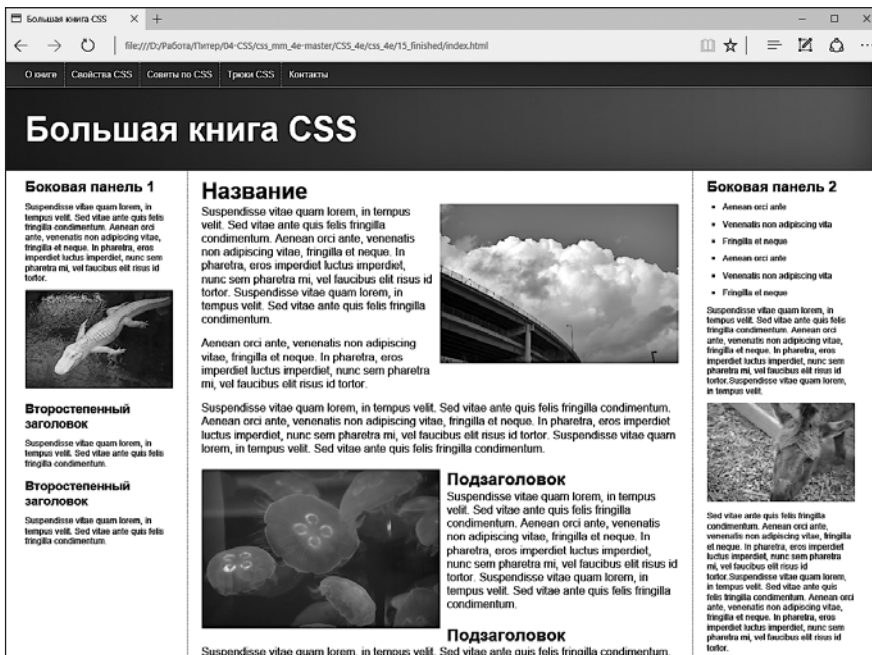


Рис. 15.6. Создание адаптивных изображений не составляет особого труда. Нужно лишь убрать из HTML-кода атрибуты высоты и ширины изображений и присвоить значение свойству `max-width`. Если к изображениям нужно применить другие размеры, создайте классы с требуемыми значениями свойства `max-width` и примените их к элементам `img`

Это первый медиазапрос. Он предназначен для устройств с экранами, имеющими разрешение по горизонтали не менее 481 и не более 768 пикселей. В следующем упражнении этого практикума будет создан медиазапрос для устройств, экранное разрешение по горизонтали которых не превышает 480 пикселей. Поэтому данный медиазапрос исключает такие устройства, как и устройства с экранным разрешением по горизонтали, превышающим 768 пикселей. Иными словами, любые стили, помещаемые в этом разделе таблицы, не будут применяться к браузерам компьютеров (пока вы не уменьшите их окно до слишком узкого) и большинства смартфонов.

Первое, что нужно сделать, — это удалить свойство `float` из стиля правой боковой панели.

2. Внутри медиазапроса, добавленного при выполнении предыдущего шага, создайте следующий стиль:

```
@media (min-width: 481px) and (max-width:768px) {
    .sidebar2 {
        float: none;
        width: auto;
    }
}
```

Этот код отменяет выравнивание боковой панели и присваивает свойству `width` значение `auto` (замещая значение `20%` из стиля `.sidebar2`, ранее определенного в данной таблице). Но этот стиль не заставит боковую панель опуститься ниже двух других колонок. Поскольку `div`-контейнер колонок выровнен по левому краю, вторая боковая панель будет обтекать его. Вам нужно с помощью свойства `clear` освободить боковую панель от обтекания.

3. Отредактируйте стиль, созданный на предыдущем шаге, добавив в него код, выделенный полужирным шрифтом:

```
.sidebar2 {  
  float: none;  
  width: auto;  
  clear: both;  
  border-top: 2px solid black;  
  padding-top: 10px;  
}
```

Свойство `clear` (рассмотренное в главе 7) опускает эту боковую панель ниже двух других колонок. Кроме этого, здесь добавляется характерная верхняя граница для еще большего визуального отделения этого контейнера от верхних колонок.

Теперь, если просмотреть страницу и уменьшить ширину окна браузера до 768 пикселей, вы увидите, что две колонки не распространяются на всю ширину страницы (рис. 15.7). Причина в том, что ранее на шаге 8 предыдущей последовательности действий для контейнера этих колонок было установлено значение ширины, равное `80%` ширины страницы. Вам нужно сбросить этот стиль в медиазапросе для планшетных устройств.

4. Добавьте после стиля `.sidebar2` следующий код:

```
.columnWrapper {  
  width: auto;  
}
```

Убедитесь в том, что этот стиль находится внутри медиазапроса. Он сбрасывает ширину контейнера колонок, присваивая свойству `width` значение `auto` (то же самое, что и `100%`), поэтому распространяется на всю ширину страницы. Поскольку теперь у нас только две колонки, правую границу, применявшуюся к колонке основного контента, можно удалить.

5. Добавьте еще один стиль после класса `.columnWrapper`. Весь медиазапрос должен приобрести следующий вид (изменения выделены полужирным шрифтом):

```
@media (min-width: 481px) and (max-width:768px) {  
  .sidebar2 {  
    float: none;  
    width: auto;  
    clear: both;  
    border-top: 2px solid black;  
    padding-top: 10px;  
  }  
  .columnWrapper {
```

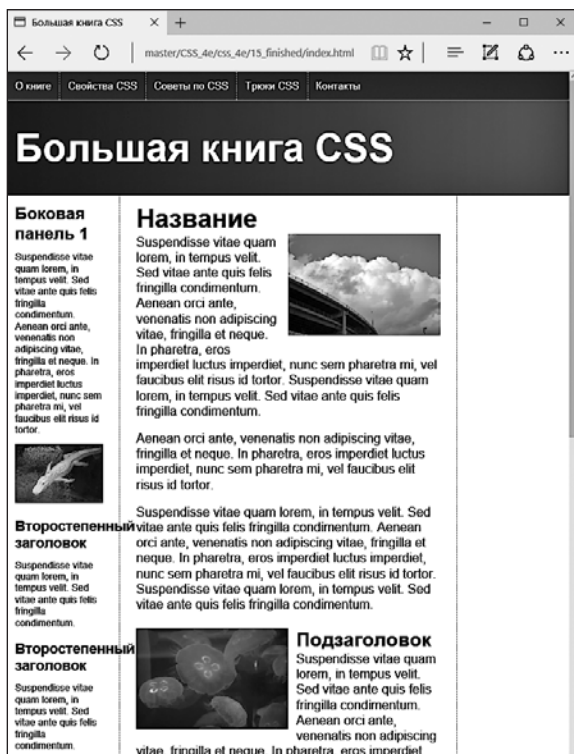


Рис. 15.7. От трехколоночного дизайна к двухколоночному с помощью одного медиазапроса и трех простых стилей

```
width: auto;
}
.main {
border-right: none;
}
}
```

- Сохраните файл `styles.css` и просмотрите файл `index.html` в браузере. Уменьшайте ширину окна браузера до появления только двух колонок.

Страница должна иметь вид, показанный на рис. 15.8.

Добавление стилей для смартфонов

И наконец, настал черед добавить стили для устройств, разрешение экрана которых по горизонтали составляет менее 480 пикселей.

- Откройте в редакторе HTML-кода файл `styles.css`. Перейдите в конец файла и добавьте еще один медиазапрос:

```
@media (max-width:480px) {
}
```

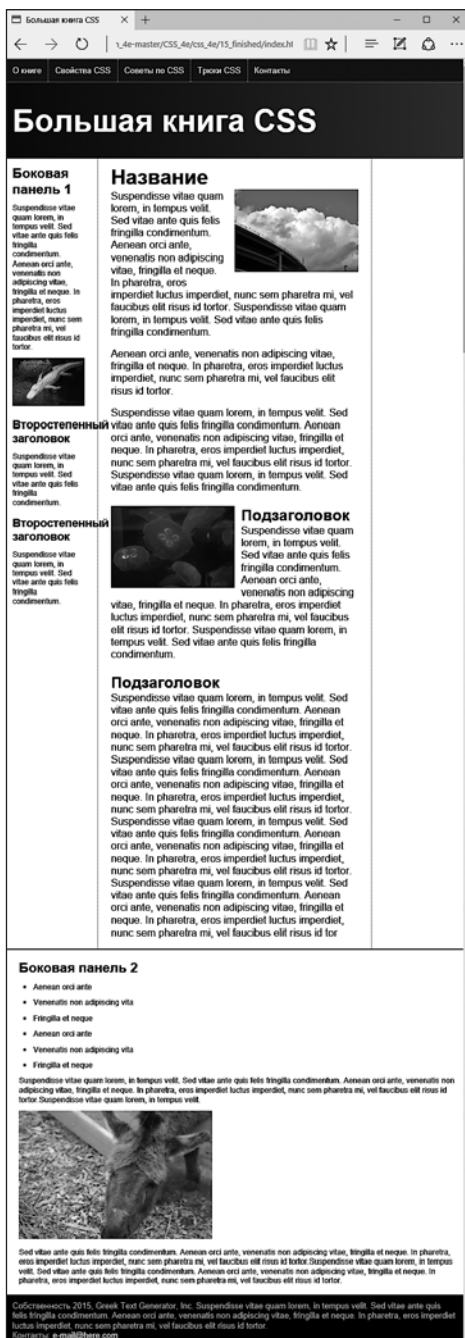


Рис. 15.8. Теперь дизайн зависит от ширины окна браузера — три колонки превращаются в две, а вторая боковая панель смещается вниз. Ширина контента страницы соответствует ширине окна браузера. Это гарантирует, что колонки будут обладать достаточной шириной для удобства чтения даже на устройствах с небольшим экраном

Укажите этот медиазапрос после предыдущего. Теперь мы нацелимся на устройства, разрешение экрана которых по горизонтали составляет менее 480 пикселей. Сначала нужно будет *линеаризовать* дизайн, то есть удалить все обтекаемые элементы, чтобы контент выстроился друг за другом в одну колонку для повышения его читабельности на небольшом экране.

2. Внутри этого медиазапроса добавьте следующий стиль:

```
.columnWrapper, .main, .sidebar1, .sidebar2 {  
  float: none;  
  width: auto;  
}
```

Он удалит обтекаемые (то есть выровненные) элементы из контейнера, обертывающего колонки, и из всех колонок. Присвоение свойству `width` значения `auto` позволяет элементам заполнять свои контейнеры полностью. Таким образом, теперь три колонки становятся простыми блочными элементами, располагаемыми друг за другом, с основным контентом в начале и двумя боковыми панелями следом.

Область основного контента все еще имеет границы, создаваемые стилями, ранее добавленными в CSS-файл. В трехколоночном дизайне эти левые и правые границы служили визуальными разделителями колонки основного контента и левой/правой боковых панелей. В этом последовательном дизайне они не нужны, поэтому их можно удалить. С другой стороны, желательно добавить визуальное разделение между блоками контента, которые теперь следуют друг за другом.

3. После только что добавленного стиля (внутри медиазапроса `@media (max-width:480px) {}`) введите следующий код:

```
.main {  
  border: none;  
}  
.sidebar1, .sidebar2 {  
  border-top: 2px solid black;  
  margin-top: 25px;  
  padding-top: 10px;  
}
```

Представленный здесь групповой селектор добавляет границу над каждой боковой панели, а также верхние поле и отступ для четкого разделения различных областей контента страницы.

Сохраните CSS-файл и просмотрите в браузере файл `index.html`. Нажав и удерживая кнопку мыши на краю окна браузера, перетащите мышью так, чтобы уменьшить ширину окна до 480 пикселей. Теперь вы увидите дизайн, состоящий из одной колонки. Если просмотреть страницу на смартфоне, можно заметить, что заголовок выглядит слишком крупным (рис. 15.9). Кроме того, элементы навигации смотрятся не очень хорошо. Далее мы займемся устранением этих проблем.



Рис. 15.9. Даже если получена подходящая для экрана смартфона ширина страницы, нам еще есть чем заняться. Например, заголовком, который больше подходит по размеру для браузера компьютера, а на экране смартфона выглядит огромным, занимая слишком много полезного пространства

4. Добавьте внутри медиазапроса `@media (max-width:480px)` { стиль для элемента `h1`:


```
header h1 {
  font-size: 1.5em;
}
```

Этот стиль уменьшает размер шрифта текста в элементах `h1`, чтобы он поместился на одной строке. Уменьшение размера заголовков при преобразовании дизайна для просмотра на небольших экранах смартфонов зачастую отнюдь не лишено смысла. Можно также прийти к выводу, что для повышения читабельности следует увеличить размер шрифта основного текста. Для настройки дизайна страницы для смартфонов существует множество способов, а каким должен быть наилучший внешний вид — решать вам.

Теперь займитесь панелью навигации. Пунктирные границы, разделяющие кнопки, смотрятся плохо и сбивают с толку. Кроме того, кнопки выровнены по левому краю, создавая при этом несбалансированную асимметрию. Они будут лучше смотреться по центру и без границ.

5. После стиля `header h1`, созданного на предыдущем шаге, добавьте следующий код:

```
nav {
  text-align: center;
}
```

Элементы навигации содержатся внутри HTML5-элемента `nav`. Присвоение свойству `text-align` значения `center` позволяет центрировать весь текст, находящийся внутри элемента `nav`. Но элементы навигации в данный момент выровнены по левому краю, поэтому это только первый шаг. (Использование обтекаемых элементов для создания горизонтальной панели навигации подробно рассмотрено в разделе «Создание панелей навигации» главы 9.)

6. Добавьте после стиля `nav` еще два стиля:

```
nav li {
  float: none;
  display: inline-block;
}

nav a {
  float: none;
  display: inline-block;
  border: none;
}
```

Вы отменяете выравниваете как элементов списка, так и самих ссылок. Кроме того, вы превращаете эти элементы в строчные блоки. Как уже упоминалось, использование строчных блоков в элементах навигации позволяет размещать ссылки рядом друг с другом, сохраняя при этом отступы и поля. Кроме того, применение строчных блоков — единственный способ центрирования этих кнопок.

7. В окончательном варианте код медиазапроса `@media (max-width:480px)` { должен иметь следующий вид:

```
@media (max-width:480px) {
  .columnWrapper, .main, .sidebar1, .sidebar2 {
    float: none;
    width: auto;
  }
  .main {
    border: none;
  }
  .sidebar1, .sidebar2 {
    border-top: 2px solid black;
  }
}
```

```

margin-top: 25px;
padding-top: 10px;
}
header h1 {
font-size: 1.5em;
}
nav {
text-align: center;
}
nav li {
float: none;
display: inline-block;
}

nav a {
float: none;
display: inline-block;
border: none;
}
}

```

Сохраните все ваши файлы и просмотрите файл `index.html` в браузере. Измените размер окна браузера, чтобы его ширина стала меньше 480 пикселей, и изучите дизайн, предназначенный для смартфонов. Увеличивайте размер окна, пока в поле зрения не появится дизайн, состоящий из двух колонок, а затем продолжайте изменение, пока не увидите версию для компьютеров, имеющую три колонки. Вы должны видеть дизайны, показанные на рис. 15.10 и 15.11. Полную версию файлов этого практикума вы найдете в папке `15_finished`.

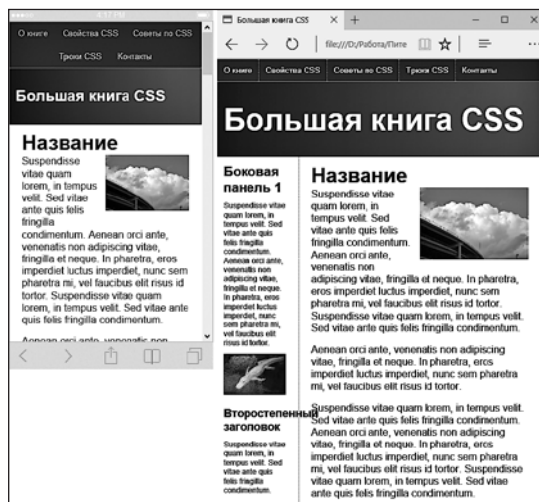


Рис. 15.10. Одна и две колонки, и все на одной веб-странице. Благодаря адаптивному дизайну отдельную веб-страницу можно преобразовать так, чтобы она выглядела наилучшим образом на экране устройства с любым разрешением

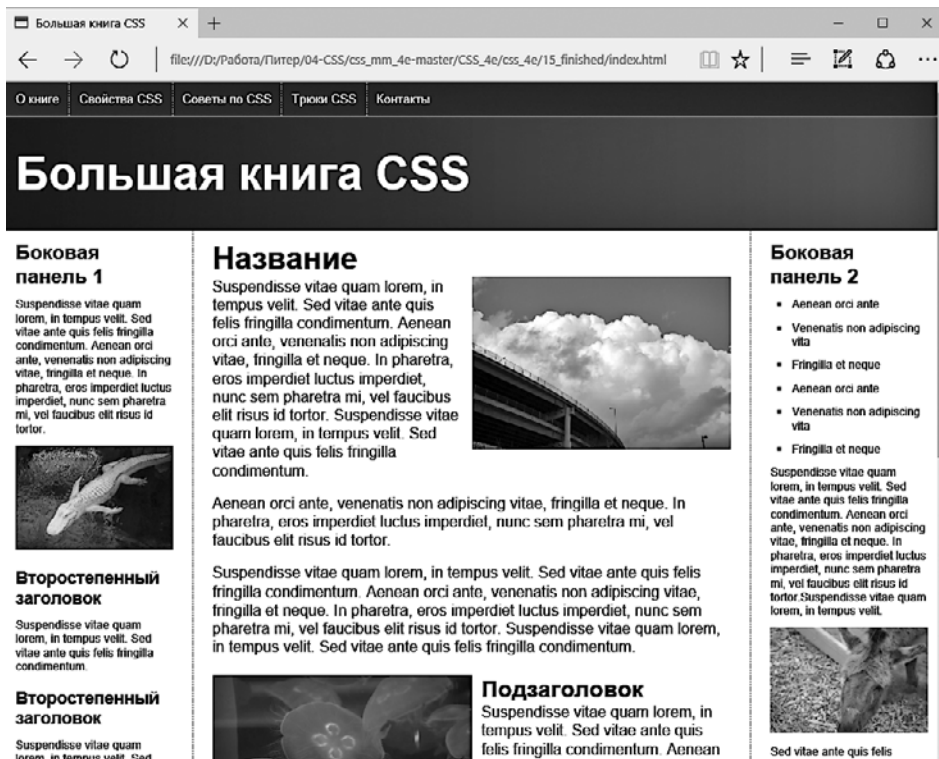


Рис. 15.11. Три колонки на одной веб-странице. Благодаря адаптивному дизайну отдельную веб-страницу можно преобразовать так, чтобы она отлично выглядела на экране устройства с любым разрешением

16 Система модульной верстки Skeleton

Вы уже узнали несколько способов верстки веб-страниц, например, как с помощью макета на основе обтекаемых элементов создать несколько колонок контента, расположенных рядом друг с другом, и адаптивный дизайн, который изменяет компоновку блоков контента в зависимости от размеров экрана устройства, использующегося для просмотра сайта. И если ваш сайт просматривается на экранах самых разных форм и размеров, вам нужен максимально возможный контроль над ним.

В этой главе более подробно рассматриваются концепции, с которыми вы уже познакомились в предыдущих главах, и рассказывается о максимально точном, адаптивном способе компоновки дизайна страниц — модульном, с помощью систем модульной верстки. Система модульной верстки предоставляет модульную сетку — структуру расположения строк и колонок, используемых для организации блоков контента. В отношении каскадных таблиц стилей модульная сетка представляет собой шаблонную таблицу стилей, которая позволяет веб-дизайнерам легко добавлять модульную сетку на страницы своего сайта. Модульные сетки основаны на именах классов, добавленных в элементы `div` HTML-кода страницы, и позволяют создавать соответствующие колонки необходимых размеров.

Принцип модульной сетки

Сетки имеют долгую историю в области графического дизайна и предпечатной подготовки. Существуют различные теории о том, как наилучшим образом организовать контент с помощью сеток, чтобы создать красивый дизайн, но большинство из них полагаются на неизменный печатный дизайн и не подходят для гибкой среды браузеров.

В веб-дизайне сетка действует как способ организации контента в строки и колонки, для чего используются столбцы модульной сетки одинаковой ширины. Страница по ширине делится на определенное количество *столбцов*¹ *модульной сетки*, которые легко группируются для создания *колонок контента* различной ширины. Распространенное число столбцов модульной сетки — 12, поскольку 12 легко де-

¹ В этой книге вертикальная единица модульной сетки называется столбцом вместо колонки, чтобы избежать путаницы с привычными колонками контента.

лится на 2, 3 и 4. Поскольку сетки используют определенное количество столбцов, колонки контента, как правило, прекрасно выровнены и имеют отличный вид, используя согласованные размеры (рис. 16.1).



Рис. 16.1. Многие сайты используют модульные сетки, чтобы размещать контент на странице с помощью строк и колонок предустановленной ширины

Например, веб-страница, показанная на верхнем изображении рис. 16.1, организована в строки и колонки с помощью модульной сетки. Страница выглядит сбалансированной и выровненной, поскольку колонки имеют согласованную ширину. В этом примере сетка состоит из 12 столбцов, а страница разделена на пять строк. Первая строка содержит два блока: первый блок, ширина которого составляет четыре столбца, отображает название компании; и второй, в котором находится панель навигации шириной восемь столбцов. В большинстве модульных сеток между колонками всегда есть средник — пустое пространство. В этом случае между двумя блоками расположен один такой средник.

Вторая и третья строки содержат по одному блоку шириной 12 столбцов. Четвертая строка состоит из трех блоков по четыре столбца каждый. Между каждой парой блоков используется средник. Последняя строка состоит из двух блоков, ширина одного из них равна четырем столбцам, а второго — восьми (так же как в первой строке).

Как вы, возможно, догадались, столбцом является не какая-либо конкретная величина. Это относительный показатель, как и процентное значение. В самом деле в большинстве модульных сеток ширина столбцов определяется с помощью процентных значений. Таким образом, в данном примере, использующем сетку из 12 столбцов, один столбец составляет около $1/12$ от общей ширины страницы — точное значение изменяется в зависимости от размера средника в строке. Как было сказано, относительная ширина — ключ к созданию адаптивного дизайна, поэтому на экране блок шириной три столбца будет менять размер в зависимости от ширины окна браузера. Если вы измените размер окна браузера, точная ширина блока в пикселах будет изменяться, но его относительная ширина всегда будет равняться трем столбцам.

И в этом заключается прелесть модульных сеток. Помните, какие математические вычисления необходимо было проводить для расчета процентного значения ширины контейнера `div` при использовании адаптивного дизайна (см. раздел «Гибкие сетки» главы 15)? Эти действия уже выполнены за вас и внесены в CSS-файл модульной сетки (также называемый *CSS-фреймворком*). Кроме того, как вы можете видеть на рис. 16.1, использование согласованных размеров столбцов позволяет красиво разместить элементы контента. Например, второй блок в четвертой строке начинается в той же позиции слева, что и второй блок в нижней строке, поэтому они идеально выровнены. Более того, ширина блоков соответствует тому или иному количеству столбцов. К примеру, три блока в четвертой строке имеют одинаковую ширину, поскольку их размер вычисляется на основе количества столбцов в модульной сетке.

Структурирование HTML-кода под модульную сетку

Независимые веб-дизайнеры и крупные компании, такие как Twitter, создают десятки систем модульной верстки, чтобы выбрать из них лучшие. Вы можете прочитать о них во врезке далее в этой главе. Эти системы модульной верстки основаны на одном или нескольких CSS-файлах, разработанных для создания колонок, как описано в предыдущем разделе. Большинство из этих систем модульной верст-

ки используют аналогичный подход к структурированию HTML-кода — опираясь на ряд вложенных элементов `div`, образуют три различных типа элементов страницы.

- **Контейнеры.** Контейнер `div` содержит одну или несколько строк. Он помогает установить ширину всей модульной сетки и часто использует свойство `max-width`, чтобы предотвратить чрезмерное расширение контента на больших мониторах. Контейнеры обычно располагаются по центру окна браузера.
- **Строки.** Строки — это еще один элемент `div`, помещенный в контейнер. В строке находятся другие элементы `div`, содержащие колонки.
- **Колонки.** Колонки определяются элементами `div` в строке. Каждая строка содержит одну или несколько колонок.

Чтобы определить контейнер, строку или колонку, необходимо добавить имя класса к каждому элементу `div`. В различных системах модульной верстки используются разные имена, но многие дизайнеры применяют вариации слов *container*, *row* и *column*. Например, вы хотите начать с создания страницы с двумя строками. Первая строка содержит две колонки: одну колонку для логотипа, а другую — для навигации по сайту. Вторая строка содержит единственную колонку для отображения текста публикации. HTML-разметка этой модульной сетки может выглядеть следующим образом:

```
<div class="container">
  <!-- строка #1 -->
  <div class="row">
    <!-- колонка в 3 столбца сетки -->
    <div class="three columns">
      <!-- логотип -->
    </div>
    <!-- колонка в 3 столбца сетки -->
    <div class="nine columns">
      <!-- навигация -->
    </div>
  </div>
  <!-- строка #2 -->
  <div class="row">
    <!-- колонка в 12 столбцов сетки -->
    <div class="one columns">
      <!-- текст публикации -->
    </div>
  </div>
</div>
```

Этот код не делает ничего сверхъестественного. На самом деле он очень похож на HTML-код, используемый в технике верстки, описанной в главе 13. Основное отличие заключается в том, что за вас были созданы стили CSS, используемые для верстки страницы, и тем самым экономится уйма времени. Все, что вам нужно сделать, — это скачать CSS-файлы системы модульной верстки, прикрепить таблицу стилей к веб-странице и структурировать HTML-код страниц в соответствии с правилами, определенными для выбранной системы модульной верстки. В этой главе

используется система модульной верстки Skeleton, но вы можете выбрать любую другую по своему усмотрению (см. врезку далее).

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Основные системы модульной верстки

В этой главе используется простая, но мощная система модульной верстки под названием Skeleton (getbootstrap.com). Но она не одна в своем роде.

- Simple Grid (tinyurl.com/d3vwkuu) — еще одна простая система модульной верстки. Это *каркас* — она не обеспечивает создание никакого CSS-кода, кроме модульной сетки. Эта система адаптивная, то есть дизайн страницы автоматически подстраивается при изменении ширины окна браузера. Поэтому, например, на мобильном устройстве любая многоколоночная страница мгновенно изменяется и становится одноколоночной, что упрощает чтение.
- Pure.css (purecss.io) — нечто большее, чем просто CSS-*фреймворк*. Эта система позволяет создавать стили CSS, с помощью которых можно формировать не только сетки, но и оформлять кнопки, таблицы, меню и формы. Pure.css — проект компании Yahoo!.
- Foundation (foundation.zurb.com) — другой адаптивный CSS-*фреймворк*. Эта система более популярна, но и сложнее, чем большинство других систем модульной верстки. Она прекрасно подходит для создания веб-приложений и интерактивных сайтов, поскольку содержит не только таблицы каскадных стилей, но и JavaScript-код для создания раскрывающихся списков, графических элементов, подсказок и модальных диалоговых окон.
- Bootstrap (getbootstrap.com) — наиболее популярный CSS-*фреймворк* (этим фактом объясняется схожесть многих сайтов). Созданный компаниями Twitter и Bootstrap, этот *фреймворк* используется в тысячах сайтов. Он включает в себя модульную сетку и не только, как и сервис Foundation. *Фреймворк* Bootstrap содержит правила форматирования кнопок, таблиц, предупреждений, значков, ярлыков и даже компонент *jumbotron* (<http://getbootstrap.com/components/#jumbotron>). Он предназначен для создания контента или информации на сайте, который занимает всю ширину контейнера. Кроме того, *фреймворк* Bootstrap располагает многими компонентами JavaScript, такими как подсказки, карусели и модальные диалоговые окна.

Использование системы модульной верстки Skeleton

В этой главе вы узнаете, как работать со Skeleton — простой, гибкой системой модульной верстки веб-страниц. Вы также изучите несколько дополнительных правил каскадных таблиц стилей, которые призваны помочь с базовым форматированием кнопок, веб-форм и таблиц. Система Skeleton выполняет всю эту работу с помощью 400 строк кода, помещенных в файл небольшого размера. Она создана прежде всего для работы с проектами для мобильных устройств и проста в использовании.

Skeleton — адаптивная система модульной верстки, поскольку при уменьшении ширины окна браузера до 550 пикселей она сворачивает в одну колонку многоколоночную страницу. На экранах смартфонов пространство особенно ценно и отображать текст или любой другой контент в нескольких колонках нецелесообразно — его слишком трудно читать. Поэтому большинство веб-дизайнеров проектов для мобильных устройств для отображения контента используют одну колонку, в которой все элементы `div` расположены один за другим.

Система Skeleton позволяет сверстать сложный сетчатый макет, аккуратно структурировав контент страницы в несколько колонок на планшетных устройствах, ноутбуках и компьютерах. На смартфоне, однако, каскадная таблица стилей Skeleton автоматически удаляет колонки, расположенные рядом друг с другом, преобразуя дизайн страницы в одноколоночный, удобный для чтения. Вам не придется делать ничего; адаптивный дизайн встроен прямо в CSS-файл системы Skeleton.

Чтобы начать работу, посетите сайт Skeleton (getskeleton.com) и нажмите кнопку **Download** (Скачать) (рис. 16.2). Загруженный архив содержит несколько папок и файлов, но все, что вам нужно, — содержимое каталога **CSS**: файл `normalize.css`, который обеспечивает сброс базовых стилей CSS (см. раздел «Устранение конфликтов стилей в браузере» главы 18), чтобы все браузеры отображали HTML-элементы одинаково, и файл `skeleton.css`, содержащий все необходимое, чтобы скомпоновать макет с использованием модульной сетки.

Ниже приведены основные шаги по применению системы модульной верстки Skeleton.

1. Прикрепите CSS-файлы.

Вы можете прикрепить файлы `normalize.css` и `skeleton.css` с помощью следующего кода:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
```

ПРИМЕЧАНИЕ

Базовые стили системы Skeleton используют веб-шрифт Google под названием *Raleway*. Это красивый шрифт, который, к сожалению, не поддерживает кириллицу. Поэтому в рамках примеров данной книги он был заменен на аналогичный с поддержкой кириллицы — *Roboto*. Если вы хотите использовать его на своем сайте, вам необходимо добавить ссылку на файлы этого шрифта. Для этого используйте следующий код на своей веб-странице:

```
<link href="https://fonts.googleapis.com/css?family=Roboto:400,300,700&subset=latin,Cyrillic"
rel="stylesheet" type="text/css">
```

2. Добавьте контейнеры `div`.

Как упоминалось ранее в этой главе, в модульной сетке контейнеры `div` используются для построения строк и колонок. Страница может иметь только один контейнер, но бывают случаи, когда для создания определенного эффекта необходимо несколько контейнеров (подробнее об этом читайте в следующем разделе). Контейнер в системе Skeleton — это обычный элемент `div` с классом `container`:

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

```
</div>
```

Не помещайте какой-либо HTML-код в контейнер `div`, кроме элементов `div` для строк, как вы это сделаете на следующем шаге.

3. Добавьте элементы `div` для строк.

В каждый из контейнеров вам необходимо добавить одну или несколько строк. Это обычные элементы `div`, вложенные в контейнер `div`. Им присваивается класс `row`:

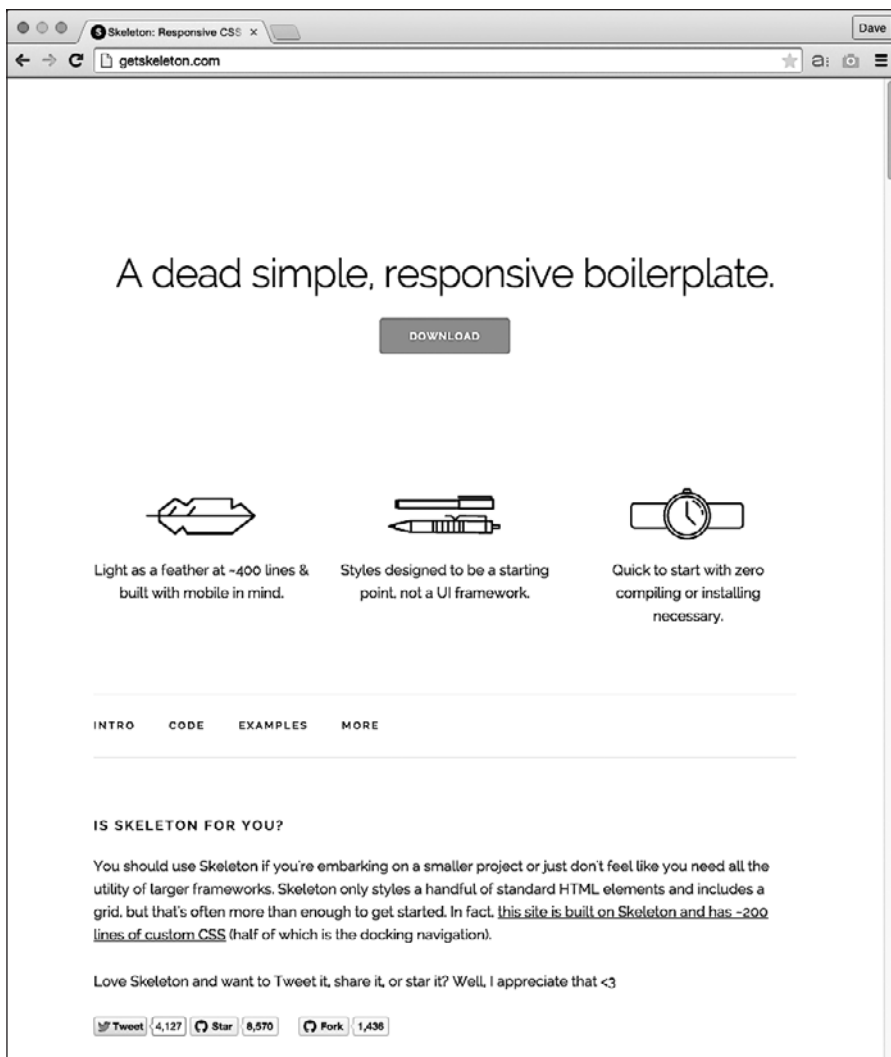


Рис. 16.2. Skeleton — это набор правил CSS, которые предоставляют простую, адаптивную модульную сетку для верстки веб-страниц

```
<div class="container">
  <div class="row">

    </div>
  </div>
```

В один контейнер можно поместить любое количество строк. В самом деле, если вы не используете определенный эффект дизайна, описанный в следующем разделе, можете применять единственный контейнер для страницы, содержащей несколько строк и колонок:

5. Добавьте контент в элементы `div` колонок.

Контейнер и элементы `div` строк — структурные элементы, используемые для создания колонок. Контент, например текст, фотографии и видеоролики, находится внутри каждого из элементов `div` колонок.

6. Создайте собственные стили.

Таблица стилей Skeleton не позволяет ничего, кроме создания базовой структуры вашего контента, то есть *каркаса*. Чтобы придать странице оформление, вам нужно добавить собственные стили, с помощью которых сайт декорируется цветом, шрифтами, фоновыми изображениями и т. д. Я рекомендую для этих целей создать еще один CSS-файл под именем, к примеру, `custom.css` и связать его с веб-страницей после ссылки на файл `skeleton.css`:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
<link rel="stylesheet" href="css/custom.css">
```

Система Skeleton в плане дизайна в первую очередь отдает предпочтение мобильным устройствам (см. раздел «Медиазапросы» главы 15), поэтому для подгонки дизайна под окна браузеров различной ширины вы будете использовать медиазапросы, начиная с нацеленных на узкий экран смартфона (в подразделе «Приоритет мобильных устройств» раздела «Создание и именование колонок» далее вы узнаете, как сделать это).

Создание и именование колонок

В системе Skeleton используется модульная сетка, состоящая из 12 столбцов (рис. 16.3). Вы можете создавать колонки разной ширины, размером от 1 до 12 столбцов. Для создания колонки присвойте два класса элементу `div`. Один класс определяет ширину (`one`, `two`, `three` и т. д.), в то время как другой использует слово `columns`. Имейте в виду, что это не единое имя класса CSS: следует добавить имена двух классов, и они должны быть отделены друг от друга пробелом. Каждое имя относится к соответствующему правилу каскадной таблицы стилей Skeleton.

Обратите внимание на следующие два примера. Скажем, вы хотите создать строку с двумя колонками одинаковой ширины. Все, что нужно сделать, — это добавить два элемента `div`, каждый шириной шесть столбцов, внутри элемента `div` с классом `row`, вложенного в контейнер `div` с классом `container`, как это показано ниже:

```
<div class="container">
  <div class="row">
    <div class="six columns">

    </div>
    <div class="six columns">

  </div>
</div>
</div>
```

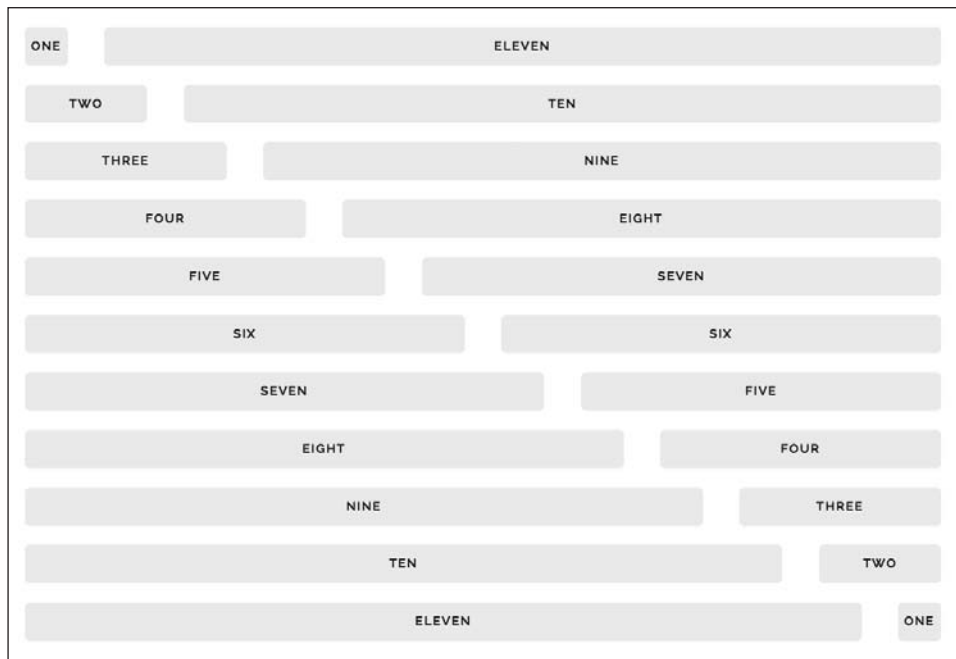


Рис. 16.3. Система модульной верстки Skeleton использует сетку, состоящую из 12 столбцов, которая позволяет создавать комбинации колонок различной ширины. Как видно из рисунка, ширина колонки в один столбец крайне мала, поэтому вряд ли вы когда-нибудь создадите колонку такой ширины. Но поскольку столбцов 12, вы можете создавать различные комбинации колонок разной ширины

Чтобы создать четыре колонки одинаковой ширины, добавьте четыре элемента `div`, по три столбца в ширину каждый:

```
<div class="container">
  <div class="row">
    <div class="three columns">

    </div>
    <div class="three columns">

    </div>
    <div class="three columns">

    </div>
    <div class="three columns">

    </div>
  </div>
</div>
```

Колонки не обязательно должны иметь одинаковую ширину. Тем не менее необходимо убедиться, что сумма всех колонок в строке равна 12 столбцам. Например,

чтобы в строке было создано две колонки, одна из которых имеет ширину четыре столбца, а другая — восемь, код должен выглядеть следующим образом:

```
<div class="container">
  <div class="row">
    <div class="four columns">

      </div>
    <div class="eight columns">

      </div>
  </div>
</div>
```

Одним очень полезным аспектом системы Skeleton является то, что она автоматически (и точно) вычисляет средники между несколькими колонками. То есть вам не нужно вычислять, какое пространство должно быть между двумя колонками. Если в вашей строке только одна колонка, таблица стилей Skeleton не добавит средник; если колонки две, она добавит один средник, чтобы разделить их; а если колонок шесть — будет добавлено пять средников, размеры которых будут вычислены автоматически в CSS-файле системы Skeleton!

СОВЕТ

Для создания колонок шириной в половину, одну треть и две трети от ширины контейнера система Skeleton использует сокращенные имена классов. Для этого укажите значения `half`, `one-third` или `two-thirds` соответственно рядом с именем класса `column`. Например, чтобы создать две колонки, ширина одной из которых равна одной трети от ширины контейнера, а второй — две трети, используйте следующий код:

```
<div class="row">
  <div class="one-third column">

    </div>
  <div class="two-thirds column">

    </div>
</div>
```

Создание разделов во всю ширину окна браузера

Контейнер `div`, добавленный на страницу, содержит строки и колонки. CSS-код таблицы Skeleton не разворачивает стиль `container` так, чтобы контейнер заполнял окно браузера в устройствах просмотра. Ширина контейнера определяется посредством медиазапросов, но никогда не распространяется от края до края окна. В некоторых случаях это допустимо, но, если вы хотите залить фон контейнера цветом или изображением, ваш дизайн будет выглядеть, мягко говоря, не очень хорошо (рис. 16.4, *вверху*). В этом случае необходимо развернуть фон до краев окна браузера (см. рис. 16.4, *внизу*).

К счастью, это очень просто сделать. Поскольку ширина любого блочного элемента, такого как `div`, по умолчанию составляет 100 %, вам необходимо лишь обернуть элементом `div` контейнер `div` и установить цвет фона для внешнего `div` (обертки).

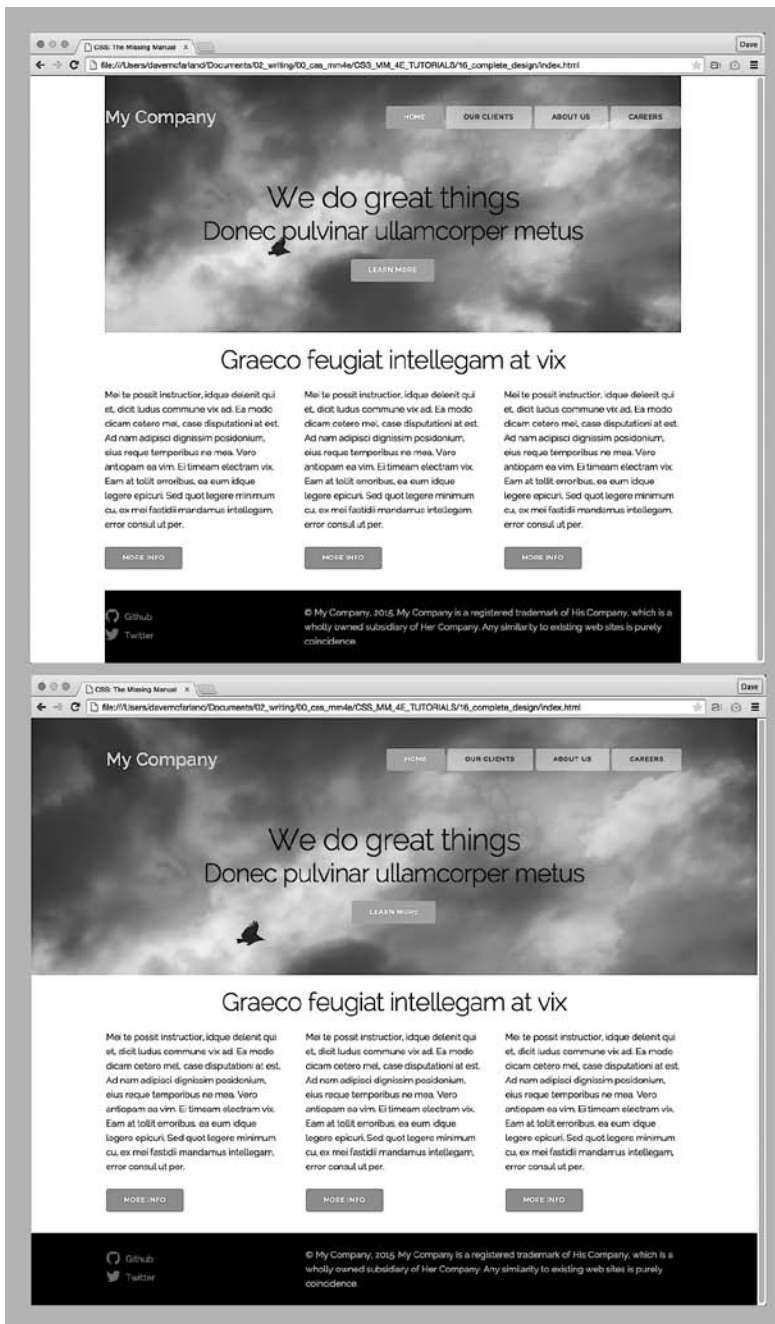


Рис. 16.4. Класс `container` таблицы стилей Skeleton не распространяется от края до края окна браузера (вверху). Если вы назначили фоновое изображение или цвет одному из таких контейнеров, назначение не будет распространяться за рамки контейнера. В большинстве случаев страница выглядит лучше, если фоновое изображение или цвет распространяются до краев окна браузера (внизу)

Например, на нижнем изображении на рис. 16.4 в верхней области страницы расположена фотография, которая заполняет область заголовка между краями окна браузера. Это достигается с помощью следующего HTML-кода для данной области:

```
<div class="section header">
  <div class="container">
    <div class="row">
      <div class="three columns">
        <p class="logo">My Company</p>
      </div>
      <div class="nav nine columns">
        <a class="button button-primary" href="#">Home</a>
        <a class="button" href="#">Our Clients</a>
        <a class="button" href="#">About Us</a>
        <a class="button" href="#">Careers</a>
      </div>
    </div>
    <div class="row action">
      <h1>We do great things</h1>
      <h2>Donec pulvinar ullamcorper metus</h2>
      <a href="#" class="button button-primary">Learn More</a>
    </div>
  </div>
</div>
```

Важной частью является внешний элемент `div`. Он окружает контейнер `div` (строки 2 и 19). Присваивая имя класса этому элементу `div`, вы можете легко установить фоновое изображение с помощью CSS:

```
.header {
  background-image: url(../imgs/header.jpg);
  background-size: cover;
}
```

Теперь, поскольку ширина внешнего элемента `div` всегда равна 100 %, его фон будет всегда заполнять всю ширину окна браузера.

Как уже упоминалось, для всей веб-страницы, возможно, потребуется только один контейнер `div`. Тем не менее вы можете добавлять *различные* фоны для различных строк, тем самым заполняя всю ширину окна.

Например, на нижнем изображении на рис. 16.4 фотография заполняет верхнюю часть страницы, а черная полоса — нижнюю. К различным частям страницы применены два разных фона. В этом случае вам необходимо использовать несколько контейнеров: один для верхней части (фотография), один для средней части (белая область) и один — для нижнего колонтитула (черная полоса). Кроме того, каждый из этих контейнеров должен быть обернут собственным элементом `div`. Эти `div`-элементы заполняют ширину окна браузера, и вы можете настроить стиль каждого из них по отдельности. В практикуме в конце этой главы вы увидите пример создания фона в полную ширину.

Форматирование кнопок

Несмотря на то что Skeleton, по сути, модульная сетка, она позволяет использовать пару забавных, привлекательных стилей для форматирования других типов элементов страницы. В частности, Skeleton создает кнопки с отличным оформлением (рис. 16.5).

CSS-файл Skeleton автоматически форматирует определенные HTML-элементы, чтобы они выглядели как кнопки, изображенные на рис. 16.5. Например, он форматирует HTML-элемент `button`, а также элементы формы типа `submit` или `button` (см. белые кнопки, изображенные в верхней части рис. 16.5).

К примеру, на вашей веб-странице находится следующий HTML-код:

```
<button>Текст кнопки</button>
```

CSS-файл Skeleton автоматически придаст им вид белой кнопки со скругленными углами и серой границей. Теперь, предположим, вы хотите придать такой же стиль другим HTML-элементам. Например, чтобы ряд ссылок, образующих панель навигации, выглядел как кнопки Skeleton. Для этого добавьте имя класса `button` к ссылке:

```
<a href="index.html" class="button">Главная</a>
```

Таблица стилей Skeleton содержит еще один класс, `.button-primary`, который создает синюю подсветку кнопки. Для ее реализации добавьте следующий класс в HTML-код:

```
<button class="button-primary">Текст кнопки</button>
```

Если вы хотите превратить обычную ссылку в подсвеченную кнопку, добавьте два класса: один из них преобразует ссылку в кнопку, а другой подсветит ее:

```
<a href="index.html" class="button button-primary">Главная</a>
```

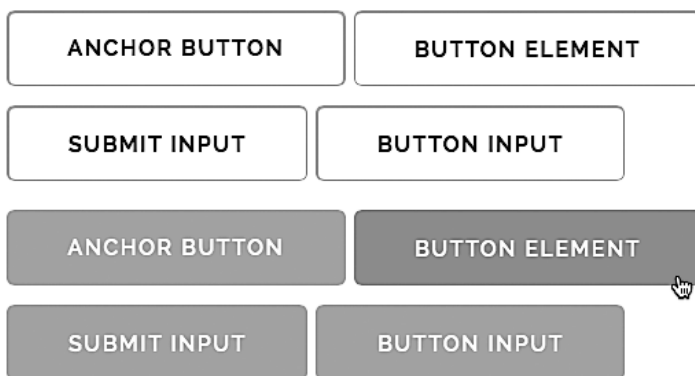


Рис. 16.5. С помощью системы Skeleton очень легко превратить ссылки и элементы формы в простые, привлекательные кнопки

Приоритет мобильных устройств

Таблица стилей Skeleton отдает предпочтение мобильным устройствам. Стратегия Mobile First, о которой вы прочитали в разделе «Медиазапросы» в главе 15, предполагает применение дизайна, который в первую очередь выглядит хорошо на небольших экранах мобильных устройств, таких как смартфоны. Используйте медиазапросы, а затем добавляйте стили, которые улучшают внешний вид страницы на экранах с разным разрешением по горизонтали. Каждый медиазапрос определяет более высокое разрешение крапа по горизонтали, а вы добавляете стили для улучшения дизайна для каждой из этих точек останова (экранное разрешение по горизонтали).

Стратегия Mobile First предполагает использование свойства `min-width` в правиле `@media`. Другими словами, каждый медиазапрос применяет разные стили, если экран обладает тем или иным минимальным разрешением по горизонтали. Например, взгляните на следующий медиазапрос:

```
@media (min-width: 400px) {  
  
}
```

Исходя из него добавленные в данный медиазапрос стили будут применяться только в тех случаях, если окно браузера имеет размер не менее 400 (и более) пикселей в ширину. Если ширина окна или разрешение экрана, к примеру, 320 пикселей, стили внутри этого медиазапроса не будут применены к странице. Подход, основанный на приоритете для мобильных устройств, заключается в применении нескольких правил `@media` с разным значением свойства `min-width`.

При использовании системы Skeleton для создания дизайна по стратегии Mobile First рекомендуется начать с отдельной таблицы стилей. Присвойте ей имя, например `custom.css` или `site.css`, и прикрепите к веб-странице после CSS-файлов системы Skeleton:

```
<link rel="stylesheet" href="css/normalize.css">  
<link rel="stylesheet" href="css/skeleton.css">  
<link rel="stylesheet" href="css/custom.css">
```

Вы сможете добавлять правила в этот CSS-файл, чтобы определить внешний вид сайта, изменив цвета, шрифты, фон и т. д. Для создания дизайна по стратегии Mobile First вы должны начать с определенных базовых медиазапросов. Разработчик системы Skeleton рекомендует настраивать медиазапросы для ряда различных точек останова, например:

```
/* Запросы Mobile First */  
/* стили для любой ширины */  
  
/* стили для ширины крупнее мобильных */  
/* стили для ширины 400 пикселей и выше */  
@media (min-width: 400px) {  
  
}  
  
/* стили для ширины крупнее фэблетов */  
/* стили для ширины 550 пикселей и выше */
```

```
@media (min-width: 550px) {  
  
}  
/* стили для ширины крупнее планшетов */  
/* стили для ширины 750 пикселей и выше */  
@media (min-width: 750px) {  
  
}  
/* стили для ширины крупнее компьютерных мониторов */  
/* стили для ширины 1000 пикселей и выше */  
@media (min-width: 1000px) {  
  
}  
/* стили для ширины крупнее компьютерных HD-мониторов */  
/* стили для ширины 1200 пикселей и выше */  
@media (min-width: 1200px) {  
  
}  
  
}
```

В дизайне по стратегии Mobile First стили, которые применяются для всех устройств (мобильных или нет), добавляются *вне* медиазапросов. Например, если вы хотите применить тот же шрифт, цвет шрифта, цвет фона для элементов страницы независимо от ширины окна браузера, добавьте эти правила CSS здесь.

Кроме того, здесь вы должны добавить стили, которые форматируют ваш сайт для просмотра на мобильном устройстве. Вы можете использовать панель **Developer Tools** (Инструменты разработчика) в браузере Chrome, чтобы просмотреть, как страница будет выглядеть на мобильном устройстве.

После того как мобильная версия будет выглядеть должным образом, расширьте окно браузера (или используйте панель **Developer Tools** (Инструменты разработчика) в браузере Chrome), чтобы увидеть, как ваша страница будет выглядеть в первой точке останова — при ширине экрана, равной 400 пикселям. Если при этой ширине страница выглядит прекрасно, вы можете расширить окно браузера до следующей точки останова (550 пикселей).

Если страница выглядит не очень хорошо, возможно, на ней не хватает воздуха вокруг элементов. В этом случае добавьте новые стили в этой точке останова, пока страница не будет выглядеть хорошо. Помните, любое дополнение, которое вы делаете в медиазапросе, не будет применяться к более ранней точке останова. Так, например, если вы добавите стиль, который увеличивает отступ по периметру заголовков, в медиазапросе со значением свойства `min-width`, равным 550 пикселям, отступы не будут добавлены к заголовкам при размере окна браузера меньше 550 пикселей в ширину. Другими словами, изменения, внесенные в этот медиазапрос, не повлияют на дизайн страницы при ее просмотре на экране смартфона.

Вы можете продолжить процесс просмотра внешнего вида сайта в каждой точке останова, добавляя нужные стили в соответствующие медиазапросы, чтобы улучшить внешний вид страницы при конкретной ширине окна браузера. Ранее приведенный код содержит пять медиазапросов с пятью различными точками останова, но вовсе не обязательно добавлять стили в каждой из них. Вероятно, вам только потребуется настроить внешний вид элементов страницы

для нескольких различных значений ширины окна браузера (разрешения экрана по горизонтали). Кроме того, вы можете обнаружить, что значения свойства `min-width`, используемые в этом коде, не работают с конкретным дизайном. Например, дизайн вашей страницы может выглядеть отлично вплоть до ширины 650 пикселей, но при дальнейшем увеличении элементы страницы не вписываются или выглядят плохо. В этом случае измените значение свойства `min-width` для конкретной точки останова и добавьте нужные стили внутри этого медиазапроса (например, измените значение `550px` на `650px`). Вполне возможно, вам и не понадобятся все эти медиазапросы, приведенные в коде, поэтому вы можете удалить их из таблицы стилей.

Лучший способ изучить дизайн по стратегии *Mobile First* — создать его самостоятельно. Сейчас вы получите такую возможность.

Практикум: использование системы модульной верстки

Этот практикум продемонстрирует, как применить модульную сетку к веб-странице с помощью системы модульной верстки *Skeleton*. Вы начнете с некоторых основных элементов HTML-кода, примените необходимые CSS-файлы, добавите и структурируете HTML-код, чтобы создать сетку, и, наконец, напишете собственные правила CSS, чтобы придать странице отличный вид даже при ее просмотре на экранах мобильных устройств.

Чтобы начать обучение, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку **Download ZIP** в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 16.

Добавление сетки

Сначала вам необходимо прикрепить CSS-файлы и добавить некоторый HTML-код, чтобы создать контейнер таблицы со строками и столбцами.

1. В своем редакторе HTML-кода откройте файл `index.html`, расположенный в папке 16.

Это простой HTML-файл. Он содержит разделы заголовка и тела, в котором еще нет HTML-кода. Однако прежде, чем добавить HTML-код, добавьте ссылки на несколько CSS-файлов. В файле уже есть ссылка на веб-шрифт *Google*, и вам необходимо прикрепить файлы таблицы стилей *Skeleton*.

2. В пустой строке сразу после закрывающего тега `</head>` добавьте следующие три строки кода:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/skeleton.css">
<link rel="stylesheet" href="css/custom.css"
```

Первая строка кода загружает файл `normalize.css`. Это очень популярный файл сброса стилей CSS (см. раздел «Устранение конфликтов стилей в браузере» главы 18), который используется во многих проектах. Второй файл — это основной файл системы модульной верстки Skeleton — `skeleton.css`, который содержит CSS-код модульной сетки. Последний файл — `custom.css` — по сути, пуст. Он содержит только несколько медиазапросов без самих стилей, которые применяются для подгонки страниц сайта под экраны различных размеров. Вы будете использовать этот файл для добавления стилей и настройки внешнего вида страницы.

Далее вы добавите контейнер Skeleton.

3. Сразу после комментария `<!-- основной раздел -->` добавьте следующий HTML-код:

```
<!-- основной раздел -->
<div class="container">

</div>
```

Как уже говорилось, таблица стилей Skeleton использует имя класса `container`, чтобы определить элемент `div`, который содержит добавленные вами строки на странице. На странице может использоваться как один, так и любое другое количество контейнеров `div` для строк. Как вы скоро убедитесь, при использовании более чем одного контейнера всегда есть преимущество.

Добавим первую строку.

4. После только что добавленного элемента `div` вставьте еще два элемента `div`, чтобы создать две строки (добавленный текст выделен полужирным шрифтом):

```
<!-- основной раздел -->
<div class="container">
  <div class="row">

  </div>
  <div class="row">

  </div>
</div>
```

Добавленные элементы `div` создают две строки. В первую строку вы добавите две колонки — одну для названия сайта, а вторую — для панели навигации.

5. В первом элементе `div` с классом `row` добавьте следующий HTML-код:

```
<div class="four columns">
  <p class="logo">Моя компания</p>
</div>
<div class="eight columns nav">

</div>
```

Система модульной верстки Skeleton использует модульную сетку в 12 столбцов. Вы можете использовать эти столбцы, чтобы сделать отдельные колонки. В данном

примере вы добавили две колонки: ширина одной из них равна четырем столбцам, а второй — восьми. Визуально первая колонка занимает одну треть, или около 33 % всей ширины контейнера, а вторая — две трети, или около 66 %.

Вторая колонка шире, потому что в ней будет находиться панель навигации. На самом деле обратите внимание, что к данному элементу добавлен класс `nav`. Это не является требованием каркаса `Skeleton`. Вы добавили его сейчас, чтобы использовать позже для форматирования панели навигации.

6. Откройте файл `01-nav.html`. Скопируйте содержащийся в нем HTML-код и вставьте его в элемент `div` с классом `eight columns nav` файла `index.html`, чтобы HTML-код выглядел следующим образом (добавленный текст выделен полужирным шрифтом):

```
<div class="eight columns nav">
  <a href="#">Главная</a>
  <a href="#">Клиенты</a>
  <a href="#">0 нас</a>
  <a href="#">Вакансии</a>
</div>
```

Это простые ссылки. Тем не менее таблица стилей `Skeleton` включает в себя несколько очень красивых стилей для форматирования любого элемента в качестве кнопки. Для этого вы добавите имя класса к элементам `a`.

7. Добавьте атрибут `class` к каждой ссылке и присвойте ему имя `button`:

```
<div class="eight columns nav">
  <a class="button" href="#">Главная</a>
  <a class="button" href="#">Клиенты</a>
  <a class="button" href="#">0 нас</a>
  <a class="button" href="#">Вакансии</a>
</div>
```

Система `Skeleton` также включает в себя специальный класс с именем `button-primary`, который придает кнопке особый вид. Это отличный способ выделить ссылку на текущей странице среди остальных ссылок, чтобы посетители видели, на какой странице они находятся. Страница, над которой вы работаете, является главной, поэтому вы добавите специальный класс к ссылке «Главная».

8. Добавьте имя класса `button-primary` к первому элементу `a`:

```
<div class="eight columns nav">
  <a class="button button-primary" href="#">Главная</a>
  <a class="button" href="#">Клиенты</a>
  <a class="button" href="#">0 нас</a>
  <a class="button" href="#">Вакансии</a>
</div>
```

Осталась последняя строка, к которой необходимо добавить контент. Это просто, поскольку она содержит только одну колонку, заполняющую контейнер по всей ширине.

9. Откройте файл `02-action.html`. Скопируйте HTML-код, вернитесь к файлу `index.html` и вставьте его во второй элемент `div` с классом `row`, находящийся внутри

элемента `div` с классом `container`. Результат операции должен выглядеть следующим образом (добавленный текст выделен полужирным шрифтом):

```
<!-- основной раздел -->
<div class="container">
  <div class="row">
    <div class="four columns">
      <p class="logo">Моя компания</p>
    </div>
    <div class="eight columns nav">
      <a class="button button-primary" href="#">Главная</a>
      <a class="button" href="#">Клиенты</a>
      <a class="button" href="#">0 нас</a>
      <a class="button" href="#">Вакансии</a>
    </div>
  </div>
  <div class="row">
    <h1>Мы творим великие дела</h1>
    <h2>Donec pulvinar ullamcorper metus</h2>
    <a href="#" class="button button-primary">Подробнее</a>
  </div>
</div>
```

Содержимое этой строки будет распространяться на всю ширину контейнера, другими словами, будет создана только одна колонка. В этом случае вы не добавляли дополнительные элементы `div`; вы просто вставили HTML-код внутри элемента `div` с классом `row`.

10. Добавьте класс `action` ко второму элементу `div` с классом `row`:

```
<div class="row action">
  <h1>Мы творим великие дела</h1>
  <h2>Donec pulvinar ullamcorper metus</h2>
  <a href="#" class="button button-primary">Подробнее</a>
</div>
```

Вы будете использовать это имя класса позже, при форматировании этой области страницы. Имя `action` не имеет ничего общего с таблицей стилей Skeleton — это лишь имя класса, которое вы можете использовать для форматирования данной области страницы.

11. Сохраните файл `index.html` и просмотрите его в браузере.

Страница должна выглядеть так, как показано на рис. 16.6. На данный момент не похоже, что для нее используется разрекламированная модульная сетка. Но мы это исправим. Добавим еще одну строку и три колонки. Чтобы выполнить эту операцию быстрее, базовый HTML-код контента для этой страницы сохранен в отдельном файле.

12. Откройте файл `03-info.html`. Скопируйте HTML-код, вернитесь к файлу `index.html` и вставьте его после комментария `<!-- раздел контента -->`.

Это базовый набор вложенных элементов `div`. Один из элементов `div` представляет собой контейнер для строк, а два элемента `div` внутри него — строки. В первой



Рис. 16.6. Таблица стилей Skeleton предоставляет не только модульную сетку, но и некоторые базовые текстовые стили и красивые кнопки

строке содержится только элемент `h2`, который будет занимать одну колонку во всю ширину контейнера. Во второй строке находятся три дополнительных элемента `div`, которые представляют собой три колонки.

HTML-код еще не содержит имен классов Skeleton — это секретный ингредиент, который применяет модульную сетку. Сейчас вы его добавите.

13. Добавьте код `class="container"` к первому элементу `div` в HTML-код, который вы только что вставили на страницу:

```
<!-- раздел контента -->
<div class="container">
```

Этот код применяет стиль контейнера Skeleton к элементу, создавая пространство для добавления строк. Далее вы добавите несколько строк.

14. Добавьте код `class="row"` к следующим двум элементам `div` внутри контейнера, как это показано в следующем примере (добавленный код выделен полужирным шрифтом):

```
<div class="container">
  <div class="row">
    <h2>Graeco feugiat intellegam at vix</h2>
  </div>
  <div class="row">
```

Теперь, когда вы создали строки, во второй строке необходимо создать три колонки. Вы можете сделать это, добавив атрибут `class` с двумя именами классов: первое имя — количество столбцов, которое занимает колонка по ширине, а второе — `columns`.

15. Добавьте код `class="four columns"` к каждому элементу `div` внутри второго `div` с классом `row` в раздел контента страницы. Кроме этого, добавьте класс `button` к трем ссылкам в каждой из колонок. Законченный HTML-код для раздела контента страницы должен выглядеть следующим образом:

```
<!-- раздел контента -->
<div class="container">
  <div class="row">
    <h2>Graeco feugiat intellegam at vix</h2>
```



```

</div>
<div class="row">
  <div class="four columns">
    <p>Mei te possit instructor...</p>
    <p><a href="#" class="button">Дополнительно</a></p>
  </div>
  <div class="four columns">
    <p>Mei te possit instructor...</p>
    <p><a href="#" class="button">Дополнительно</a></p>
  </div>
  <div class="four columns">
    <p>Mei te possit instructor...</p>
    <p><a href="#" class="button">Дополнительно</a></p>
  </div>
</div>
</div>

```

(Приведенный HTML-код сокращен и не включает «бред» на латинице в каждой из колонок.) Наконец, вы добавите нижний колонтитул на страницу.

- Откройте файл `04-footer.html`. Скопируйте HTML-код, вернитесь к файлу `index.html` и вставьте его после комментария `<!-- колонтитул -->`.

Этот HTML-код содержит соответствующие классы Skeleton (наверняка вы уже знаете, как работает таблица стилей Skeleton, и практиковаться в ее использовании больше нет смысла). HTML-код, который вы только что добавили, вставляет контейнер с одной строкой и двумя колонками. Ширина одной из них равна восьми столбцам, а второй — четырем столбцам.

- Сохраните файл `index.html` и просмотрите его в браузере.

Страница должна выглядеть так, как показано на рис. 16.7. Макет сверстан: в разделе заголовка есть две колонки (для логотипа и панели навигации), три колонки основного контента и две колонки для нижнего колонтитула.

Если вы измените размер окна браузера, сделав его слишком узким, колонки разместятся друг за другом. Так проявляется гибкость системы модульной верстки Skeleton в действии. В ней используются медиазапросы, о которых вы читали в предыдущей главе, позволяющие превратить контент страницы в одноколоночный дизайн для мобильных устройств.

Создание стилей

С помощью Skeleton легко сверстать макет страницы, но для придания ей уникальности нужно добавить дополнительные каскадные таблицы стилей. В этом разделе практикума вы добавите несколько штрихов к дизайну страницы.

- В редакторе HTML-кода откройте файл `custom.css`, расположенный в папке `16\css`.

По большому счету, этот CSS-файл не содержит стилей. В нем находятся только медиазапросы, предназначенные для устройств с экранами с разным разрешением по горизонтали, для которых вы, возможно, захотите изменить размер



Рис. 16.7. С помощью правильно структурированного HTML-кода, нескольких имен классов и CSS-файла системы модульной верстки Skeleton легко создавать многоколоночные дизайны страниц

и оформление элементов страницы. Например, скорее всего, вы захотите, чтобы при просмотре страницы с помощью смартфона размер заголовка уменьшился, а лишние поля и отступы были удалены.

Для начала посмотрим, что произойдет при добавлении фонового цвета в контейнер.

2. Добавьте следующий стиль сразу после комментария `/*` стили для любой ширины `*/`:

```
/* запросы mobile first */
/* стили для любой ширины */
.container {
  background-color: pink;
}
```

Как вы, возможно, помните, наша страница содержит несколько элементов `div` с классом `container`. Это элементы, которые содержат строки.

3. Сохраните файл `custom.css` и просмотрите файл `index.html` в браузере.

Страница должна выглядеть так, как показано на рис. 16.8. Обратите внимание, что фон розового цвета ограничен по ширине контента и выровнен по центру

окна браузера. Если вы взглянете на окончательный дизайн на рис. 16.4, то увидите, что в области заголовка в качестве фона используется фотография, которая должна отображаться по всей ширине окна браузера. Контейнеры Skeleton не делают этого, поэтому вам нужно добавить еще один элемент `div`, который будет распространяться на всю ширину окна браузера.

- Откройте файл `index.html` в редакторе HTML-кода. Найдите комментарий `<!-- основной раздел -->` и добавьте элемент `div` сразу после него.

```
<!-- основной раздел -->
<div class="section header">
  <div class="container">
```

Вы только что «обернули» контейнер элементом `div`. Кроме того, вы присвоили ему два имени класса — `section` и `header`, что позволяет форматировать эти области отдельно от других областей страницы. Сейчас необходимо закрыть элемент `div`.

- Добавьте закрывающий тег `</div>` перед комментарием `<!-- раздел контента -->`:

```
</div>
<!-- раздел контента -->
```

Теперь новый элемент `div` обернут вокруг контейнера. По умолчанию, как и все блочные элементы, `div` имеет ширину 100%. То есть он растягивается, заполняя свой родительский элемент, который в данном случае является телом страницы. Поэтому новый элемент `div` заполнит окно браузера по всей ширине, но выйдет за пределы области заголовка.

- Сохраните файл `index.html` и вернитесь к файлу `custom.css` в редакторе HTML-кода. Удалите правило `.container`, добавленное в шаге 2, и замените его следующим кодом:

```
/* запросы mobile first */
/* стили для любой ширины */
.header {
  padding: 50px 0 70px 0;
  background-image: url(../imgs/header.jpg);
  background-size: cover;
}
```

Свойство `padding` добавляет небольшой отступ в верхней и нижней областях страницы. Свойство `background-size` масштабирует изображение, добавленное с помощью свойства `background-image`, всегда заполняя фон. Если вы сейчас сохраните файлы и просмотрите файл `index.html`, то заметите, что изображение распространяется по всей ширине окна браузера, но располагается только в области заголовка.

Теперь вы можете отформатировать область заголовка.

- Добавьте стиль для логотипа сразу после стиля `.header`:

```
.logo {
  font-weight: 600;
```

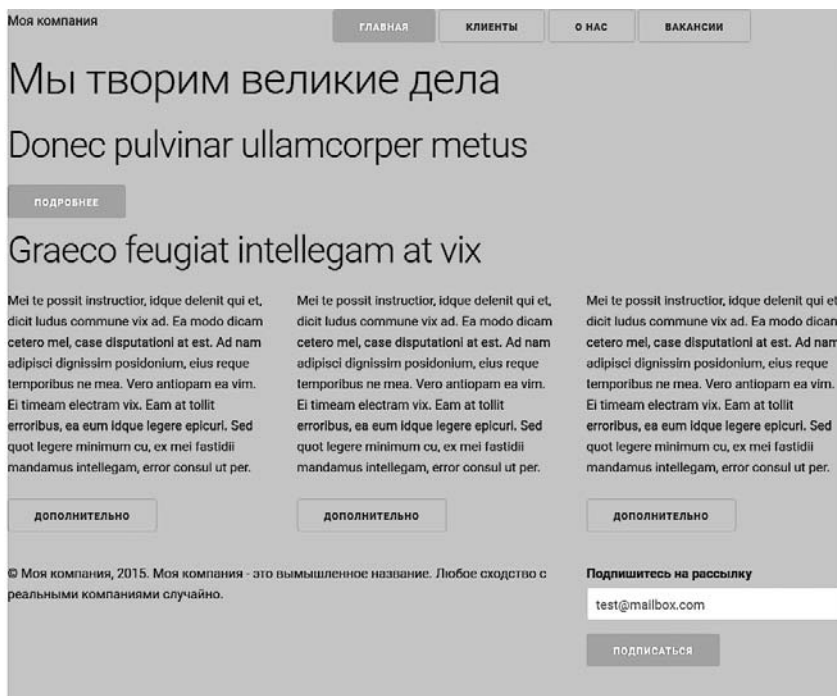


Рис. 16.8. Класс `container` таблицы стилей `Skeleton` не заполняет все окно браузера по ширине. Это проблема, если вы используете фоновый цвет или изображение

```
color: rgb(255, 209, 143);
}
```

Эта страница использует веб-шрифт Google под названием `Roboto`, который включает в себя три различных начертания. Как упоминалось, вы можете использовать числа, чтобы указать различные стили шрифта, начиная с очень тонких и заканчивая очень толстыми (`700` — это полужирная версия шрифта). Далее вы выровняете элементы навигации по правому краю страницы и улучшите внешний вид кнопок, расположенных поверх фонового изображения.

- Добавьте следующий код после стиля `.logo`:

```
.nav {
  text-align: right;
}
.button {
  background-color: rgba(255,255,255,.5);
}
.button:hover {
  background-color: rgba(255,255,255,.3);
}
```

Стиль `.nav` форматирует колонку с кнопками навигации. В шаге 5 в предыдущем подразделе вы добавили имя `nav` в класс элемента `div`. Два других стиля

изменяют внешний вид кнопки так, чтобы они лучше выделялись на фоне изображения.

Наконец, вы выровняете по центру текст и кнопку под областью навигации.

9. Добавьте следующий код под только что добавленными стилями кнопок:

```
.action {
  text-align: center;
  padding-top: 37px;
}
.action h1 {
  margin: 0;
}
.action h2 {
  margin: 0 0 20px 0;
}
```

В шаге 10 ранее вы добавили класс `action`. Сейчас вы выровняли контент в этой области и удалили воздух вокруг заголовков.

10. Сохраните файл и просмотрите `index.html` в браузере.

Страница должна выглядеть так, как показано на рис. 16.9.

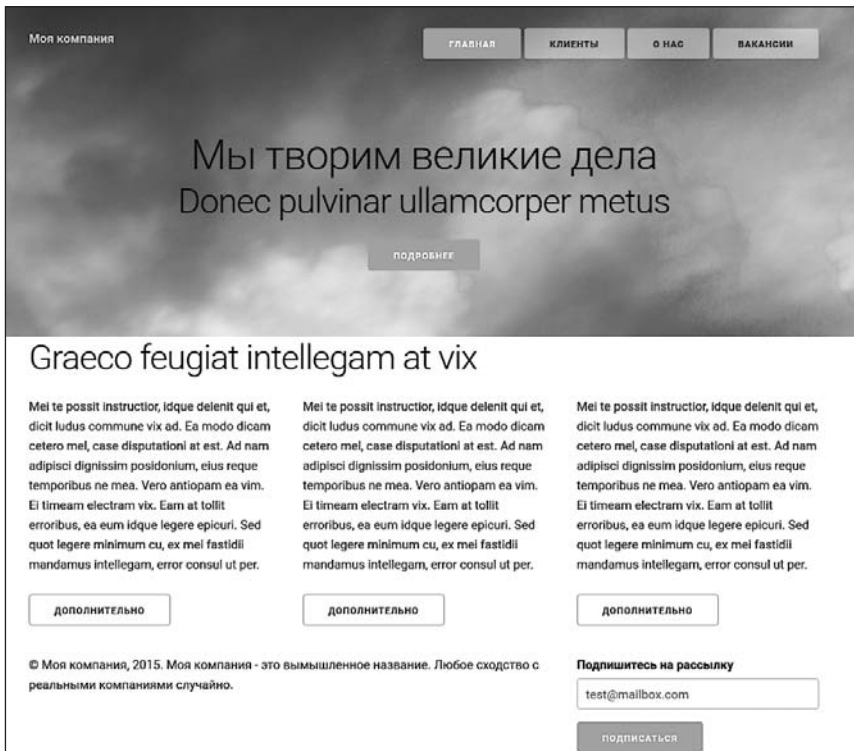


Рис. 16.9. Добавленное изображение и несколько стилей образуют оболочку для каркаса Skeleton

Адаптация под мобильные устройства

Таблица стилей Skeleton в первую очередь предназначена для мобильных устройств. Это значит, что вы начинаете с дизайна для мобильных устройств с небольшим экраном, таких как смартфоны. Затем вы добавите стили внутри нескольких медиазапросов, которые предназначены для устройств с экранами с более высоким разрешением по горизонтали. Каждый набор стилей предыдущего запроса применяется к текущему запросу и всем остальным запросам с более высокими значениями свойства `min-width`.

Задумайтесь об усовершенствованиях, которые можно сделать на страницах для посетителей, использующих смартфоны. Рисунок 16.10 демонстрирует, как текущая страница будет выглядеть на смартфоне iPhone 5: на ней очень много пространства в верхней части экрана. Вы можете исправить стиль логотипа, чтобы разместить название компании в верхней части экрана, и исправить кнопки навигации, чтобы они лучше соответствовали текущему размеру экрана.

1. В редакторе HTML-кода откройте файл `custom.css`, расположенный в папке `16\css`.

Стили, добавленные в файл ранее в этом уроке, находятся в верхней части файла, перед медиазапросами. В соответствии со стратегией Mobile First эти стили будет применяться, независимо от ширины окна браузера, как на малых экранах смартфонов, так и на очень широких компьютерных мониторах.

Во-первых, вы измените стиль `.logo` так, чтобы дизайн лучше смотрелся на небольшом экране смартфона.

2. Найдите стиль `.logo` и добавьте к нему семь новых строк кода (добавленный код выделен полужирным шрифтом):

```
.logo {  
  font-weight: 600;  
  color: rgb(255, 209, 143);  
  background-color: black;  
  position: fixed;  
  top: 0;  
  left: 0;  
  right: 0;  
  padding-left: 10px;  
  z-index: 100;  
}
```

Свойство `background-color` визуально выделяет название компании. Значения `position`, `top`, `left` и `right` фиксируют этот элемент в верхней части окна браузера. Свойство `padding-left` добавляет небольшой отступ в левой части экрана. И наконец, свойство `z-index` гарантирует, что при прокрутке страницы вниз логотип останется поверх прочего контента страницы (рис. 16.11, *вверху слева*).

Кнопки навигации будут выглядеть лучше, если они будут одинаковой ширины, а свободного пространства между ними останется меньше.

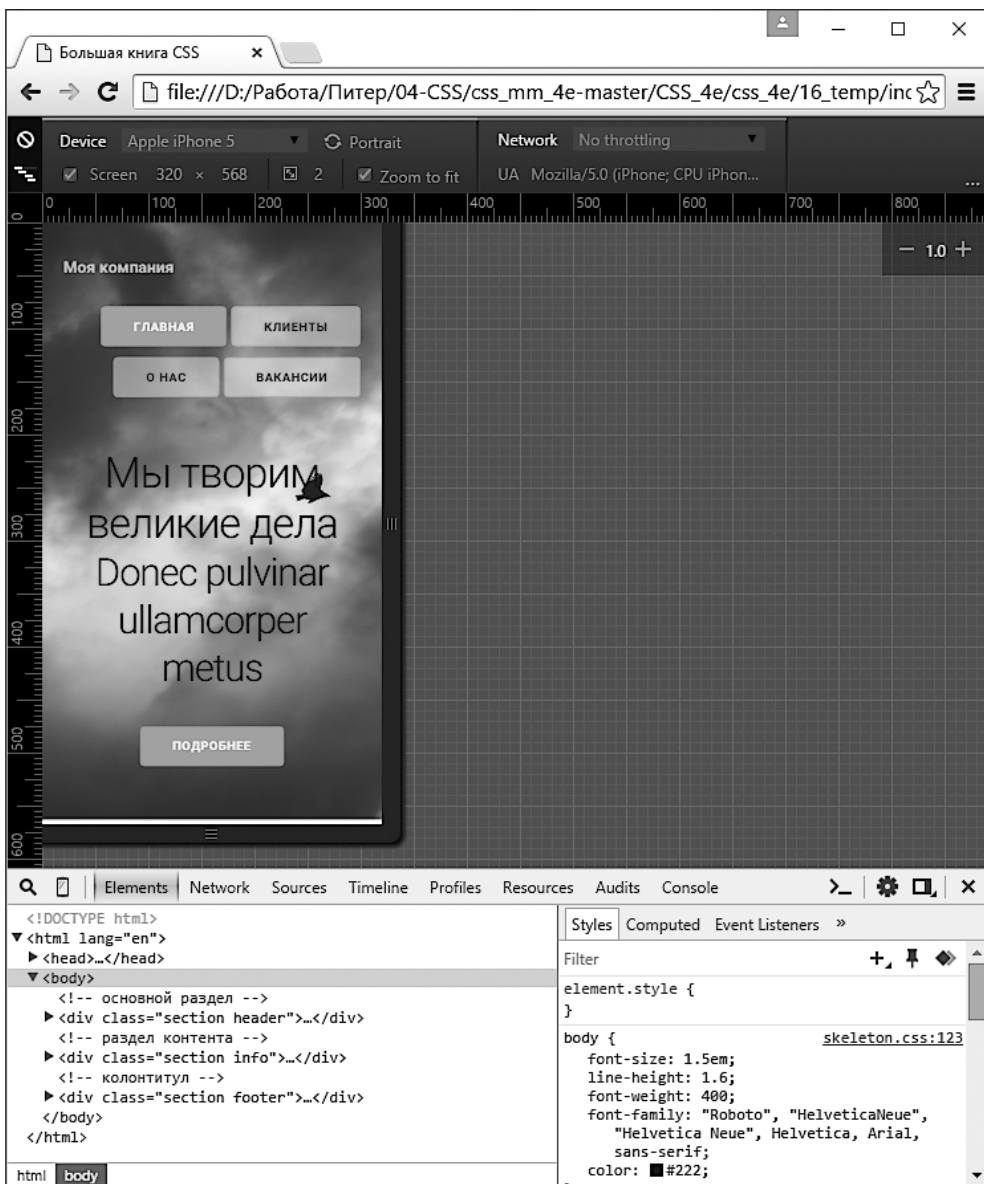


Рис. 16.10. С помощью панели Developer Tools (Инструменты разработчика) браузера Chrome вы можете эмулировать экран различных устройств, включая iPhone 5 (в данном случае)

3. Добавьте следующий стиль под правилом `.nav` в файле `custom.css`:

```

.nav .button {
  width: 48%;
  margin: 2px 0;
}

```

Этот стиль выравнивает ширину всех кнопок на панели навигации, а также настраивает поля, чтобы кнопки не касались друг друга (см. рис. 16.11, *вверху справа*).

В верхней части страницы все еще слишком много пустого пространства.

4. Найдите первый стиль в файле `custom.css` — правило `.header` — и измените значение свойства `padding` на `30px 0 20px 0;`, чтобы удалить пустое пространство в верхней части страницы.

Теперь панель навигации и логотип лучше подходят для экранов с небольшим разрешением. Тем не менее под кнопкой **Подробнее** слишком много воздуха (см. рис. 16.11, *внизу слева*). Наконец, было бы неплохо немного уменьшить слишком большой заголовок и добавить пространство под кнопками навигации.

5. Найдите правило `.action` и измените значение свойства `padding-top` на `37px`.

Этот код добавляет некоторое пространство между первым заголовком и кнопками навигации. Теперь необходимо уменьшить размер шрифта заголовков.

6. В файле `custom.css` сразу после правила `.action h2` добавьте два следующих стиля:

```
h1 {
  font-size: 3rem;
}
h2 {
  font-size: 2.5rem;
}
```

Если вы сохраните файл и просмотрите его с помощью панели **Developer Tools** (Инструменты разработчика) браузера Chrome и в режиме эмуляции смартфона на iPhone 5, то увидите нечто похожее на показанное на рис. 16.11.

Форматирование точек останова

До этого момента вы занимались базовым дизайном для мобильных устройств. Созданная вами страница выглядит прекрасно на экранах, разрешение по горизонтали которых не превышает 550 пикселей. Другими словами, вам не нужно добавлять какие-либо стили к первому медиазапросу, значение свойства `min-width` которого равно 400 пикселям. Тем не менее при экранном разрешении по горизонтали 550 пикселей и выше дизайн страницы начинает выглядеть немного странно. Это точка, в которой встроенная в систему **Skeleton** модульная сетка изменяется, а дизайн страницы становится многоколоночным.

Как вы можете увидеть на рис. 16.12, кнопки навигации занимают много места и не выровнены. Самое время улучшить внешний вид вашей страницы для экранов с более высоким разрешением по горизонтали.

1. Откройте файл `custom.css` в редакторе HTML-кода. Найдите медиазапрос, значение свойства `min-width` которого равно `550px`, и добавьте следующий код (добавленный код выделен полужирным шрифтом):

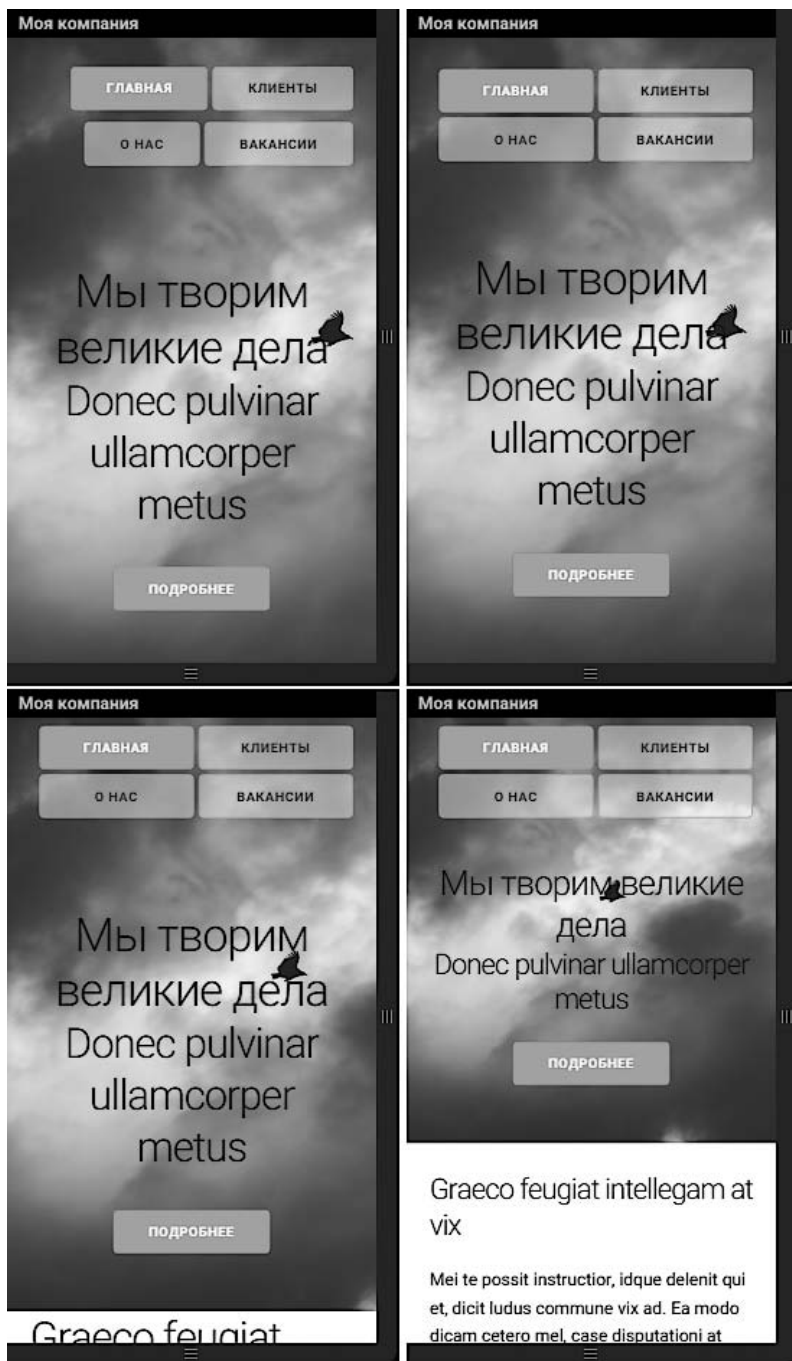


Рис. 16.11. Создание дизайна, предназначенного в первую очередь для мобильных устройств, требует постоянной итерации — изменения стилей с целью настройки воздуха, размера шрифта и позиционирования элементов для устройств с различным экранным разрешением

```

/* стили для ширины 550 пикселей и выше */
@media (min-width: 550px) {
  .header {
    padding: 40px 0 50px 0;
  }
  .logo {
    position: static;
    background-color: transparent;
    font-size: 2rem;
    line-height: 1;
  }
  .nav .button {
    width: auto;
  }
  h1 {
    font-size: 5rem;
  }
  h2 {
    font-size: 4.2rem;
  }
}

```

Эти стили сбрасывают пространство в области заголовка страницы, изменяют логотип (он больше не фиксируется на экране), настраивают ширину кнопок и увеличивают размер шрифта заголовков (рис. 16.13, *вверху справа*).

Наконец, вам необходимо увеличить размер шрифта текста логотипа.

2. В медиазапрос, значение свойства `min-width` которого равно 750px, добавьте следующий стиль:

```

/* стили для ширины 750 пикселей и выше */
@media (min-width: 750px) {
  .logo {
    font-size: 3rem;
    position: relative;
    top: 5px;
  }
}

```

Этот код изменяет размер шрифта, который используется для названия компании, и немного смещает его.

3. Сохраните файл `custom.css`. Откройте файл `index.html` в браузере и уменьшите ширину окна таким образом, чтобы она не превышала 550 пикселей. Затем медленно увеличьте ее.

Дизайн страницы, когда окно браузера уже 550 пикселей, показан вверху слева на рис. 16.13. По мере увеличения ширины окна браузера страница будет изменяться два раза: один раз при ширине 550 пикселей (*вверху справа*) и еще раз при ширине 750 пикселей (*внизу*).



Рис. 16.12. Модульная сетка системы Skeleton меняется при отображении на устройствах с экраным разрешением по горизонтали выше 550 пикселей. Медиазапрос начинает действовать, а стили, расположенные в нем, преобразуют страницу, добавляя колонки, заливая в них ваш контент

Вы можете и дальше улучшать дизайн страницы, используя различные точки останова. Попробуйте добавлять разные стили для различных точек останова медиазапросов и посмотреть, как еще можно улучшить этот проект.

Полную версию файлов этого практикума можно найти в папке `16_finished`.

Как вы можете видеть, система модульной верстки представляет собой очень удобный инструмент, который позволяет легко создавать колонки с согласованной шириной. Тем не менее, если присмотреться, вы обнаружите те же принципы, о которых узнали ранее в этом разделе книги: обтекаемые элементы, значения ширины в процентах и т. д. Если вы потратите некоторое время и изучите файл `skeleton.css`, то узнаете о принципах его работы.

СОВЕТ

Статью о том, как создать собственную систему модульной верстки, вы найдете по адресу tinyurl.com/otqef3v.



Рис. 16.13. Система модульной верстки Skeleton позволяет легко создавать веб-страницы, адаптируемые под браузеры различных устройств, начиная со смартфонов и заканчивая телевизорами

17 Профессиональная flexbox-верстка

За короткий период существования Всемирной паутины дизайнеры использовали различные способы верстки макетов веб-страниц. Сначала они применяли таблицы HTML для организации контента в строках и столбцах. Хотя HTML-элемент `table` никогда не предназначался для компоновки макетов страниц, дизайнеры обнаружили креативные (и сложные) способы использования элемента `table` для достижения своих целей.

Позже каскадные таблицы стилей и макеты на основе обтекаемых элементов (см. главу 13) обеспечили более простой и логичный способ управления дизайном страницы. Макеты на основе обтекаемых элементов по-прежнему наиболее распространены, и дизайнеры продолжают совершенствовать их. Например, простая система модульной верстки, с которой вы познакомились в предыдущей главе, представляет собой сложный инструмент для создания макетов, но и она использует обтекаемые элементы для своих «волшебных» преобразований.

Тем не менее Всемирная паутина развивается, и рабочая группа CSS Консорциума W3C работает над тем, чтобы предоставить дизайнерам гибкие и мощные свойства каскадных таблиц стилей для верстки макетов страниц. Одним из новых методов, который уже довольно широко поддерживается браузерами, является спецификация *Flexbox*.

Знакомство с методом flexbox-верстки

Flexbox — это новый метод верстки страниц с помощью каскадных таблиц стилей. Вы уже читали о строчных и блочных элементах (см. главу 7). Таблицы и позиционируемые элементы (см. главу 14) также являются методами верстки с помощью CSS. Flexbox (сокращение от *flexible box* («гибкий блок»)) реализует способ верстки так называемых *flex-макетов*.

Flexbox обеспечивает весьма полезный набор свойств, которые позволяют заливать элементы в строке макета без необходимости их выравнивания или использования свойства `inline-block`. В самом деле «гибкость» flexbox-верстки позволяет автоматически настраивать ширину элементов, находящихся внутри flex-контейнера, что очень похоже на выравнивание на странице обтекаемых элементов с применением процентных значений.

Но использовать метод flexbox-верстки намного проще, чем обтекаемые элементы. Кроме того, он обеспечивает все те же преимущества верстки, как это показано на рис. 17.1. Самым большим недостатком является то, что не все браузеры поддерживают flex-дизайны. К счастью, новые браузеры — Internet Explorer, начиная с версии 10 и выше¹, Edge, Chrome, Safari, Opera, Firefox — поддерживают flex-дизайны.

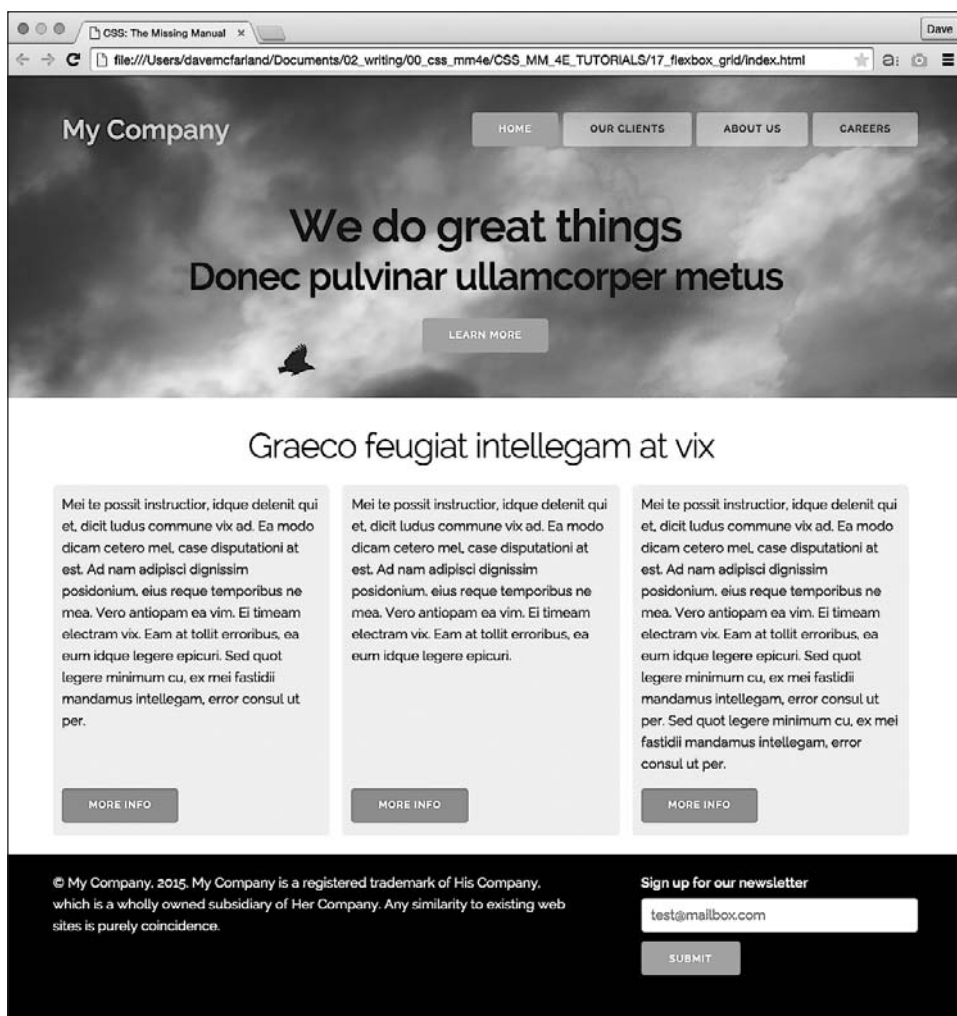


Рис. 17.1. С помощью flexbox-верстки вы можете создавать такие же дизайны, как и путем выравнивания элементов, но при этом используя намного меньше HTML- и CSS-кода. На самом деле существуют вещи, которые очень легко сделать с помощью flexbox-верстки и чрезвычайно трудно осуществить каким-либо другим способом. Видите, что три колонки в центре страницы имеют одинаковую высоту? И как кнопки More Info выровнены в нижней части каждой колонки? С помощью flexbox-верстки сделать это очень просто

¹ Частичная поддержка. — *Примеч. пер.*

Основы flexbox-верстки

Суть flexbox-верстки довольно проста. Для работы с ней необходимы лишь два компонента.

1. **Flex-контейнер.** Любой HTML-элемент может быть flex-контейнером, но обычно им становится `div` или какой-либо другой структурный HTML-элемент. Элемент, который вы станете использовать в качестве контейнера, будет содержать дочерний и другие элементы, которые составляют второй компонент модели flexbox.
2. **Flex-элементы.** Элементы, вложенные непосредственно в flex-контейнер, называют *flex-элементами*. Каждый прямой потомок элемента контейнера автоматически становится flex-элементом. Вы можете поместить любой HTML-элемент внутрь flex-контейнера. Более того, дочерние элементы даже не должны быть того же типа. Например, страница может содержать абзац и четыре элемента `div` внутри flex-контейнера, и каждый из них будет являться flex-элементом.

Имейте в виду, что только дочерние элементы flex-контейнера превращаются в flex-элементы. Если у вас есть элемент `div`, который вы поместили в flex-контейнер и добавили неупорядоченный список в него, то только `ul` будет являться flex-элементом. Элементы `li`, вложенные в `ul`, не станут flex-элементами.

Другими словами, как вы, наверное, заметили, использовать метод flexbox-верстки так же просто, как добавить элемент `div` на страницу и вложить в него другие элементы `div`. В качестве примера ниже приведен простой HTML-код, который может быть легко использован для создания строки элементов с помощью метода flexbox:

```
<div class="container">
  <div>A flex item</div>
  <div>Another flex item</div>
  <div>A third flex item</div>
</div>
```

Внешний элемент `div` является контейнером, а элементы `div` внутри него — дочерние. Браузер отобразит эту серию `div` в виде блочных элементов, заполняя ими всю ширину внешнего элемента `div` и размещая их друг за другом, как это показано на рис. 17.2, *вверху*.

С помощью свойства `display` и присвоения ему значения `flex` вы можете легко преобразовать внешний элемент `div` в flex-контейнер:

```
.container {
  display: flex;
}
```

Эта одна строка CSS-кода позволяет получить эффект, показанный на втором сверху изображении на рис. 17.2. Все дочерние элементы `div` автоматически преобразуются в flex-элементы, который помещаются в строке рядом друг с другом: никаких обтекаемых или строчных элементов.

Наконец, вы можете согласовать ширину всех элементов `div` внутри контейнера и заполнить контейнер, добавив свойство `flex` и присвоив ему значение `1`:


```
.container div {  
  flex: 1;  
}
```

Подробнее о свойстве `flex` вы узнаете позднее, но, если говорить вкратце, эта строка кода позволяет браузеру отобразить каждый элемент `div` (`flex`-элемент) с одинаковой шириной. На втором снизу изображении на рис. 17.2 показан пример использования данного свойства.

СОВЕТ

`Autoprefixer` (tinyurl.com/pqomk7e) — это инструмент, который автоматически добавляет вендорные префиксы в CSS-код. Напишите, как обычно, CSS-код, запустите его в приложении `Autoprefixer`, и таблица стилей будет дополнена всеми необходимыми вендорными префиксами.

Поскольку `flex`-элементы автоматически соприкасаются друг с другом, возможно, вы захотите добавить пространство, чтобы разделить их. Существует много способов сделать это, но для данного примера доступно одно простое решение:

```
.container div:nth-of-type(1n+2) {  
  margin-left: 20px;  
}
```

Селектор `nth-of-type` выделяет каждый элемент `div`, начиная со второго, и добавляет к нему отступ, равный 20 пикселям (этот хитрый прием гарантирует, что к первому элементу `div` не будет применен отступ 20 пикселей от края контейнера). Результат действия кода показан на нижнем изображении на рис. 17.2.

В теории метод `flexbox`-верстки довольно прост. Как вы можете видеть, для достижения эффекта не требуется длинного CSS-кода. Самое главное заключается в том, что вам не нужно беспокоиться об объектах, выпадающих из своих контейнеров, как это было с обтекаемыми элементами (см. раздел «Решение проблем с обтекаемыми элементами» главы 13), и вы можете легко создать одинаковые по высоте колонки. Единственное, что затрудняет использование метода `flexbox`-верстки, — необходимо знать и понимать большое количество свойств метода `flexbox` и представлять почти безграничные их комбинации.

Свойства `flex`-контейнера

`Flex`-контейнеры и `Flex`-элементы имеют собственные наборы свойств CSS, которые определяют, как браузер отображает их. Существует несколько свойств, характерных только для `flex`-контейнеров, наиболее важным из которых является свойство `display`. Чтобы преобразовать любой HTML-элемент во `flex`-контейнер, используйте следующий код:

```
display: flex;
```

Вы встречались со свойством `display` раньше. Вы также можете использовать его, чтобы превратить элементы в блочные или строчные блочные элементы и даже скрыть их: `display: none`. То же свойство можно применять, чтобы преобразовать элемент в `flex`-элемент.

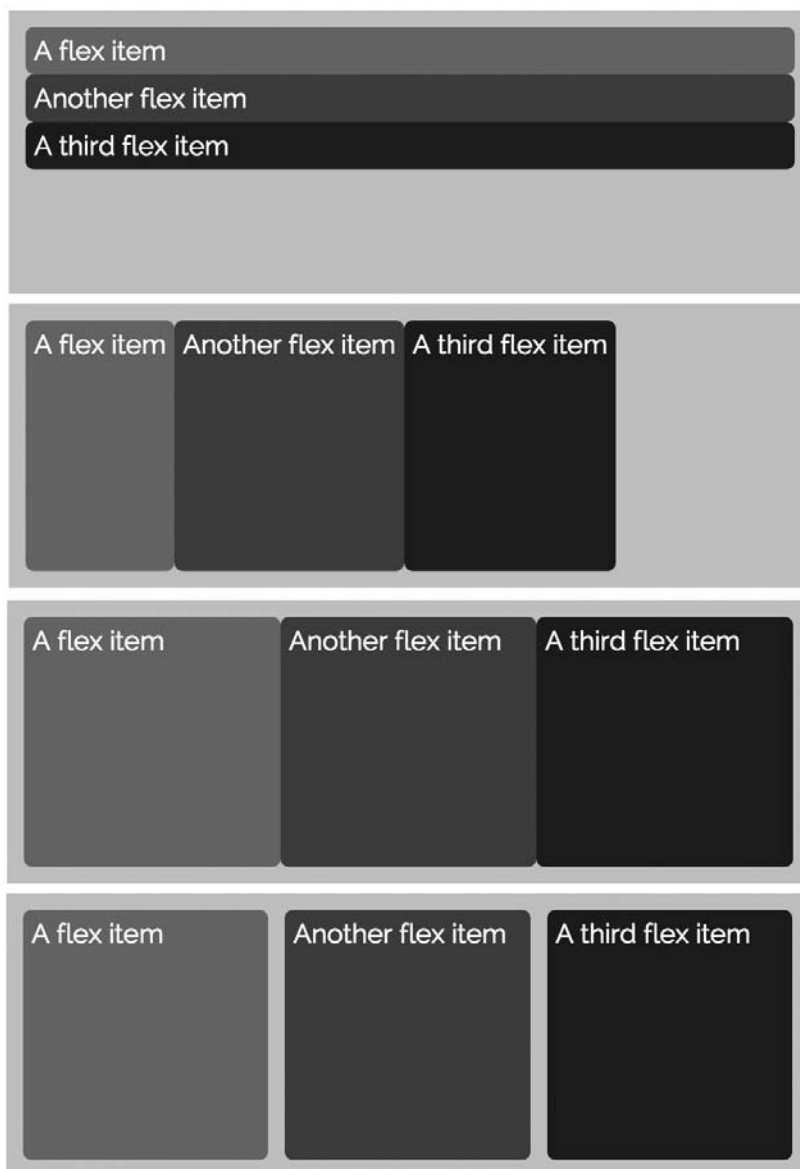


Рис. 17.2. Метод flexbox-верстки позволяет легко создавать колонки одинаковой ширины и высоты, расположенные рядом друг с другом без использования обтекаемых элементов. Дополнительное пространство, которое вы видите вокруг элементов (ярко-серый фон), — отступ, добавленный в flex-контейнер

Помните, что вы применяете это свойство к контейнеру — элементу, который оборачивается вокруг других элементов, являющихся flex-объектами. Вы даже можете превратить flex-элемент в flex-контейнер для достижения некоторых интересных эффектов. Пример этого метода приведен в практикуме к текущей главе.

Свойство flex-flow

По умолчанию flex-элементы помещаются друг за другом, как ячейки в строке таблицы. Кроме того, браузер отображает их в одной строке, без обертки. Другими словами, независимо от того, каким узким будет окно браузера, он будет отображать элементы рядом друг с другом в строке, не перемещая, даже если при этом контент выйдет за пределы flex-элемента (рис. 17.3, *слева*).

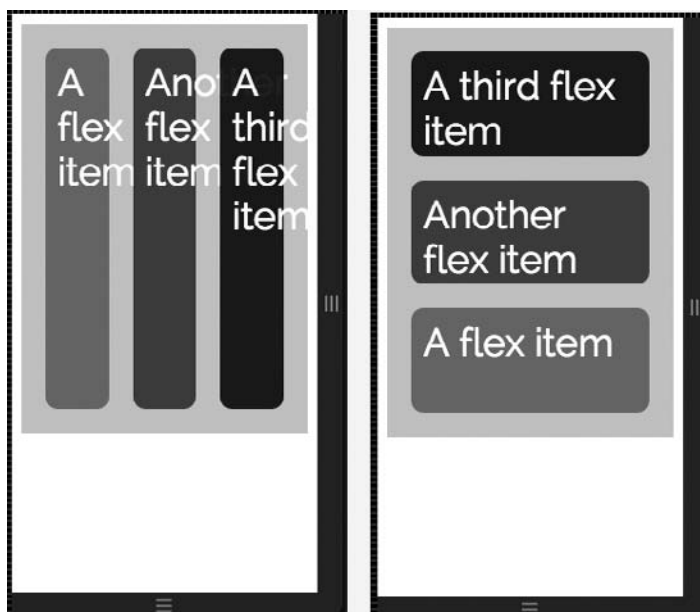


Рис. 17.3. Обычно flex-элементы не оборачиваются, и если окно браузера станет достаточно узким, контент внутри flex-элемента выйдет за пределы (*слева*). Но вы также можете отобразить flex-элементы друг над другом и даже изменить порядок, в котором они отображаются на странице (*справа*)

Свойство flex-flow позволяет выбрать направление, в котором элементы отображаются, а также указать, будут ли они переноситься на следующую строку. Для свойства flex-flow необходимо задать два значения, разделяя их пробелом. Первое определяет направление, а второе указывает, допустим ли перенос на следующую строку. Ниже приведен пример:

```
.container {
  display: flex;
  flex-flow: column-reverse nowrap;
}
```

Первое значение свойства flex-flow определяет направление и может быть одним из следующих.

- row — значение по умолчанию. Оно отображает flex-элементы последовательно, рядом друг с другом. Самый первый элемент в исходном HTML-коде отображается в крайнем левом положении, а последний — справа (см. рис 17.3, *слева*).

- `row-reverse` также отображает flex-элементы рядом друг с другом, но обращает порядок их следования. Другими словами, последний элемент в исходном HTML-коде отображается в крайнем левом положении контейнера, а первый — у правого края контейнера.
- `column` отображает flex-элементы в виде блоков, друг над другом. Это обычный режим отображения элементов `div`, поэтому, скорее всего, вы редко будете использовать это значение. Тем не менее оно пригодится в медиазапросах и при адаптации дизайна под мобильные устройства. Вы можете отображать flex-элементы рядом друг с другом в строке (для экранов с высоким разрешением) или друг над другом для мобильных устройств, изменяя направление потока в медиазапросе (вы научитесь это делать в практикуме в конце текущей главы).
- `column-reverse` — это значение аналогично `column`, с той лишь разницей, что изменяет направление отображения элементов на противоположное. Объекты, которые отображаются последними в исходном HTML-коде, находятся в верхней части контейнера (см. рис. 17.3, *справа*).

Второе свойство определяет, будут ли flex-элементы переноситься на новую строку (при значении `row`) или в новую колонку (при значении `column`). Свойство может принимать три возможных значения.

- `nowrap` — обычное поведение flex-элемента в flex-контейнере. Браузер будет отображать элементы в одной строке, независимо от того, насколько узким станет окно браузера (см. рис. 17.3, *слева*). При установке значения `column` браузер будет размещать элементы друг над другом (см. рис. 17.3, *справа*).
- `wrap` — позволяет объектам, которые не помещаются в контейнер по ширине, переноситься на новую строку (или в колонку), как показано на верхнем изображении на рис. 17.4. Чтобы flex-элементы занимали новые строки (или колонки), необходимо использовать некоторые свойства, описанные в разделе «Свойства flex-элементов» далее.
- `wrap-reverse` — работает аналогично значению `wrap`, но элементы переносятся в обратном порядке (см. рис. 17.4, *внизу*).

Flex-элементы могут размещаться в строке, рядом друг с другом, или в колонке, друг за другом, в зависимости от выбранного направления (`row` или `column`). Тем не менее, поскольку значение `row` наиболее распространено при верстке страниц, для всех последующих примеров flexbox-верстки, приведенных в этой главе, будет использоваться строчное размещение flex-элементов.

ПРИМЕЧАНИЕ

Свойство `flex-flow` состоит из двух свойств каскадных таблиц стилей: `flex-direction` и `flex-wrap`. Например, код:

```
flex-flow: row wrap;
```

аналогичен коду:

```
flex-direction: row;
```

```
flex-wrap: wrap;
```

Поскольку свойство `flex-flow` более лаконично, в книге мы будем использовать именно его.

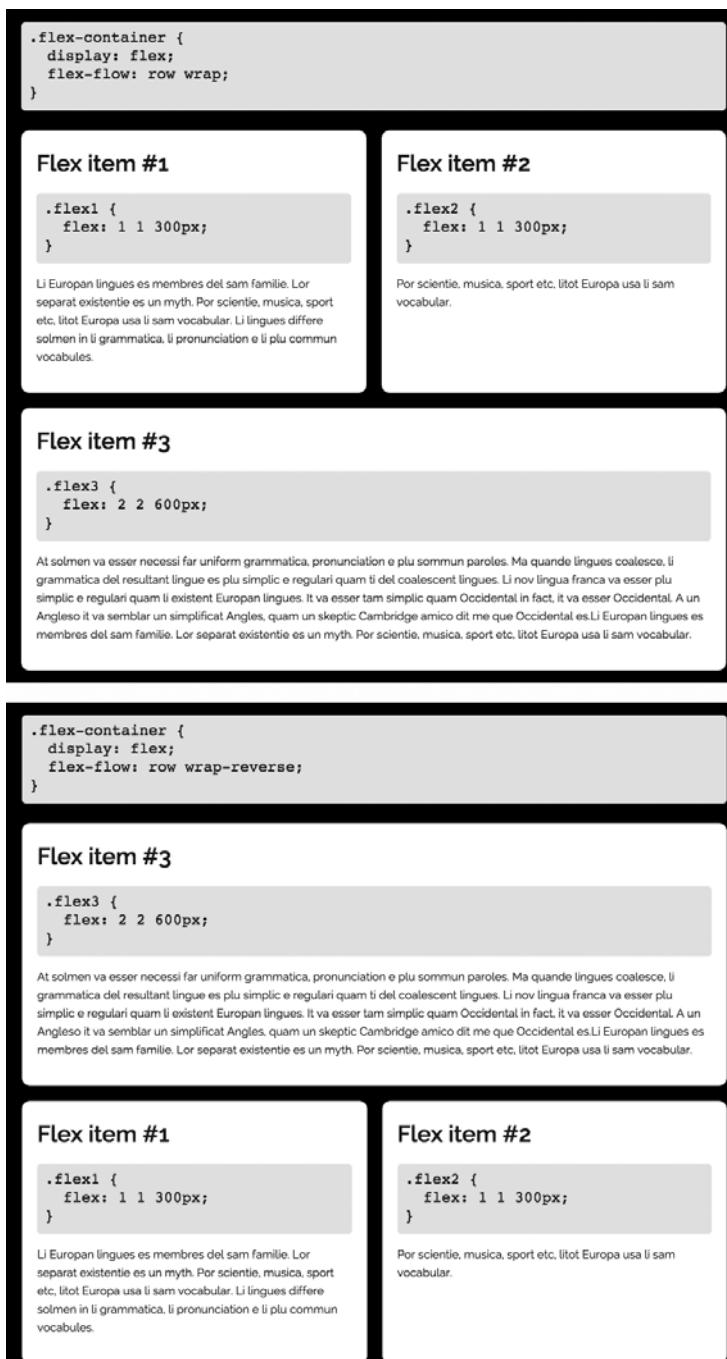


Рис. 17.4. Flex-элементы, которым не хватает ширины flex-контейнера, могут переноситься на новую строку. Значение wrap переносит последний элемент на новую строку (*вверху*), в то время как значение wrap-reverse перемещает элементы в обратном порядке

Свойство justify-content

Свойство justify-content определяет способ выключки (выравнивания по ширине) flex-элементов в строке. Это свойство применимо только в тех случаях, если для flex-элементов указана ширина и если суммарная ширина элементов меньше, чем ширина flex-контейнера. Если вы не ограничиваете ширину flex-элементов, то свойство justify-content не окажет никакого эффекта. Существует пять возможных значений свойства.

- flex-start — выравнивает элементы по левому краю строки (рис. 17.5, 1). Понимается, если вы присвоили свойству flex-flow значение row-reverse, все элементы выравниваются по *правому* краю строки.
- flex-end — выравнивает элементы по правому краю строки (см. рис. 17.5, 2). Понимается, если вы присвоили свойству flex-flow значение row-reverse, все элементы выравниваются по *левому* краю строки.
- center — выравнивает flex-элементы по центру контейнера (см. рис. 17.5, 3).
- space-between — равномерно распределяет flex-элементы, разделяя пространство между ними поровну, поместив первый элемент у левого края строки, а правый — у правого (см. рис. 17.5, 4). Это значение отлично подходит для отображения строки кнопок, которые заполняют всю ширину контейнера, например кнопок панели навигации, которая равномерно охватывает верхнюю часть страницы, или для отображения нумерованных ссылок в нижней части блога.
- space-around — равномерно распределяет оставшееся пространство между всеми элементами, добавляя его также и по левому и правому краям крайних элементов (см. рис. 17.5, 5).

Чтобы использовать это свойство, необходимо добавить его в стиль, форматизирующий flex-контейнер. Кроме того, необходимо убедиться, что для всех элементов указана ширина. Например, чтобы выровнять элементы, как показано на изображении 5 на рис. 17.5, можно использовать следующий CSS-код:

```
.container {
  display: flex;
  justify-content: space-around;
}
.container div {
  width: 200px;
}
```

Свойство align-items

Свойство align-items определяет, как flex-элементы различной высоты будут выровнены по высоте строки в flex-контейнере. По умолчанию flex-элементы растягиваются до одинаковой высоты, чтобы заполнить контейнер (рис. 17.6, 5). Тем не менее существуют и другие варианты.

- flex-start — выравнивает верхний край всех flex-элементов по верхнему краю контейнера (см. рис. 17.6, 1).

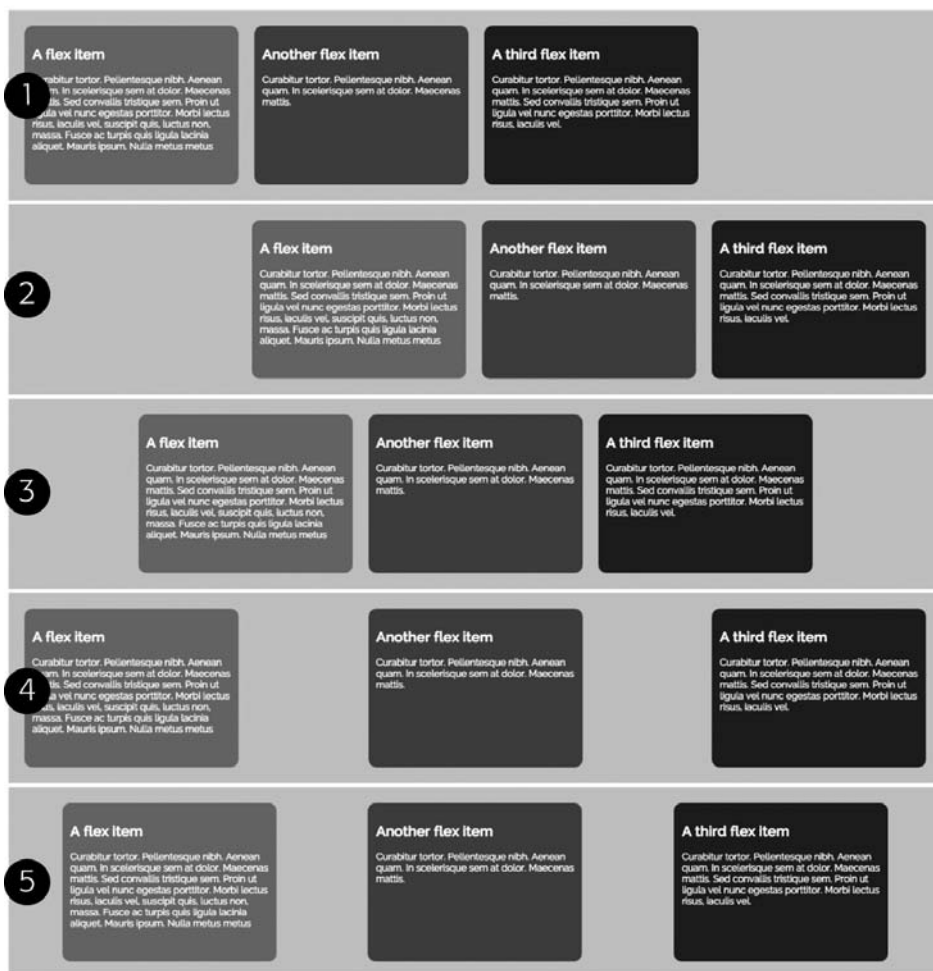


Рис. 17.5. Свойство `justify-content` полезно в тех случаях, когда ширина контейнера превышает суммарную ширину элементов внутри него. Оно позволяет выровнять по ширине их положения внутри контейнера

- `flex-end` — выравнивает нижний край всех flex-элементов по нижнему краю контейнера (см. рис. 17.6, 2).
- `center` — выравнивает вертикальный центр всех flex-элементов по вертикальному центру контейнера (см. рис. 17.6, 3).
- `baseline` — выравнивает базовые линии всех flex-элементов по базовой линии первого элемента (см. рис. 17.6, 4).
- `stretch` — обычное поведение flex-элементов. Растягивает каждый элемент по высоте контейнера, делая их высоты одинаковыми (см. рис. 17.6, 5). Этого эффекта особенно трудно достичь с использованием других методов каскадных таблиц стилей.



Рис. 17.6. Свойство `align-items` полезно в тех случаях, когда flex-элементы имеют разную высоту, а вы хотите выровнять их по вертикали (по высоте) в контейнере. Вы также можете управлять `align-items` каждым из них с помощью свойства `align-self`, которое будет рассматриваться далее в этой главе

Эти настройки приводят к другому эффекту, если в качестве направления для flex-контейнера используется значение `column`. Тогда свойство `align-items` определяет позиции элементов внутри flex-контейнера по горизонтали, в котором они будут располагаться друг за другом и иметь разную ширину.

Чтобы использовать свойство `align-items`, добавьте его в стиль, формирующий flex-контейнер. Например, чтобы получить дизайн, показанный на изображении 2 на рис. 17.6, можно использовать следующий CSS-код:

```
.container {  
  display: flex;  
  align-items: flex-end;  
}
```

Свойство `align-content`

Последнее свойство, которое можно применить к flex-контейнеру, — это `align-content`. Оно определяет, как браузер будет размещать flex-элементы, занимающие несколько строк. Это свойство работает только в случае соблюдения двух условий: во-первых, к flex-контейнеру должно быть применено значение `wrap`; во-вторых, flex-контейнер должен быть выше, чем строки flex-элементов. Другими словами, внутри контейнера должно быть дополнительное пространство по вертикали, которое больше, чем сумма всех высот строк элементов. Подобная ситуация возникает нечасто, тем не менее рассмотрим ее.

Свойство `align-content` может принять одно из шести значений.

- `flex-start` — помещает строки flex-элементов у верхнего края flex-контейнера (рис. 17.7, 1).
- `flex-end` — помещает строки flex-элементов у нижнего края flex-контейнера (см. рис. 17.7, 2).
- `center` — выравнивает центры всех строк по вертикальному центру контейнера (см. рис. 17.7, 3).
- `space-between` — равномерно распределяет дополнительное пространство по вертикали между строками, помещая верхнюю строку у верхнего края контейнера, а нижнюю — у нижнего (см. рис. 17.7, 4).
- `space-around` — равномерно распределяет оставшееся пространство между всеми строками, добавляя его также и по верхнему и нижнему краям крайних строк (см. рис. 17.7, 5).
- `stretch` — обычное поведение строк flex-элементов. Значение растягивает каждый элемент в строке таким образом, чтобы он соответствовал высоте других элементов в строке. Следует отметить, что, в зависимости от содержимого каждого элемента, строки могут быть разной высоты. Например, на изображении 6 на рис. 17.7 элементы нижней строки содержат меньше контента, чем элементы верхней строки.

Чтобы использовать свойство `align-content`, добавьте его в стиль, формирующий flex-контейнер. Кроме того, необходимо убедиться, что свойству `flex-flow`



Рис. 17.7. Свойство `align-content` полезно только в тех случаях, когда flex-контейнер и flex-элементы соответствуют определенным критериям: к flex-контейнеру применено значение `wrap`, flex-элементы размещены на нескольких строках, а высота контейнера больше, чем сумма высот всех строк с flex-элементами

присвоено значение `wrap`, а высота контейнера больше, чем высота строк элементов. Например, чтобы получить дизайн, показанный на изображении 5 рис. 17.6, используйте следующий CSS-код:

```
.container {
  display: flex;
  flex-flow: row wrap;
  align-content: space-between;
  height: 600px;
}
```

ПРИМЕЧАНИЕ

Flex-контейнеры не являются блочными элементами, поэтому некоторые свойства нельзя применить к flex-контейнерам или flex-элементам. Например, свойство `column` нельзя применить к flex-контейнеру, а свойства `float` и `clear` — к flex-элементам.

Свойства flex-элементов

Настройка свойств flex-контейнера — это только начало. Существуют дополнительные свойства, которые можно применить к flex-элементам (непосредственным дочерним элементам flex-контейнера). Указанные свойства определяют порядок отображения элементов, их ширину и способ выравнивания внутри контейнера.

Свойство order

В начале развития Всемирной паутины контент отображался от верхней части окна браузера до нижней в соответствии с тем, как HTML-код был указан в файле. Этот прямолинейный подход был прекрасен, когда вам было необходимо опубликовать текст научной статьи. После того как Всемирную паутину стали осваивать дизайнеры, они начали организовывать контент в строки и колонки, используя методы, описанные в этой книге: сначала таблицы, а затем обтекаемые элементы с помощью каскадных таблиц стилей.

Несмотря на то что дизайнеры разработали способы, позволяющие разбивать на колонки поток контента, которым по умолчанию была представлена любая веб-страница, они всегда зависели от HTML-кода. Например, если требовалось создать страницу, на которой в одной колонке отображается основной контент, а две других являются боковыми панелями, расположенными по краям от основного контента, общий подход заключался в создании трех элементов `div` и их выравнивании с целью создать колонки:

```
<div class="sidebar1">
  <!-- навигация и дополнения -->
</div>
<div class="main">
  <!-- основной контент -->
</div>
<div class="sidebar2">
  <!-- еще дополнения -->
</div>
```

Проблема этого подхода заключается в том, что первая боковая панель, которая содержала, например, элементы навигации, дополнения или даже рекламу, должна быть указана первой в исходном коде. Это означает, что поисковым системам, таким как Google, при посещении страницы придется сканировать внушительное количество лишнего контента, прежде чем они доберутся до основного контента, размещенного во второй колонке. Кроме того, программам экранного доступа, используемым посетителями с ограничениями зрения, также придется прочесть уйму дополнительной информации, прежде чем они доберутся до самой публикации.

Существуют хитрые способы, позволяющие обойти эту проблему, например отрицательные значения свойства `margin`, которые помогают изменить порядок следования колонок, но всем им далеко до элегантности и удобства. Должен существовать другой выход. И благодаря методу flexbox-верстки он есть. Свойство `order`

позволяет назначать числовое значение приоритета flex-элемента, которое определяет его положение в строке (или колонке). Порядок следования HTML-кода в исходном файле не играет роли: вы можете сделать так, чтобы последний блок HTML-кода отображался в начале строки, или первый блок — в конце (рис. 17.8, *внизу*).

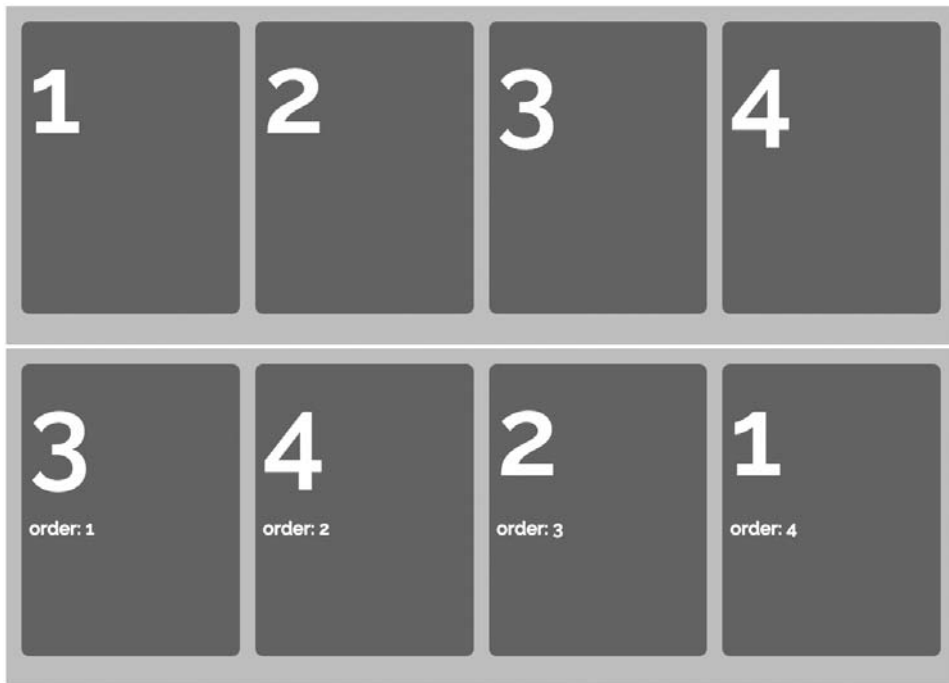


Рис. 17.8. Обычно порядок следования HTML-кода определяет последовательность отображения элементов на экране (*вверху*). Однако с помощью свойства `order` можно отобразить flex-элементы в любом порядке

Например, вы хотели бы создать страницу, содержащую три колонки: слева и справа боковые панели, а между ними — область основного контента. Можете изменить HTML-код, сделав его более «дружественным» для поисковых систем и программ экранного доступа, поместив область основного контента в начало файла:

```
<div class="content">
  <div class="main">
    <!-- основной контент -->
  </div>
  <div class="sidebar1">
    <!-- навигация и дополнения -->
  </div>
  <div class="sidebar2">
    <!-- еще дополнения -->
  </div>
</div>
```

Затем можно преобразовать внешний элемент `div` (элемент `div` с контентом) в flex-контейнер и использовать свойство `order` для размещения элементов `div` в нужном порядке на странице:

```
.content {
  display: flex;
}
.sidebar1 {
  order: 1;
}
.main {
  order: 2;
}
.sidebar2 {
  order: 3;
}
```

Теперь, хотя элемент `div` с основным контентом указан первым в HTML-коде, он отображается между двумя боковыми панелями, в то время как первая боковая панель находится в левой части flex-контейнера. Все просто!

Числа, которые используются для указания значения свойства `order`, аналогичны значениям свойства `z-index`. То есть вам не нужно использовать именно числа 1, 2 или 3. Браузер разместит элементы, начиная от наименьшего числа к наибольшему. Например, в приведенном выше примере числа 1, 2 и 3 можно заменить на 10, 20 и 30 или 5, 15, и 60. Однако для удобства лучше использовать простую систему; рекомендую использовать числа 1, 2, 3, 4 и т. д.

Тем не менее иногда вам может потребоваться сместить ту или иную колонку влево или вправо в строку. В этом случае измените число возле этого конкретного элемента, не затрагивая другие. Например, указанный выше код можно упростить и сместить первую боковую панель в крайнее левое положение:

```
.content {
  display: flex;
}
.sidebar1 {
  order: -1;
}
```

Значение `-1` смещает объект к левому краю flex-контейнера перед остальными элементами в строке. Наоборот, вы могли бы переместить эту боковую панель в крайнее правое положение строки, присвоив его свойству `order` значение 1, не устанавливая свойство `order` для других элементов.

ПРИМЕЧАНИЕ

Если в качестве направления потока контента flex-контейнера использовать значение `column`, то свойство `order` будет определять порядок следования элементов сверху вниз: flex-элементы с меньшим номером отображаются выше flex-элементов с большим номером. Кроме того, значения `column-reverse` и `row-reverse` изменяют порядок следования элементов в строках и колонках на противоположный. Учитывайте этот факт при присвоении значений свойству `order`.

Свойство align-self

Свойство `align-self` работает аналогично свойству `align-items`, которое используется для flex-контейнеров. Отличие в том, что `align-item` применяется ко всем flex-элементам в контейнере, а `align-self` — к отдельным. Свойство, примененное к элементу (не контейнеру), замещает любое установленное значение свойства `align-items`. Другими словами, к примеру, можно выровнять по верхнему краю контейнера все flex-элементы, кроме одного, который, в свою очередь, будет выровнен по нижнему краю.

Свойство `align-self` может принимать те же значения, что и свойство `align-items`, — оно оказывает тот же эффект, только на отдельные flex-элементы (рис. 17.9).

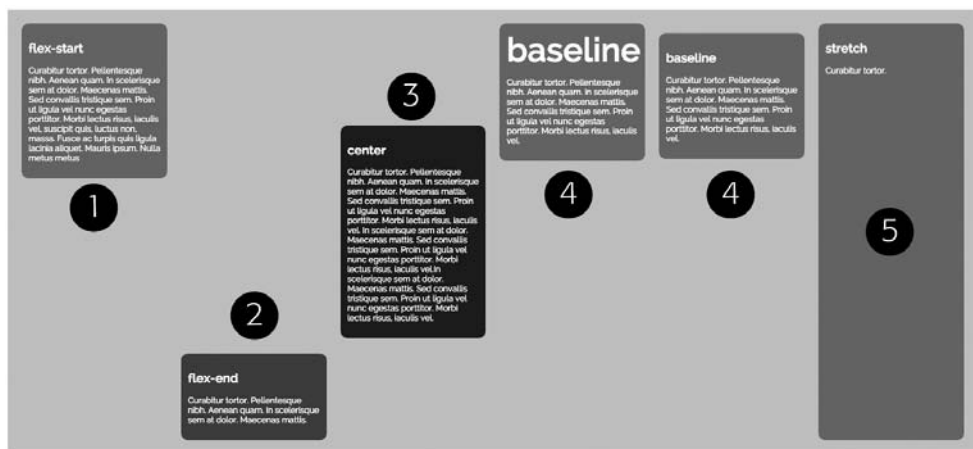


Рис. 17.9. Свойство `align-self` аналогично свойству `align-items`, но применяется к отдельным элементам. Кроме того, значение `baseline` имеет смысл применять к нескольким элементам, а не к одному (4)

Мини-практикум: автоматически настраиваемые поля и flex-элементы

Рассмотрим еще одну удивительную особенность flex-элементов: их поля не схлопываются. На первый взгляд это может показаться неважным, но такое поведение означает, что вы можете присваивать значения `auto` свойству `margin`, чтобы добавлять поля, которые будут управлять свободным пространством. Вы видели нечто подобное, когда использовали правило `margin: 0 auto;`, чтобы выровнять элемент по центру веб-страницы (см. раздел «Свойства flex-контейнера» выше).

Чтобы лучше понять, как работают автоматически настраиваемые поля и чем они привлекательны, выполним небольшое упражнение. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами (нажав кнопку `Download ZIP` в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 17.

1. В текстовом редакторе откройте файл `nav-bar.html`, расположенный в папке `17/nav-bar`.

Это простой HTML-файл. Он содержит разделы заголовка и тела и HTML-код баннера. В нем же находится CSS-код с основными стилями. Страница показана вверху на рис. 17.10. Интересующий нас HTML-код выглядит следующим образом:

```
<div class="banner">
  <p class="logo">Наша компания</p>
  <a href="#">Клиенты</a>
  <a href="#" class="highlight">0 нас</a>
  <a href="#">Вакансии</a>
</div>
```

Он содержит элемент `div`, внутри которого находятся абзац и три ссылки. В первых, преобразуем элемент `div` в flex-контейнер.

2. В разделе заголовка страницы находится пустой элемент `style`. Щелкните кнопкой мыши между открывающим и закрывающим тегами элемента `style` и добавьте следующий стиль:

```
.banner {
  display: flex;
}
```

Этот код превращает элемент `div` баннера в flex-контейнер, а абзац и ссылки, расположенные в нем, — в flex-элементы. Если вы сейчас сохраните и просмотрите страницу, то она будет выглядеть так, как показано на втором сверху изображении на рис. 17.10. Кнопки навигации слишком высоки. Это нормальное поведение flex-элементов: они увеличиваются, чтобы все элементы стали одинаковой высоты. С помощью свойства `align-items` можно изменить это поведение и выровнять кнопки по нижнему краю.

3. Измените стиль `.banner`, добавив свойство `align-items`:

```
.banner {
  display: flex;
  align-items: flex-end;
}
```

Значение `flex-end` свойства `align-items` выравнивает нижнюю часть flex-элементов по нижнему краю flex-контейнера. Панель навигации выглядит так, как показано на третьем сверху изображении на рис. 17.10. Вы могли бы достичь подобного эффекта, сделав абзац и расположенное в нем название компании обтекаемым объектом, но для выравнивания кнопок по нижнему краю потребовался бы слишком сложный CSS-код. В методе flexbox-верстки эту работу выполняет лишь одно свойство.

Наконец, чтобы выровнять логотип компании по левому краю, а кнопки навигации — по правому, мы используем автоматически настраиваемые поля.

4. Добавьте стиль `.logo` во внутреннюю таблицу стилей страницы. Окончательный код должен выглядеть следующим образом (добавленный текст выделен полужирным шрифтом):

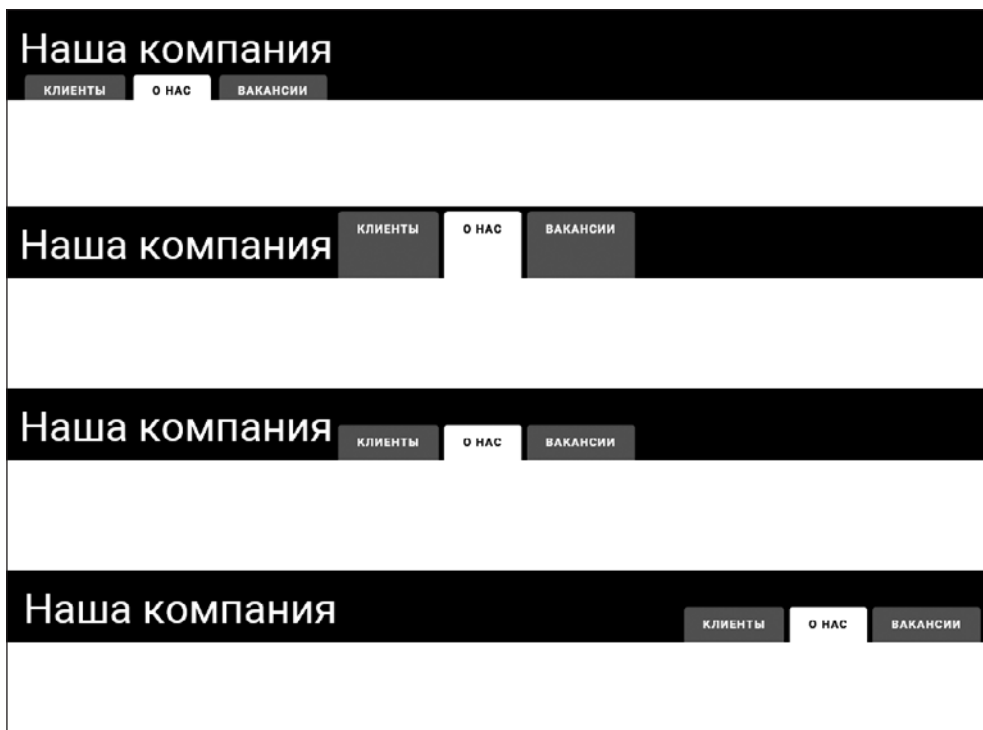


Рис. 17.10. Модуль flexbox позволяет легко создавать общие элементы интерфейса, такие как этот баннер, состоящий из логотипа и трех кнопок навигации. Для выравнивания логотипа компании по левому краю баннера, а кнопок — по правому и нижнему краям баннера необходимо добавить всего несколько свойств и поместить кнопку в нижней части баннера

```
<style>
.banner {
  display: flex;
  align-items: flex-end;
}
.logo {
  margin-right: auto;
}
</style>
```

Автоматически настраиваемые поля flex-элементов определяют свои размеры на основе доступного пространства. В данном случае логотип и три кнопки навигации не заполняют весь баннер, поэтому свойство `margin-right: auto;`, примененное к логотипу, дает инструкцию браузеру расположить доступное пустое пространство внутри баннера, справа от надписи **Наша компания**. В результате кнопки навигации смещаются к противоположному краю баннера. Удивительно!

Готовый баннер показан на нижнем изображении на рис. 17.10. Вы можете добиться аналогичного эффекта, присвоив значение `auto` свойству `margin-left` первой кнопки. При этом все доступное пространство расположилось бы слева от первой кнопки, сместив остальные кнопки вправо.

ПРИМЕЧАНИЕ

Для просмотра примера более сложной панели навигации, адаптируемой под мобильные устройства и основанной на методе flexbox и медиазапросах, обратитесь к файлу `nav-bar.html` в папке `17_finished/nav-bar-responsive`.

Свойство flex

Со свойствами, которые вы уже узнали, доступны многие интересные возможности верстки веб-страниц. Тем не менее именно свойство `flex` обеспечивает гибкость flexbox-дизайнов. Оно является ключевым в управлении шириной flex-элементов; позволяет легко создавать «гибкие» колонки или изменять их ширину в соответствии с размером контейнера, даже если размер неизвестен или меняется динамически. Таким образом, свойство `flex` позволяет быстро создавать страницы с адаптивным дизайном наподобие изученных в главе 15, требуя от вас гораздо меньше действий.

К сожалению, свойство `flex` может немного запутать, так как сочетает в себе сразу три flex-свойства. В этой главе мы рассмотрим каждое свойство по отдельности, чтобы разобраться в их действии.

Первое значение свойства `flex` — число параметра `flex-grow`, которое указывает на относительную ширину flex-элемента. Например, если внутри flex-контейнера находится три элемента `div`, вы можете присвоить каждому из них значение 1, чтобы сделать их ширину одинаковой:

```
.container {
  display: flex;
}
.container div {
  flex: 1;
}
```

Поскольку все три элемента `div` имеют одинаковое значение, то их ширина будет равна (рис. 17.11, *вверху*). Используемая вами величина — относительная, а не абсолютная. На верхнем изображении рис. 17.11 ширина каждого flex-элемента составляет примерно 33% от ширины контейнера. Если значение одного из элементов `div` равно 2, то ширина элементов изменится (см. рис. 17.11, *посередине*). Ширина двух элементов `div`, расположенных с левой стороны, равна 1, в то время как ширина правого элемента равна 2. Другими словами, он в два раза шире остальных элементов и занимает половину пространства flex-контейнера.

Таким образом, фактическая ширина flex-элемента зависит как от первого значения свойства `flex` в вашем CSS-коде, так и от количества flex-элементов в контейнере. Например, на верхнем и нижней изображениях рис. 17.11 первое значение свойства `flex` всех flex-элементов равно 1. Но в верхнем примере каждый элемент занимает около 33% от ширины контейнера, а в нижнем — около 50%. Они имеют одинаковое первое значение свойства `flex`, но количество элементов определяет точную ширину каждого из них.

Второе значение свойства `flex` — также число, определяющее параметр `flex-shrink`. Это свойство применимо, если ширина flex-контейнера меньше суммарной ширины flex-элементов внутри него. В этом случае свойство `flex-shrink` определяет, насколько узким может быть flex-элемент — то есть насколько он может сжаться

ся. Это зависит от ширины flex-элементов, которая определяется последним значением свойства `flex`. В следующем подразделе мы поговорим об этом значении, а затем вернемся к принципу работы свойства `flex-shrink`.

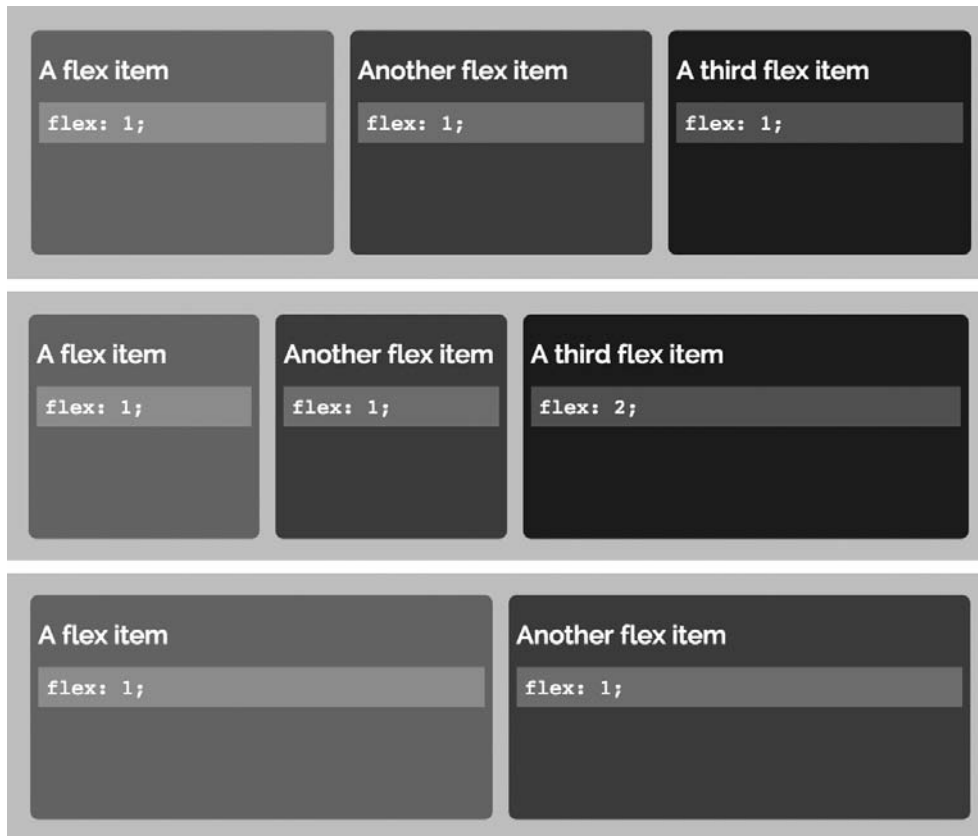


Рис. 17.11. Свойство `flex` заставляет все flex-элементы в строке увеличиваться или уменьшаться, чтобы заполнить все доступное пространство контейнера

ПРИМЕЧАНИЕ

На сайте tinyurl.com/qzwnflh находится статья, подробно описывающая использование свойства `flexbox`.

Последнее значение — свойство `flex-basis`, которое определяет базовую ширину flex-элемента. Вы можете использовать абсолютное значение, например `100px` или `5em`, или значение в процентах: `50%`. Свойство `flex-basis` можно трактовать как *минимальную* ширину flex-элемента. Если вы укажете значение `flex-basis`, оно установит ширину конкретного элемента, но, в зависимости от других flex-параметров, flex-элемент может стать шире (или уже), чем значение `flex-basis`.

Например, на верхнем изображении на рис. 17.12 значение свойства `flex-basis` каждого элемента составляет 250 пикселей. Первое значение, которое определяет,

насколько элемент растягивается или сжимается, равно 0. Если оно равно 0, элемент не изменяет своих размеров, поэтому его ширина совпадает со значением свойства `flex-basis` и составляет 250 пикселей.

Тем не менее, если в качестве первого значения вы присвоите число больше 0, строка будет заполнена на всю ширину. Таким образом, на центральном изображении на рис. 17.12, хотя значение `flex-basis` и составляет 250 пикселей, фактическая ширина элементов изменяется, так как значения свойства `flex-grow` различны для каждого элемента и составляют 1, 2 и 4 соответственно.

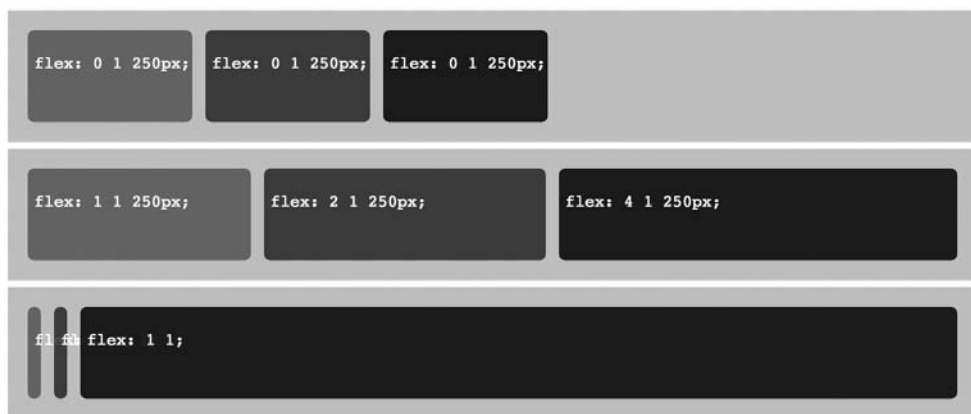


Рис. 17.12. Если вы не присвоите свойству `flex-basis` никакого значения, а свойству `flex-grow` зададите значение 0, то элемент сожмется до минимальных размеров

Математика процесса

Как же браузер вычисляет, какой должна быть ширина элемента, если свойствам `flex-grow` и `flex-basis` присвоены некие значения? Самое время заняться математикой. Эта арифметика несложна, но вам придется складывать, вычитать, делить и умножать.

Взгляните на рис. 17.13. На нем изображен `flex`-контейнер с тремя `flex`-элементами. Поскольку значения свойства `flex-grow` каждого из элементов больше 0, элементы расширяются, чтобы занять по ширине весь контейнер (ширина контейнера составляет 1000 пикселей). Значения свойств `flex-basis` каждого из элементов различны и составляют 300, 200 и 100 пикселей соответственно. Для начала вычислим сумму минимальной ширины каждого элемента: $300 + 200 + 100 = 600$.

Результат (600) является общей шириной элементов. Однако ширина контейнера превышает это значение и составляет 1000 пикселей. Таким образом, путем вычитания полученного нами результата из ширины контейнера можно определить дополнительное пространство: $1000 - 600 = 400$.

Другими словами, доступно 400 пикселей дополнительного пространства в окне браузера и браузер должен выяснить, что с ним делать. Поэтому он обращается к зна-

чениям свойства `flex-grow` каждого элемента (которые равны 1, 3 и 4 соответственно) и определяет, как разделить оставшееся пространство. Значения 1, 3 и 4 в сумме дают 8, поэтому первый элемент должен получить $1/8$ дополнительного пространства; $1/8$ от 400 (то же самое, что $400 / 8$) равняется 50. Ширина первого элемента состоит из суммы значения свойства `flex-basis` и доли дополнительного пространства, приходящегося на него: $300 + 50 = 350$ пикселей.

Ширина второго элемента состоит из суммы значения свойства `flex-basis` и доли дополнительного пространства, приходящегося на него: $200 + (50 \times 3) = 350$ пикселей.

А к ширине последнего элемента необходимо прибавить $1/2$ от 400 пикселей: $100 + 200 = 300$ пикселей.

Этот метод требует выполнения кое-каких математических действий. Как вы видите, браузер должен выполнить некоторые вычисления, чтобы определить ширину flex-элементов. Далее описаны несколько способов определения значений свойства `flex`.

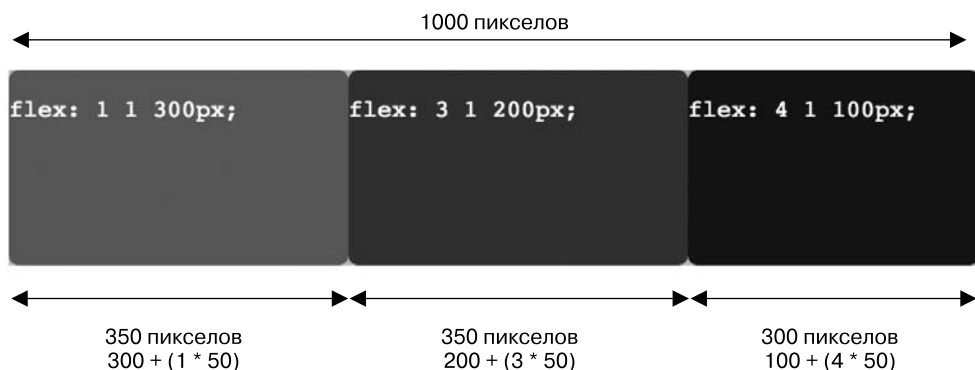


Рис. 17.13. Фактическая ширина flex-элемента зависит от ширины flex-контейнера и значений свойств `flex-grow` и `flex-basis`

Свойство `flex-shrink`

Второе значение свойства `flex` определяет, насколько flex-элемент может быть сжат, если суммарная ширина элементов *больше* ширины контейнера. Это значение важно только в том случае, если свойству `flex-flow` контейнера присвоено значение `row`, согласно которому все элементы должны находиться рядом друг с другом в одной строке.

Например, flex-элементы, изображенные на рис. 17.14, занимают 1000 пикселей в ширину. Поскольку значение свойства `flex-basis` каждого из них составляет 400 пикселей (общая ширина — 1200 пикселей), они не поместятся внутри контейнера без сжатия. В первой строке значение `flex-shrink` всех элементов равно 1. Это означает, что все они должны сжиматься одинаково.

ИНФОРМАЦИЯ ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Значения свойства flex по умолчанию

Если вы не укажете свойство `flex` для flex-элементов в flex-контейнере, браузер применит его со *стандартными* настройками:

```
flex: 0 1 auto;
```

Благодаря этим настройкам ширина каждого flex-элемента автоматически определяется его контентом. Flex-элемент, содержащий объемный текст или изображение, будет сжат меньше, чем тот, что содержит только пару слов.

Однако браузеры применяют различные настройки по умолчанию, если вы явно назначаете свойство `flex` элементу.

Свойство `flex` — это сокращенная запись трех следующих свойств: `flex-grow`, `flex-shrink` и `flex-basis`. Поэтому строка кода:

```
flex: 1 1 400px;
```

идентична такой записи:

```
flex-grow: 1;
flex-shrink: 1;
flex-basis: 400px;
```

В тех случаях, если вы не указываете значения свойств `flex-shrink` и `flex-basis`, скажем `flex: 1;`, браузер назначает значение 1 свойству `flex-shrink` по умолчанию, а свойству `flex-basis` — значение 0%. Другими словами, код:

```
flex: 1;
```

идентичен следующей записи:

```
flex-grow: 1;
flex-shrink: 1;
flex-basis: 0%;
```

Эти настройки существенно отличаются от тех, которые браузер использует по умолчанию в случаях, если свойство `flex` не указано полностью. При присвоении свойству `flex-basis` значения 0% ширина каждого flex-элемента полностью определяется свойством `flex-grow`. Другими словами, объем контента внутри каждого flex-элемента не влияет на ширину самого элемента.

Чтобы получить более подробную информацию о положительных и негативных сторонах использования flexbox-верстки, посетите сайт Консорциума W3C: tinyurl.com/o2eaava.

flex-flow: row nowrap



Рис. 17.14. Если значение свойства `flex-shrink` равно 0, а свойству `flex-flow` контейнера присвоено значение `nowrap`, то flex-элементы в строке будут расширяться за пределы контейнера (внизу)

Тем не менее на центральном изображении на рис. 17.14 значение свойства `flex-shrink` первого элемента равно 1, в то время как значения этого же свойства других элементов равны 4. Таким образом, ширина объектов различна. В отличие от свойства `flex-grow`, от значения которого зависит ширина элемента, значение свойства `flex-shrink` определяет, насколько *узким* элемент может быть по отношению к другим элементам в этой строке. На центральном изображении на рис. 17.14 два элемента справа узкие, поскольку значение их свойства `flex-shrink` выше, чем у элемента слева.

Значение `flex-shrink` добавляет свои сложности в понимание принципа работы свойства `flex`. По этой причине, а также потому, что он не оказывает никакого эффекта, если свойству `flex-flow` контейнера присвоено значение `nowrap`, в большинстве случаев используйте значение 1 для свойства `flex-shrink`.

ПРИМЕЧАНИЕ

Свойство `flex-shrink` начинает действовать только в тех случаях, когда элементы в контейнере не переносятся на следующую строку. Другими словами, если код свойства `flex-flow` выглядит так:

```
.container {
  display: flex;
  flex-flow: row nowrap;
}
```

В противном случае flex-элемент переносится на следующую строку, если ему не хватает места в текущей (рис. 17.15).

Обтекание flex-элементов

Свою полную силу свойство `flex-basis` демонстрирует, если вы допустили перенос элементов в flex-контейнере:

```
.container {
  display: flex;
  flex-flow: row wrap;
}
```

В этом случае flex-элементы переносятся на другую строку, если они не помещаются в контейнере. Например, представьте, что у вас есть гибкий контейнер, ширина которого составляет 1000 пикселей, и три flex-элемента, свойству `flex-basis` каждого из которых присвоено значение 400px. Поскольку $400 + 400 + 400$ больше, чем 1000 пикселей, все три flex-элемента не помещаются в контейнере. Однако, поскольку $400 + 400$ меньше, чем 1000 пикселей, два из них без проблем уместятся на одной строке.

В этом случае браузер будет отображать первые два элемента на одной строке и перенесет третий элемент на вторую строку (см. рис. 17.15, *вверху*). Поскольку каждому из трех элементов назначено свойство `flex-grow`, они заполнят всю строку, а нижний элемент, одинокий на второй строке, растянется, чтобы заполнить контейнер по всей ширине.

Если контейнер сожмется так, что flex-элементам не хватит пространства рядом друг с другом, браузер поместит каждый из них на отдельной строке (см. рис. 17.15, *внизу*).

flex-flow: row wrap



Рис. 17.15. Свойство `flex-basis` используется для установки базовой ширины flex-элемента.

Оно очень удобно, если flex-контейнер недостаточно широк для того, чтобы элементы располагались рядом друг с другом. Свойство `flex-flow` позволяет переносить flex-элементы на следующую строку (внизу)

Рекомендации по flexbox-верстке

Как вы могли заметить, существует множество факторов, которые следует учитывать при использовании всех трех значений свойства `flex`. Каждое из них взаимодействует по-разному, в зависимости от ширины контейнера, и пытаться предусмотреть все возможные комбинации может быть крайне затруднительно. Тем не менее ниже перечислены рекомендации, которые могут помочь вам при использовании этого свойства.

- **Заливайте все flex-элементы в одну строку.** Если вы хотите создать строку элементов различной ширины, запрещайте перенос элементов в flex-контейнере и используйте только одно значение свойства `flex`. Например, в строке расположены две боковые панели и элемент с основным контентом. Вы хотите, чтобы область с основным контентом занимала половину доступной ширины

контейнера, а каждая из боковых панелей — по 25 %. Вы могли бы использовать следующий код:

```
.container {
  display: flex;
  flex-flow: row nowrap;
}
.sidebar1 {
  flex: 1;
}
.sidebar2 {
  flex: 1;
}
.main {
  flex: 2;
}
```

В данном случае нет необходимости применять свойство `flex-shrink` или `flex-basis`, поскольку вы используете несколько flex-элементов, ширина которых изменяется пропорционально.

- **Сохраняйте пропорции строк, но добавляйте перенос в случаях, когда контейнер слишком мал и не позволяет отобразить все элементы рядом друг с другом.** Если вы хотите, чтобы flex-элементы переносились на новую строку, когда их содержимое не помещается в одной строке, присваивайте значение `wrap` свойству `flex-flow` контейнера и значение свойству `flex-basis` конкретного элемента с соответствующим коэффициентом `flex-grow`:

```
.container {
  display: flex;
  flex-flow: row wrap;
}
.sidebar1 {
  flex: 1 1 100px;
}
.sidebar2 {
  flex: 1 1 100px;
}
.main {
  flex: 2 1 200px;
}
```

Чтобы убедиться, что flex-элементы сохраняют пропорции по ширине, определенные свойством `flex-grow`, необходимо установить значения `flex-basis`, которые соответствуют пропорциям свойства `flex-grow`. Например, в коде, представленном выше, содержится три flex-элемента. Значения их свойства `flex-grow` равны 1, 1 и 2 соответственно. Поэтому присвойте свойству `flex-basis` этих элементов значения с аналогичными пропорциями, например 100px, 100px и 200px. Правильные значения зависят от ширины окна браузера, в котором вы хотите выполнить перенос элементов. С учетом кода, показанного в примере, ширина flex-контейнера должна стать менее 400 пикселей, чтобы произошел перенос элементов.

Свойство `flex-grow` используется для распределения пустого пространства, оставшегося после суммирования значений ширины `flex`-элементов. Поэтому, если значения `flex-basis`, используемые вами, пропорционально не соответствуют значениям свойства `flex-grow`, конечный результат вас несколько удивит. Например, как показано в нижней части на рис. 17.16, значение свойства `flex-grow` центрального элемента равно 2. Однако он шире любого другого элемента не в два раза. Это обусловлено тем, что значение присвоенного ему свойства `flex-basis` такое же, как у двух других элементов, и составляет 100 пикселей. Поэтому, хотя к его ширине и добавляется больше свободного пространства, чем к другим элементам, браузер добавляет его к 100-пиксельной ширине — такой же, как и у других элементов.

- Свойства `flex-basis` выступают в роли точек останова, определяющих, когда элементы переносятся на новые строки. Вы можете использовать значение свойства `flex-basis` для имитации точек останова, с которыми вы познакомились при изучении адаптивного дизайна в главе 15. Например, если существует три элемента, которые вы хотите отобразить друг рядом с другом, а ширина страницы превышает 600 пикселей, можно присвоить свойству `flex-basis` каждого из них значение 200px. Если окно браузера станет уже 600 пикселей, то первый и второй элементы будут располагаться рядом друг с другом, а третий будет перенесен на новую строку. Если окно браузера станет уже 400 пикселей, то каждый из элементов будет находиться на отдельной строке, располагаясь друг над другом.

Практикум: создание flexbox-макета

В этом практикуме вы возьмете за основу макет, в котором элементы `div` располагаются друг над другом (рис. 17.17, *слева*), и сформируете колонки с помощью метода `flexbox` (см. рис. 17.17, *справа*). Кроме того, вы реализуете несколько визуальных эффектов, что будет очень легко благодаря методу `flexbox`-верстки, но потребовало бы невероятных усилий при использовании какого-либо другого метода CSS.

Чтобы начать работу, вы должны иметь в распоряжении файлы с учебным материалом. Для этого нужно загрузить файлы для выполнения заданий практикума, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив (нажав кнопку `Download ZIP` в правом нижнем углу страницы). Файлы текущего практикума находятся в папке 17.

Форматирование панели навигации

Представленный дизайн состоит из трех `flex`-контейнеров — один используется для кнопок навигации, второй — для основного контента, а третий — для нижнего колонтитула (рис. 17.18). Каждый контейнер будет содержать разное количество `flex`-элементов: четыре — для панели навигации, три — для области основного контента и два — для нижнего колонтитула. Для начала создадим `flex`-контейнеры.

1. В редакторе HTML-кода откройте файл `custom.css`, расположенный в папке `17\flexbox-layout\css`.



Рис. 17.16. Чтобы размеры flex-элементов сохранили свои пропорции, убедитесь, что значения свойства `flex-basis`, определяющего базовую ширину ваших flex-элементов, пропорционально соответствуют значениям свойства `flex-grow` этих элементов (*вверху*). Если это не так, вы не сможете точно установить пропорции при определении ширины flex-элементов в строке (*внизу*)

Это пустой CSS-файл, в который вы добавите код каскадной таблицы стилей для реализации метода flexbox-верстки. Страница `index.html` содержит ссылки на две таблицы стилей, связанные с ней: пустой файл `custom.css` и файл `base.css`, который включает базовые стили, придающие основным элементам страницы некоторое форматирование.

Сначала вы создадите основной flex-контейнер.

2. Добавьте код группового селектора в файле `custom.css` следующим образом:

```
.nav, .boxes, .footer {
  display: flex;
  flex-flow: row wrap;
}
```

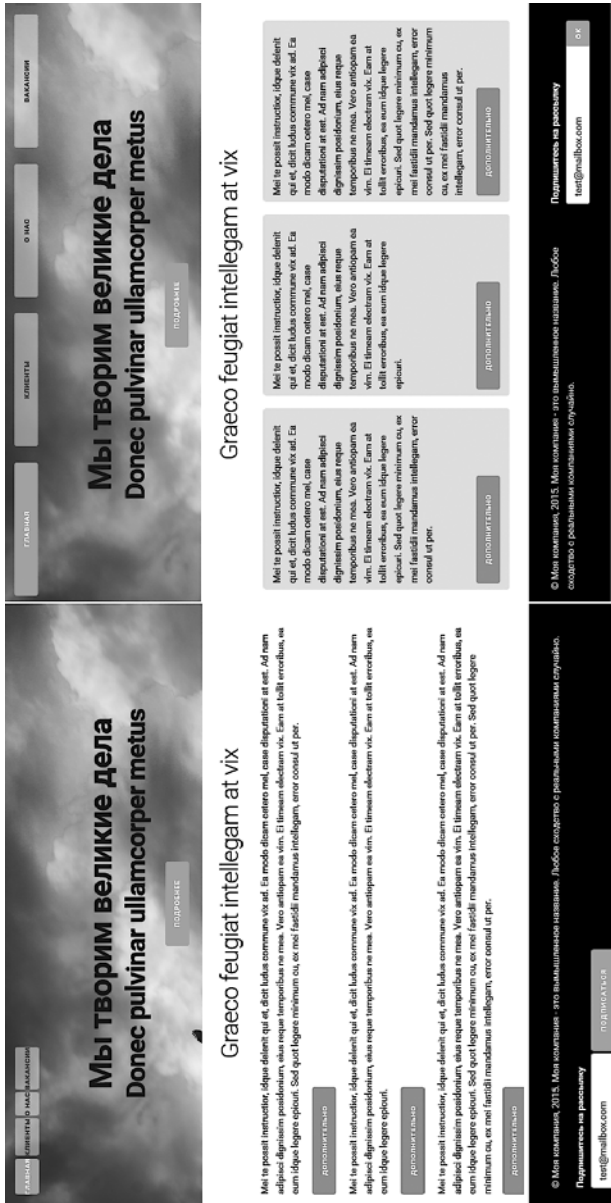


Рис. 17.17. С помощью flexbox-верстки можно создать многоколоночный дизайн и при этом не прибегать к выравниванию элементов

Каждый из этих имен классов соответствует классу, применяемому к трем областям, показанным на рис. 17.18. Это основное правило превращает элементы `div` в flex-контейнеры и отображает их в строке в виде flex-элементов, которые могут быть перенесены на дополнительную строку.

Далее вам придется выполнить некоторые довольно сложные настройки. Вы разместите кнопки навигации равномерно по контейнеру таким образом, чтобы крайняя левая кнопка находилась у левого края контейнера, правая кнопка — возле правого края контейнера, а две другие кнопки были равномерно распределены между ними.

3. Добавьте еще один стиль сразу после созданного на предыдущем шаге:

```
.nav {
  justify-content: space-between;
}
```

Свойство `justify-content`, примененное к flex-контейнеру, определяет, как flex-элементы размещаются внутри контейнера. В данном случае значение `space-between` распределяет все flex-элементы равномерно по контейнеру. Это действительно здорово и, как правило, трудноосуществимо с помощью других методов каскадных таблиц стилей.

Вы также можете придать этим кнопкам гибкую ширину, указав их размер в процентах. Сделаем это сейчас.

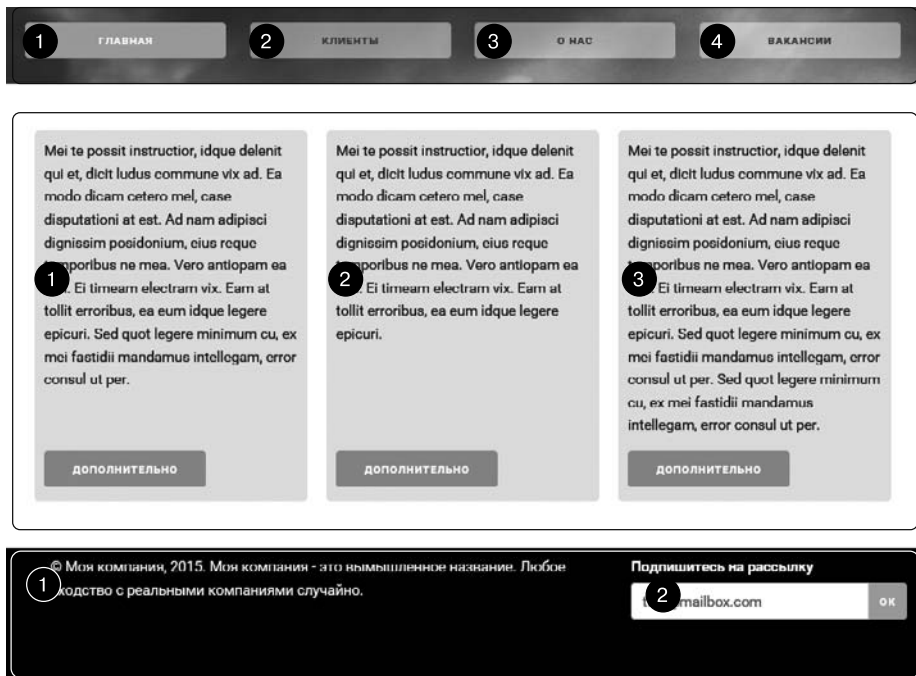


Рис. 17.18. Дизайн этой страницы содержит три flex-контейнера (верхняя, средняя и нижняя строки), в которых находятся flex-элементы (пронумерованы)

4. Отформатируйте ссылки, добавив в таблицу стилей следующий код:

```
.nav a {  
  width: 23%;  
}
```

Стиль устанавливает ширину каждой ссылки равной 23 % от общей ширины flex-контейнера. При сжатии контейнера кнопки также будут сжаты. Оставшееся пространство равномерно распределяется между каждой кнопкой благодаря стилю, который вы добавили на предыдущем шаге.

Кнопки на панели навигации распределены равномерно, и их ширина настраивается автоматически. Все это сделано без обтекаемых элементов или другого сложного CSS-кода.

Добавление трех колонок

Пришло время заняться областью основного контента. В данный момент существует три элемента `div`, каждый из которых содержит кнопку **Дополнительно**, размещенные друг над другом. Добавив свойство `flex` можно разместить их в строке. Помните, как на шаге 1 ранее вы превратили `div`, который содержит эти элементы `div`, в flex-контейнер? Теперь вы превратите их в flex-элементы.

1. В нижней части файла `custom.css` добавьте следующее правило:

```
.boxes div {  
  flex: 1 1 250px;  
}
```

С его помощью каждый элемент `div` будет занимать равную область flex-контейнера. Элементы одинаковой ширины, но в настоящее время состыкованы друг с другом, что затрудняет чтение (а также тот факт, что текст на латыни). Дизайн должен выглядеть так, как показано на верхнем изображении на рис. 17.19. Чтобы упростить чтение текста в колонках, вы добавите фоновый цвет и немного воздуха.

2. Измените стиль, созданный на предыдущем шаге, добавив в него четыре свойства (добавленный текст выделен полужирным шрифтом):

```
.boxes div {  
  flex: 1 1 250px;  
  margin: 10px;  
  border-radius: 5px;  
  padding: 10px 10px 0 10px;  
  background-color: rgba(0,0,0,.1);  
}
```

Обратите внимание, что каждая колонка имеет одинаковую высоту, даже если их содержимое — разной длины. Это еще один волшебный момент метода `flexbox`-верстки: по умолчанию высота всех колонок в строке одинакова. Это очень трудно сделать с помощью обычного CSS-кода, а некоторые дизайнеры даже прибегают к помощи JavaScript, чтобы добиться такого эффекта.

Тем не менее кнопки, так хорошо заметные на странице, находятся в хаотичных положениях, что отвлекает. Было бы здорово, если бы существовал способ поместить их в нижней части каждой колонки. Благодаря методу flexbox-верстки он есть!

Поля flex-элементов работают немного по-другому. Например, сверху каждой кнопки можно добавить автоматически настраиваемое поле, чтобы сместить кнопки к основанию колонки. Тем не менее вы можете использовать эту технику для установки полей flex-элементов. На данный момент колонки сами по себе являются flex-элементами; кнопки и абзацы внутри них — обычные элементы.

К счастью, модель flexbox позволяет назначить несколько ролей flex-элементам. Колонки могут быть flex-элементами по отношению к внешним контейнерам (строкам) и flex-контейнерами, содержащими flex-элементы (абзац и кнопка) внутри себя, одновременно.

3. Добавьте еще две строки кода в стиль `.boxes div`:

```
.boxes div {
  flex: 1 1 250px;
  margin: 10px;
  border-radius: 5px;
  padding: 10px 10px 0 10px;
  background-color: rgba(0,0,0,.1);
  display: flex;
  flex-flow: column;
}
```

Колонки должны теперь выглядеть так, как показано на центральном изображении на рис. 17.19. Этот код определяет элементы `div` как flex-контейнеры и задает направление flex-элементов внутри колонки, размещая текст и кнопки друг над другом. Flex-элементы могут использовать автоматически настраиваемые поля, поэтому вам нужно разместить кнопки в нижней части каждого элемента `div`.

4. Добавьте следующий стиль после правила `.boxes div`:

```
.boxes .more {
  margin-top: auto;
}
```

Теперь к каждой кнопке применен класс `more`, который воздействует на кнопки внутри колонок. Свойству `margin-top` присвоено значение `auto`, которое дает инструкцию браузеру автоматически добавлять пустое пространство внутри flex-элемента над кнопкой. При этом все кнопки будут смещены к нижнему краю колонок (см. рис. 17.19, *внизу*).

Форматирование колонтитула

Мы подобрались к нижней части страницы. Осталось выполнить форматирование нижнего колонтитула. В нем находится информация об авторских правах и веб-форма подписки на рассылку. Для этого контента достаточно двух колонок.



Рис. 17.19. Флекс-элементы имеют много достоинств: по умолчанию у всех колонок одинаковая высота. Этим, как правило, сложно управлять. Кроме того, пустое пространство можно распределить автоматически, благодаря чему браузер будет выполнять некоторые замечательные действия, например выравнивать кнопки в нижней части каждой колонки (внизу)

1. Добавьте два новых стиля в нижней части файла `custom.css`:

```
.footer .copyright {
  flex: 2 1 500px;
  margin-right: 30px;
}
.footer .signup {
  flex: 1 1 250px;
}
```

Эти стили увеличивают область с информацией об авторских правах в два раза по сравнению с веб-формой подписки на рассылку. Небольшое поле справа отодвигает веб-форму подписки на рассылку от уведомления об авторских правах (рис. 17.20, *вверху*).

Последнее, что осталось сделать, — это организовать элементы веб-формы. Было бы прекрасно, если бы кнопка отправки данных (ОК) находилась рядом с текстовым полем и была выровнена с ним. Это можно сделать с помощью метода flexbox, но сначала необходимо превратить элемент `form` в flex-контейнер. Затем все элементы внутри — текстовая метка, поле ввода и кнопка отправки данных — будут функционировать как flex-элементы.

2. Обновите стиль в верхней части файла `custom.css`, добавив в него селектор `.footer`:

```
.nav, .boxes, .footer, .footer form {
  display: flex;
  flex-flow: row wrap;
}
```

Этот селектор превратит веб-форму внутри нижнего колонтитула в flex-контейнер, а текстовую метку, поле ввода и кнопку отправки данных внутри веб-формы — в flex-элементы. Далее вы растянете метку на всю ширину веб-формы.

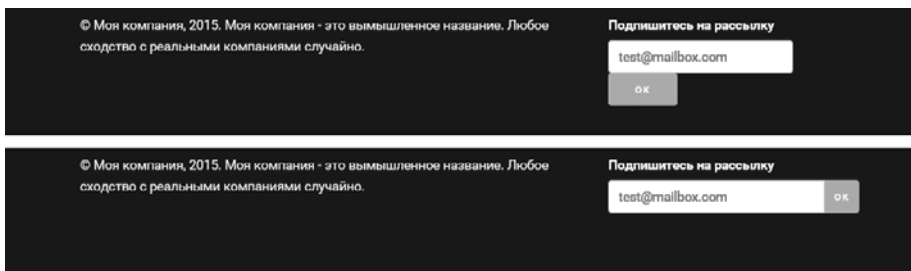


Рис. 17.20. С помощью flexbox легко выравнивать элементы веб-формы, такие как поле ввода и кнопка отправки данных (*внизу*)

3. Добавьте еще один стиль в конец файла `custom.css`, чтобы метка заполнила колонку:

```
.signup label {
  width: 100%;
}
```

На предыдущем шаге вы превратили веб-форму в flex-контейнер, для которого допускается перенос элементов. Присвоение свойству `width` метки значения `100%` переносит поле ввода и кнопку отправки данных веб-формы на другую строку. Тем не менее поле ввода и кнопка не находятся рядом друг с другом. Пара стилей исправят это.

4. Добавьте следующие два правила в конце таблицы стилей файла `custom.css`:

```
.signup input[type="email"] {  
  border-radius: 4px 0 0 4px;  
  flex: 1;  
}  
.signup input[type="submit"] {  
  border-radius: 0 4px 4px 0;  
  padding: 0 10px;  
}
```

Эти правила используют селекторы атрибутов, определяющие поле ввода адреса электронной почты и кнопку отправки данных веб-формы. Присвоение свойству `flex` поля ввода значения `1` растягивает поле до заполнения всего доступного пространства, не занятого кнопкой отправки данных (с надписью ОК).

Кроме того, с помощью свойства `border-radius` — скругления левых углов поля ввода и правых углов кнопки отправки данных — вы создали подобие виджета. Результат работы показан на нижнем изображении на рис. 17.20.

Адаптация панели навигации под мобильные устройства

Благодаря гибкости flex-элементов созданный вами дизайн великолепно смотрится в браузерах с различной шириной окна. Сохраните файл `custom.css`; откройте страницу `index.html` и измените размер окна браузера. Вы увидите, что дизайн сжимается до размеров окна браузера: три колонки в центре страницы преобразуются в две, а после и вовсе в одну колонку.

Последнее, что следует добавить, — простой медиазапрос, который будет преобразовывать кнопки панели навигации, расположенные рядом друг с другом, в колонку при малых размерах экрана.

1. В нижней части файла `custom.css` добавьте медиазапрос:

```
@media (max-width: 500px) {  
  
}
```

Он вступает в силу, если ширина окна браузера (или разрешение по горизонтали на экране устройства) не превышает 500 пикселей. Другими словами, стили, размещенные в нем, будут применяться только к устройствам (и окнам браузеров), экранное разрешение по горизонтали которых — менее 501 пиксела.

Мы добавим стиль внутри медиазапроса, позволяющий изменить отображение flex-элементов (кнопок навигации) внутри панели навигации.

2. Добавьте следующий стиль в медиазапрос (добавленный код выделен полужирным шрифтом):

```
@media (max-width: 500px) {  
  .nav {  
    flex-flow: column;  
  }  
}
```

Этот код изменит направление потока flex-элементов. Вместо того чтобы располагаться рядом друг с другом в строку, кнопки навигации разместятся друг под другом в виде колонки.

В завершение вы измените ширину кнопок, чтобы они заполняли контейнер.

3. Добавьте еще один стиль в медиазапрос (добавленный код выделен полужирным шрифтом):

```
@media (max-width: 500px) {  
  .nav {  
    flex-flow: column;  
  }  
  .nav a {  
    width: 100%;  
    margin-bottom: 2px;  
  }  
}
```

Последний добавленный стиль позволяет каждой кнопке-ссылке заполнить flex-контейнер по ширине. Он также добавляет немного пространства ниже каждой ссылки, чтобы реализовать визуальное разделение.

4. Сохраните файл `custom.css`. Откройте страницу `index.html` в браузере и постепенно уменьшайте размер окна по горизонтали.

Видите, как изменяется дизайн при уменьшении окна браузера (рис. 17.21)? Во-первых, при полной ширине (1) область основного контента содержит три колонки, а нижний колонтитул — две. Затем, когда окно браузера станет немного уже, три колонки превращаются в две, а элементы нижнего колонтитула перетекают и располагаются друг под другом (2). Наконец, при ширине окна браузера менее 500 пикселей все содержимое страницы отображается в одной колонке, в том числе и панель навигации (3).

Способ flexbox-верстки — это веселое и интересное дополнение к каскадным таблицам стилей. С его помощью многие ранее трудные (или почти невозможные) задачи, связанные с версткой макета страницы, выполняются очень легко. Он упрощает работу по созданию адаптивного дизайна и уменьшает количество математических расчетов, которые необходимо выполнить, при проектировании колонок пропорциональных размеров. Его небольшой недостаток заключается в том, что старые версии браузеров поддерживают синтаксис flexbox-кода и попросту игнорируют flex-свойства. Тем не менее поддержка реализована во всех современных браузерах, включая Internet Explorer и Edge, поэтому нет никаких причин, чтобы не начать экспериментировать с приемами flexbox-верстки в своих проектах.

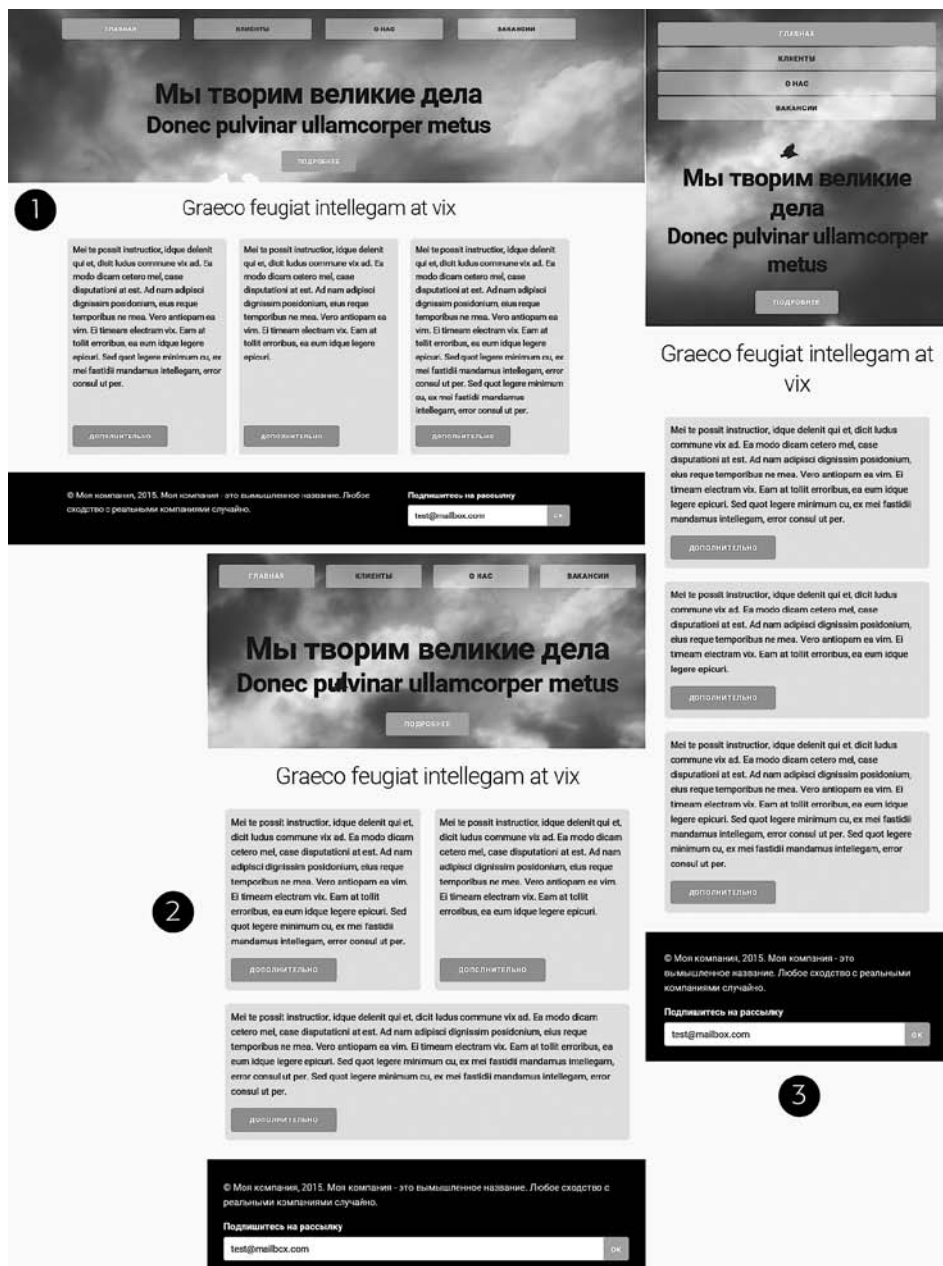


Рис. 17.21. Метод flexbox-верстки позволяет создавать «гибкий» дизайн, изменяющийся в зависимости от ширины окна браузера. А с помощью медиазапросов flexbox-страницы адаптируются под экраны мобильных устройств с различным разрешением

ЧАСТЬ IV

Профессиональные приемы CSS-верстки

Глава 18. Профессиональные приемы CSS-верстки

Глава 19. Профессиональный дизайн с помощью Sass

18 Профессиональные приемы CSS-верстки

На данный момент мы рассмотрели большинство принципов использования каскадных таблиц стилей. С применением CSS-кода при верстке веб-страниц вы можете быстро и эффективно разрабатывать сайты. Но даже теперь, когда вы овладели всеми свойствами, которые предлагают каскадные таблицы стилей, научились устранять недостатки браузеров и освоили искусные приемы разработки красивых веб-страниц, вы все еще можете изучить несколько методик, которые упростят создание, использование и поддержку вашего CSS-кода.

Эта глава содержит некоторые рекомендации по созданию и использованию каскадных таблиц стилей. Они не являются основополагающими, но могут ускорить вашу работу, уменьшая риски разочарования и повышая производительность.

Добавление комментариев

Если приходится редактировать таблицу стилей через недели, месяцы или даже годы после того, как она была создана, вы можете задаться вопросом: «Зачем я создавал этот стиль? Что он делает?» Как в любых других проектах, создавая сайт, вы должны хранить заметки о том, что вы сделали и для чего. К счастью, для этого вам не придется исписывать кучу бумаги. Вы можете включать ваши заметки прямо в таблицы стилей с помощью *комментариев* в CSS-коде.

Комментарий в CSS-коде — это лишь заметка, содержащаяся между двумя наборами символов: /* и */. Как и в языке HTML, комментарии в CSS-коде не считаются и не выполняются браузером, но позволяют добавлять полезные напоминания к таблицам стилей. Вам не нужно комментировать в своих таблицах стилей абсолютно *все*, в конечном счете большинство свойств, таких как color, font-family, border-color и т. д., именами говорят сами за себя. Но вполне резонно будет добавить комментарий для стиля, если сразу непонятно, что именно он или определенное его свойство делает. Например, вы можете сбросить блочную модель CSS, чтобы ширина и высота элемента вычислялись с учетом границ и отступов:

```
* {  
  box-sizing: border-box;  
}
```

В то время, когда вы писали стиль, вы знали, что он делает. Но будете ли вы так же хорошо помнить это три месяца спустя? А что, если кому-нибудь, кто не знаком с этим приемом, когда-нибудь понадобится отредактировать ваш CSS-код? Добавьте комментарий, и вы или пользователь, который будет работать с вашим сайтом, легко выясните, что делает этот стиль и зачем он был создан:

```
/* Учитываем в размерах элемента отступы и границы */
* {
  box-sizing: border-box;
}
```

Если вы хотите оставить более обширный комментарий, то можно добавить несколько строк. Просто начните с символов `/*`, введите текст комментария, а затем завершите комментарий символами `*/`. Это удобно при добавлении сопровождающей информации в начале таблицы стилей, как показано на рис. 18.1.

```
/*! normalize.css v3.0.2 | MIT License | git.io/normalize */

/**
 * 1. Set default font family to sans-serif.
 * 2. Prevent iOS text size adjust after orientation change, without disabling
 *    user zoom.
 */

html {
  font-family: sans-serif; /* 1 */
  -ms-text-size-adjust: 100%; /* 2 */
  -webkit-text-size-adjust: 100%; /* 2 */
}

/**
 * Remove default margin.
 */

body {
  margin: 0;
}

/* HTML5 display definitions
   ========================================================================= */

/**
 * Correct `block` display not defined for any HTML5 element in IE 8/9.
 * Correct `block` display not defined for `details` or `summary` in IE 10/11
 * and Firefox.
 * Correct `block` display not defined for `main` in IE 11.
 */

article,
aside,
details,
figcaption,
figure,
footer,
header,
hgroup,
main,
menu,
nav,
section,
summary {
  display: block;
}
```

Рис. 18.1. Комментарии в CSS-коде помогают идентифицировать стили для их редактирования спустя некоторое время. Вы также можете использовать их для хранения полезной информации, которая позволяет отслеживать версии сайта или таблиц стилей или добавлять информацию об авторских правах

Имейте в виду, что добавление комментариев увеличивает объем кода в ваших файлах, что увеличивает скорость их загрузки. В действительности, прежде чем увидеть разницу в скорости загрузки файла, необходимо добавить очень много комментариев, но вы можете использовать онлайн-инструменты, такие как CSS Minifier (cssminifier.com), чтобы вырезать комментарии перед созданием CSS-файла и загрузкой его на сайт. А лучшим инструментом является Sass. Вы познакомитесь с ним в следующей главе, но, забегая вперед, скажу, что его особенность заключается в возможности оставлять комментарии в редактируемом вами CSS-коде и удалять их автоматически в CSS-файлах, которые используются на веб-страницах.

Организация стилей

О создании стилей и таблиц стилей уже было рассказано многое. Но если вы проектируете сайт, который подразумевает дальнейшую поддержку, вы можете объединить несколько принципов, которые помогут вам в будущем. Придет день, когда вы должны будете изменить вид сайта, скорректировать определенный стиль или передать свою тяжелую работу сотруднику, который станет ответственным за нее. Помимо комментариев для себя или других людей, небольшое планирование и организация CSS-кода помогут избежать различных проблем в будущем.

Тонкости присвоения имен

Вы уже изучили технические аспекты именования различных типов селекторов — имена классов начинаются с точки (.), а идентификаторы — с символа #. Кроме того, имена, которые вы присваиваете идентификаторам и классам, должны начинаться с буквы и не могут содержать такие символы, как &, * или !, а также кириллицу. Помимо выполнения этих требований, необходимо соблюдать некоторые практические правила, что может помочь вам более эффективно управлять стилями.

Присвоение имен стилям согласно назначению, а не виду

Очень заманчиво бывает использовать такие имена, как `.redhighlight`, при создании стиля для форматирования текста огненно-красным цветом шрифта, бросающегося в глаза. Но что произойдет, если вы (шеф, клиент) решите, что оранжевый, синий или бледно-зеленый будет смотреться лучше? В результате получится, что стиль, названный `.redhighlight`, на самом деле форматирует текст бледно-зеленым цветом, что, конечно, сбивает с толку. Лучше использовать имя, которое описывает *назначение* стиля. Например, если красный цвет шрифта предназначен для указания ошибок, которые допустил посетитель при заполнении формы, используйте имя `.error`. Если стиль должен оповестить посетителя о некоторой важной информации, назовите его `.alert`. В любом случае изменение цвета или другого форматирования в стиле не вызовет путаницы, так как стиль все равно предназначен для указания на ошибки или оповещения пользователей, независимо от его цвета.

Следует также избегать имен с указанием конкретных размеров, например `.font20px`. Сегодня этот шрифт может иметь размер 20 пикселей, а завтра, может

быть, вы присвоите ему размер 24 пиксела или перейдете от пикселов к единицам em или процентам. Лучше уж воспользоваться селектором тега: назначьте размер шрифта элементу h2 или p или даже добавьте селектор потомков типа .sidebar1 p.

Исключение имен на основе положения элемента

По той же самой причине, по которой вы избегаете имен стилей согласно их настройкам форматирования, вы должны избегать их именования по расположению. Иногда такое имя, как .leftSidebar, кажется очевидным выбором: «Я хочу, чтобы весь материал в этом блоке был размещен у левого края страницы!» Но возможно, что кто-то захочет переместить левую боковую панель вправо, вверх или даже вниз страницы. И внезапно имя .leftSidebar перестанет иметь какой-либо смысл. Имена, соответствующие назначению этой боковой панели, — .news, .events, .secondaryContent, .mainNav — идентифицируют ее независимо от места расположения. Имена, которые вы видели до сих пор в этой книге, — .gallery, .figure, .banner, .wrapper и т. д. — подчиняются этому правилу.

Часто есть соблазн использовать имена типа .header и .footer (для колонтитулов, которые всегда находятся в верхней или нижней части страницы), поскольку их названия более чем очевидны. Но вы во многих случаях сможете подобрать имена, которые лучше определяют содержимое элементов, например .branding вместо .header. С другой стороны, использование имен с информацией о позиции иногда имеет смысл. Скажем, вы хотите создать два стиля: один для перемещения изображения в левую часть страницы, а другой — для перемещения изображения в правую часть. Поскольку эти стили существуют исключительно для размещения изображений в левой или правой части страницы, использование этой информации в имени стиля вполне оправданно. Так, .floatLeft и .floatRight — разумные имена.

Исключение невнятных имен

Такие имена, как .s, .s1 и .s2, могут уменьшить размер ваших файлов, но при этом доставят много хлопот при обновлении сайта. В итоге вам придется «поломать голову», чтобы вспомнить, для чего же все-таки были созданы эти загадочные стили. Будьте лаконичными, но понятными: имена .sidebar, .copyright и .banner не потребуют много времени для набора, но их назначение сразу очевидно.

ПРИМЕЧАНИЕ

Вы также можете выполнить поиск соглашений об именовании, используемых на других сайтах. Панели инструментов веб-разработчика, встроенные во многие браузеры, позволяют быстро подобрать имена стилям.

Исключение повторов

При вводе одного и того же кода снова и снова тратится ваше время, а также к стилям добавляется дополнительный код, который замедляет загрузку страниц для посетителей сайта. Когда вы создадите несколько страниц с элементами, которые очень похожи между собой, но обладают небольшими визуальными различиями, вы, вероятно, обнаружите, что несколько раз набрали одинаковые свойства каскадных таблиц стилей.

Например, у вас есть три различных типа кнопок — оранжевая кнопка для добавления нового элемента в корзину, красная кнопка для удаления элемента из корзины покупок и зеленая кнопка для оформления заказа. Каждой из них вы можете добавить различные классы в HTML-коде, например:

```
<button class="add">В корзину</button>  
<button class="delete">Удалить</button>  
<button class="order">Оформить заказ</button>
```

Затем создать три стиля:

```
.add {  
  border-radius: 3px;  
  font: 12px Arial, Helvetica, sans-serif;  
  color: #444;  
  background-color: orange;  
}  
.delete {  
  border-radius: 3px;  
  font: 12px Arial, Helvetica, sans-serif;  
  color: #444;  
  background-color: red;  
}  
.order {  
  border-radius: 3px;  
  font: 12px Arial, Helvetica, sans-serif;  
  color: #444;  
  background-color: green;  
}
```

Обратите внимание, что большинство из используемых в данном коде стилей одинаковы. Помимо того что дублирование кода увеличивает размер файла, если вы решите изменить шрифт текста на кнопках, вам потребуется изменить три стиля вместо одного.

Лучше создать «базовый» стиль, который будет распространяться на все кнопки, а также индивидуальные стили, которые будут содержать уникальное форматирование для кнопок. Например, если вы используете HTML-элемент `button`, можно создать стиль для него, который будет применяться для всех трех кнопок. Затем создайте еще три стиля, по одному для каждой кнопки:

```
button {  
  border-radius: 3px;  
  font: 12px Arial, Helvetica, sans-serif;  
  color: #444;  
}  
.add {  
  background-color: orange;  
}  
.delete {  
  background-color: red;  
}
```



```
.order {
  background-color: green;
}
```

Этот код не только менее объемный, но и более управляемый. Если вы хотите изменить шрифт текста на всех кнопках, просто внесите изменения в стиль `button`.

Несколько классов для экономии времени

Вы можете следовать девизу «Не повторяйтесь», используя несколько классов. Возможно, вы захотите, чтобы некоторые изображения были смещены влево и имели поле справа, в то время как другие должны сместиться вправо и содержать поле слева. Кроме того, вам захочется, чтобы границы по периметру этих изображений были выполнены в одном стиле; однако вы не хотите, чтобы все изображения на странице имели границу (рис. 18.2).

Самое очевидное решение — создать два класса, чтобы каждый имел одни и те же настройки границы, но различные свойства выравнивания и полей. Затем вы применяете один класс для изображений, которые должны быть смещены влево, а другой — для изображений, которые должны быть смещены вправо. Но как поступить, если вы должны обновить стиль границы для всех этих изображений? Вам понадобится отредактировать оба стиля, и, если вы забудете про один из них, у всех изображений на одной стороне страницы будет неправильное форматирование!

Существует прием, который работает во всех браузерах и преимуществами которого пользуется удивительно мало разработчиков, — применение к одному элементу *нескольких классов*. Это означает, что, применяя атрибут `class` к элементу, вы добавляете два (или больше) имени класса, например, так: `<div class = "note alert">`. В этом примере элемент `div` получает форматирование от стилей `.note` и `.alert`.

Скажем, вы хотите использовать один и тот же стиль границы для группы изображений, но одну их часть желаете поместить слева, а другую — справа. Подойдите к решению этой задачи следующим образом.

1. Создайте класс, который содержит настройки форматирования для всех изображений.

Этот стиль можно назвать `.imgFrame`, и он устанавливает сплошную черную границу шириной 2 пиксела со всех четырех сторон.

2. Создайте два дополнительных класса, один для изображений, смещаемых влево, а другой — для смещаемых вправо.

Например, `.floatLeft` и `.floatRight`. Один стиль будет содержать свойства, уникальные для одного набора изображений (сместить элемент влево и добавлять небольшое поле справа), в то время как другой стиль — свойства для второй группы изображений.

3. Примените оба класса к каждому элементу:

```

```

и

```

```

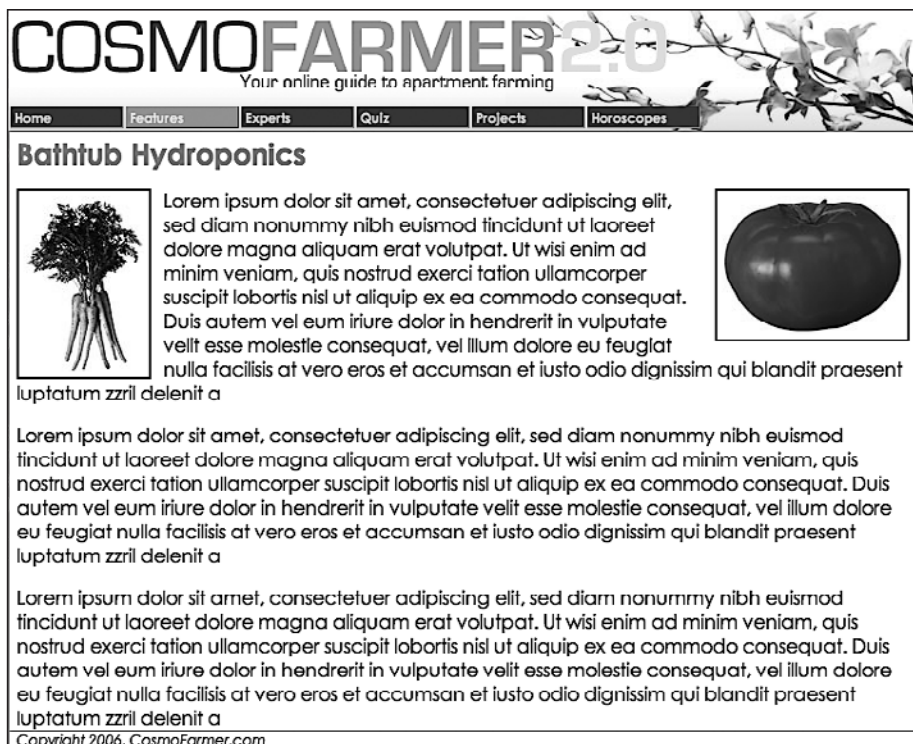


Рис. 18.2. Две фотографии, показанные на этом рисунке, имеют один и тот же примененный к ним класс. Его стиль предоставляет границу по периметру изображения. Кроме того, изображению слева присвоен класс, который смещает изображение влево, а правому изображению — другой класс, смещающий его вправо. То есть у каждого изображения есть два примененных к ним класса

На данный момент классы применяются к каждому элементу и браузер комбинирует стили каждого класса, чтобы отформатировать элемент. Теперь, если вы хотите изменить вид границы, отредактируйте стиль `.imgFrame`. Это позволит обновить границы по периметру изображений, смещенных как влево, так и вправо.

СОВЕТ

Вы можете перечислить этим методом более двух классов; только не забывайте ставить пробел между их именами.

Эта методика полезна, если вы планируете скорректировать несколько свойств одного элемента, оставив при этом остальные элементы, отформатированные подобным образом, неизменными. Вы можете разработать базовое форматирование боковой панели, которое сместит ее вправо, добавит фоновые изображения и будет включать тщательно разработанное оформление. Вы можете использовать этот стиль повсюду в вашем сайте, но ширина боковой панели будет иной в различных случаях. Возможно, она равна 33 % на одних страницах и 25 % — на других. В таком случае создайте отдельный класс (такой как `.sidebar`) с базовым форматированием

для боковой панели и отдельные классы для настройки исключительно ширины боковой панели, например `.w33per` и `.w 25per`. Затем примените по два класса к каждой боковой панели: `<div class="sidebar w33per">`.

Сокращенные записи

Многие свойства каскадных таблиц стилей могут объединяться в сокращенных записях, которые требуют меньше кода и соответственно меньше времени для его ввода. Например, свойство `padding` объединяет в себе четыре свойства: `padding-top`, `padding-right`, `padding-bottom` и `padding-left`. Таким образом, вы можете заменить следующий код:

```
td {
  padding-top: 5px;
  padding-right: 10px;
  padding-bottom: 5px;
  padding-left: 10px;
}
```

кодом:

```
td {
  padding: 5px 10px;
}
```

Существуют сокращенные записи для настройки шрифтов, границ, отступов, полей, переходов, фона и списков.

СОВЕТ

Если для значения какого-либо свойства используется величина измерения, например `30px`, `40%`, `10em`, вы можете отбросить единицу измерения, если значение равно 0. Другими словами, необходимо добавить запись вида:

```
padding: 0;
```

а не:

```
padding: 0px
```

Группировка связанных стилей

Добавление одного стиля за другим — распространенный способ создания таблицы стилей. Однако через некоторое время то, что было простой коллекцией из пяти стилей, расширяется до массивного CSS-файла с 500 стилями. В таком случае быстрый поиск нужного стиля — то же самое, что поиск иголки в стоге сена. Если вы организуете стили с самого начала, то в конечном счете намного облегчите себе жизнь. Не существует конкретных правил по поводу группировки стилей, но вы можете использовать две общепринятые методики.

- **Группируйте стили, которые применяются к определенным частям страницы.** Группируйте все правила, которые применяются к тексту, изображениям и ссылкам в баннере страницы, в одном месте, правила, которые относятся к навигации, — в другом, а стили для основного контента — в третьем.

- **Группируйте стили со связанными задачами.** Помещайте все стили для компоновки макета в одну группу, для форматирования текста — в другую, для изменения внешнего вида ссылок — в третью и т. д.

Использование комментариев для разделения групп стилей

Неважно, какой подход вы предпочтете, но убедитесь, что используете комментарии в CSS-коде, чтобы отделить каждую группу стилей. Скажем, вы собрали все стили, которые управляют компоновкой макета страниц, в одно место в таблице стилей. Опишите эту коллекцию таким комментарием:

```
/* *** Разметка *** */
```

или

```
/* -----  
Разметка  
----- */
```

Укажите в начале `/*`, а в конце — `*/`; тогда внутри вы можете использовать любую вычурную комбинацию из звездочек, черточек или других символов, которые вам нравятся, чтобы сделать эти комментарии легко узнаваемыми. Вы найдете столько их разновидностей, сколько существует веб-дизайнеров.

Дополнительные таблицы стилей

Из главы 17 вы узнали, что можно создавать различные таблицы стилей для разных типов устройств — одну для отображения страницы на экране, а другую — для вывода на печать. Кроме того, возможно, вы захотите использовать несколько таблиц стилей для отображения страницы на экране (исключительно в организационных целях). Здесь применяется базовая концепция из предыдущего раздела — группирование связанных стилей. Когда таблица стилей становится настолько большой, что трудно находить и редактировать стили, возможно, пришло время создать отдельные таблицы стилей, каждая из которых имеет свои функции. Вы можете поместить стили для форматирования веб-форм в одну таблицу стилей; стили для компоновки макета — в другую; а стили, которые определяют цвета элементов, — в третью. Конечно, количество отдельных файлов должно быть в пределах разумного, так как, скажем, 30 внешних CSS-файлов, которые придется поддерживать, совсем не сэкономят время. Кроме того, чем больше внешних CSS-файлов используется, тем на большее количество запросов должен отвечать ваш веб-сервер, что становится одной из причин снижения скорости работы сайта.

На первый взгляд может показаться, что получится больше кода в каждой веб-странице, так как будет намного больше внешних таблиц стилей, которые следует присоединить или импортировать, — по одной строке кода для каждого файла. Но есть подход лучше: создайте отдельную внешнюю таблицу стилей, которая содержит правила `@import`, импортирующие разные таблицы стилей. Рисунок 18.3 иллюстрирует этот подход.

Рассмотрим, как его реализовать.

1. Создайте внешние таблицы стилей для форматирования различных типов элементов вашего сайта.

Например, создайте файл `color.css` со стилями для изменения цвета сайта, файл `forms.css`, который управляет внешним видом веб-форм, файл `layout.css` — для компоновки макета и файл `main.css`, который охватывает все остальное форматирование (см. рис. 18.3, *справа*).

2. Создайте еще одну внешнюю таблицу стилей и импортируйте каждую таблицу стилей, которую вы создали на шаге 1.

Вы можете присвоить файлу имя `base.css`, `global.css`, `site.css` или какое-либо еще. Этот CSS-файл не будет содержать каких-либо стилей. Вместо них используйте правило `@import`, чтобы присоединить другие таблицы стилей:

```
@import url(main.css);  
@import url(layout.css);  
@import url(color.css);  
@import url(forms.css);
```

Это весь код, который должен быть в файле, хотя вы можете еще добавить некоторые комментарии с номером версии, названием сайта и т. д., чтобы помочь идентифицировать файл.

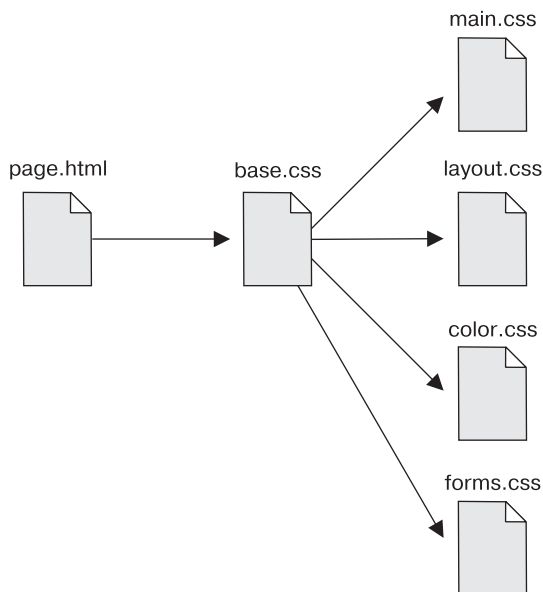


Рис. 18.3. HTML-страница может присоединить один CSS-файл (`base.css` в этом примере). HTML-код не должен изменяться, даже если вы хотите добавить или удалить дополнительные внешние таблицы стилей. Просто обновите файл `base.css`, добавляя или удаляя правило `@import`

3. Наконец, присоедините таблицу стилей из шага 2 к HTML-страницам вашего сайта, используя элемент `link` или правило `@import` (чтобы вспомнить, как

использовать эти методы, откройте раздел «Внешние таблицы стилей» главы 2). Например:

```
<link rel="stylesheet" href="base.css">
```

Теперь, когда веб-страница загружается, браузер использует файл `base.css`, который, в свою очередь, инструктирует браузер загрузить четыре остальные таблицы стилей.

Вам может показаться, что здесь происходит очень много загрузок. Однако браузер лишь однократно загружает эти файлы и сохраняет их в своем кэше — ему не приходится снова тратить на них интернет-трафик (см. врезку далее).

Есть и другая польза от применения отдельной внешней таблицы стилей, импортирующей несколько других CSS-файлов: если вы позже решите разделить стили на дополнительные таблицы стилей, то вам не придется изменять HTML-код страниц вашего сайта. Вместо этого добавьте еще одно правило `@import` к таблице стилей, аккумулирующей ссылки на другие CSS-файлы (см. шаг 2 выше). Если вы решите вырезать часть стилей из файла `main.css` и поместить их в новый файл `type.css`, вам не нужно будет затрагивать код веб-страниц сайта. Просто откройте таблицу стилей с правилами `@import` и добавьте еще одно: `@import url(type.css)`.

Эта возможность также позволяет обмениваться данными между различными таблицами стилей для временных изменений дизайна. Скажем, вы задумали изменять цвет вашего сайта каждый день, месяц или сезон. Если вы уже поместили основные, устанавливающие цвет стили в отдельный файл `color.css`, то можете создать другой файл (например, `summer_fun.css`) с иным набором цветов. Затем измените в таблице стилей, аккумулирующей ссылки на другие CSS-файлы, правило `@import` для файла `color.css`, чтобы загрузить новый файл стилей цвета (например, `@import url(summer_fun.css)`).

ПРИМЕЧАНИЕ

Препроцессоры каскадных таблиц стилей позволяют вам использовать все описанные преимущества сразу: редактировать несколько CSS-файлов и преобразовать их в одну таблицу стилей для вашего сайта, уменьшая время загрузки. Вы узнаете об этих передовых инструментах в следующей главе.

Устранение конфликтов стилей в браузере

Когда вы просматриваете в браузере веб-страницу, не использующую каскадные таблицы стилей, у HTML-элементов уже есть некоторое базовое форматирование: заголовки — полужирные, шрифт элемента `h1` крупнее остального текста, ссылки подчеркнуты и окрашены в синий цвет и т. д. В отдельных случаях разные браузеры применяют различное форматирование для каждого из этих элементов. Вас может разочаровать данный факт, но определенные элементы выглядят *почти* одинаково в Internet Explorer, Firefox и Chrome.

Из-за этих различий браузеров приходится сбрасывать исходное форматирование элементов, чтобы ваши посетители могли увидеть красивый дизайн, над созданием которого вы столь усердно работали (рис. 18.4). Все, что нужно сделать, — настроить в начале своей таблицы стилей основные стили, которые убирают ненужное форматирование.

Ниже приведены действия, которые можно выполнить, чтобы браузеры прекратили вмешиваться в ваш дизайн.

- **Удалите отступы и поля.** Браузеры добавляют поля сверху и снизу большинства блочных элементов — это, к примеру, те самые промежутки, которые появляются между элементами `p`. Это может вызвать некоторые проблемы при отображении контента, к примеру, когда конкретные размеры поля несовместимы с некоторыми браузерами. Лучше всего удалить все отступы и поля из блочных элементов, которые вы используете, а затем специально добавить нужные при создании новых стилей.
- **Применяйте единообразные размеры шрифта.** В то время как текст элемента `p` отображается размером `1em`, браузеры используют различные размеры для других элементов. Вы можете сделать так, чтобы для всех элементов был установлен размер шрифта `1em`, а затем создать дополнительные стили со специфическими размерами шрифта для нужных элементов. Таким образом, у вас будет намного больше шансов получить в результате одинаковый вид текста в различных браузерах.
- **Установите единообразную высоту строк.** Браузеры по умолчанию могут с незначительным различием отображать высоту строк текста. Применяв коэффициент к элементу `body` — `body { line-height: 1.2; }`, — можно обеспечить отображение браузерами строк с одинаковой высотой. Значение `1.2` эквивалентно `120%` размера шрифта текстовых элементов. Можно, разумеется, изменить это значение в соответствии с вашими дизайнерскими предпочтениями.
- **Улучшайте границы таблиц и создавайте согласующиеся ячейки.** Применяя границы к ячейкам таблицы, вы создаете промежуток между ячейками, а также удваиваете границы между ними. Вы должны избавиться как от лишнего пространства, так и от дополнительных границ. Кроме того, элементам `th` и `td` задаются разные типы выравнивания и плотность шрифта.
- **Удалите границы из изображений со ссылками.** Некоторые браузеры добавляют окрашенные границы по контуру любого изображения-ссылки. Скорее всего, вам, как и многим пользователям, эти границы покажутся ненужными и непривлекательными. Удалите их и задайте заново там, где считаете необходимым.
- **Задавайте согласованные отступы и маркеры для списков.** Различные браузеры используют разные отступы для маркированных и нумерованных списков, так же как и сами маркеры могут варьироваться. Желательно устанавливать согласованные отступы и типы маркеров.
- **Удалите кавычки из цитируемого контента.** Если вы когда-либо использовали элемент `q` для форматирования цитаты (например: `<q>Человеку свойственно ошибаться</q>`), то заметили, что некоторые браузеры (Firefox, Safari) автоматически добавляют кавычки (' ') вокруг цитаты, однако другие (Internet Explorer 6 и 7) — нет. И, кроме того, вид этих кавычек может быть разным. Например, Internet Explorer 8 вставляет одинарные кавычки (' '), в то время как Firefox — двойные (" "). Для согласованного представления контента лучше всего удалить кавычки.

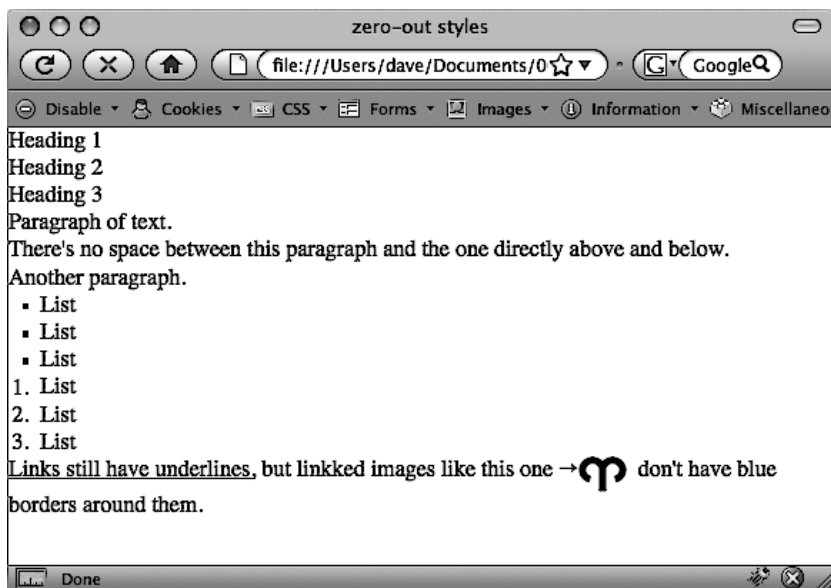


Рис. 18.4. Устраните различия в отображении страниц, сбросив исходные стили браузера. Затем создайте свои собственные стили, чтобы добавить поля, отступы и размеры шрифта, которые выглядят одинаково в разных браузерах

Для реализации описанных идей есть несколько базовых стилей, которые вы можете добавить в начало вашей таблицы стилей:

```
/* сброс стилей браузера */
```

```
* { box-sizing: border-box; }
```

```
html, body, div, span, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li,
fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed, figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary, time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    vertical-align: baseline;
}
```

```
article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav,
section {
    display: block;
}
```

```
body {
```



```
    line-height: 1.2;
  }

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

ol {
  padding-left: 1.4em;
  list-style: decimal;
}

ul {
  padding-left: 1.4em;
  list-style: square;
}

blockquote, q {
  quotes: none;
}

blockquote:before, blockquote:after, q:before, q:after {
  content: '';
  content: none;
}

/* конец кода сброса стилей браузера */
```

Первый стиль изменяет то, как браузеры трактуют свойство `width`. Он гарантирует, что любые значения ширины, которые вы будете устанавливать, будут также включать границы и отступы — вы можете прочитать о свойстве `box-sizing` в главе 7.

Следующие два стиля в коде — групповые селекторы, которые применяют одно и то же форматирование к каждому из перечисленных элементов. Добавьте эти стили в начало таблицы стилей, а затем в нижней части таблицы замените их по необходимости. После сброса полей и размера шрифта для элемента `h1` вы можете задать для него определенное значение верхнего поля и размер шрифта. Просто добавьте другой стиль, например такой:

```
h1 {
  margin-top: 5px;
  font-size: 2.5em;
}
```

Этот стиль будет применен *после* группового селектора, удаляющего поля и сбрасывающего размер шрифта, поэтому благодаря каскадности (см. главу 5) новые значения будут иметь приоритет.

Файл `reset.css` вы найдете в папке 18 примеров, прилагаемых к книге. Просто скопируйте код из этого файла в свои таблицы стилей.

ПРИМЕЧАНИЕ

Некоторые разработчики используют для решения проблемы различий отображения браузерами веб-контента другой подход. Проект `Normalize.css` (tinypurl.com/cor5s89) предназначен для применения согласованных базовых стилей с сохранением при этом основных различий в форматировании HTML-элементов. Например, вместо приведения к единому размеру шрифта текста заголовков и абзацев `Normalize.css` учитывает различия в уровнях заголовков. В проект также включено множество других стилей, предназначенных для устранения недостатков, присущих некоторым браузерам.

Использование селекторов потомков

Классы и идентификаторы отлично подходят для обозначения конкретных элементов, которые необходимо форматировать. Например, вы можете добавить класс к абзацу с помощью кода `<p class="intro">` и указать, что только один абзац будет иметь внешний вид, определенный стилями класса `.intro`. Поскольку добавить класс или идентификатор к элементу не составляет никакого труда, многие разработчики взяли моду добавлять их ко *всем* элементам (или почти ко всем), что не может не вызывать беспокойства. У профессионалов есть даже диагноз этого заболевания — *тяга к классификации*. Добавление класса к каждому элементу — это не только лишняя трата времени, но и замедление загрузки HTML-файлов. К тому же есть лучший способ управления элементами, не прибегая к слишком большому количеству классов или идентификаторов, — использование селекторов потомков.

ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Головная боль с кэшем

Кэш браузера, как правило, является помощником каждого владельца сайта. Как мы уже обсуждали в этой книге, кэш гарантирует, что частым посетителям вашего сайта не придется загружать один и тот же файл снова и снова, что замедлило бы открытие страниц и повысило плату за хостинг. Тем не менее кэш может стать головной болью, когда вам необходимо обновить внешний вид сайта. Например, если все его страницы ссылаются на внешнюю таблицу стилей с именем `main.css`, у посетителей этот файл будет кэширован. Но при обновлении файла и, соответственно, изменении внешнего вида сайта предыдущие посетители могут по-прежнему иметь доступ к старой таблице стилей, сохраненной на их жестком диске, вместо нового файла `main.css`.

Со временем кэш посетителей все же очистится и они получат новые CSS-файлы. Однако у вас есть один простой способ победить кэш — обновить элемент `link` на каждой HTML-странице. Обычно элемент `link` для внешней таблицы стилей выглядит следующим образом:

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

Однако если добавить строку запроса после имени CSS-файла (например, `main.css?v=1`), то браузер будет видеть файл как `main.css?v=1`, а не как `main.css`. Если вы измените число после символов `v=` при обновлении внешней таблицы стилей, то браузеры воспримут этот код как появление нового файла и загрузят внешнюю таблицу стилей с веб-сервера вместо использования кэшированной версии.

Предположим, что, когда вы впервые публикуете ваш сайт, файл `main.css` является первой версией каскадной таблицы стилей сайта. Вы можете использовать следующую ссылку:

```
<link rel="stylesheet" href="main.css?v=1">
```

Затем, когда вы обновите файл `main.css`, можно изменить код элемента `link` следующим образом:

```
<link rel="stylesheet" type="textcss" href="main.css?v=2">
```

Браузер определит, что таблица стилей отлична от сохраненной в кэше версии файла `main.css` и загрузит файл с веб-сервера.

На самом деле код `?v=1` ничего не делает и не влияет на то, как работает веб-сервер. Это лишь способ сооб-

щить браузеру о необходимости повторной загрузки файла.

Недостатком этого метода является необходимость обновлять код элемента `link` для каждого HTML-файла сайта.

Селекторы потомков — мощный инструмент для эффективного создания сайтов. Как мы обсуждали в главе 3, они позволяют точно определить элементы, которые нужно форматировать. В большинстве случаев требуется одинаково оформить *все* ссылки навигационной панели, но это не значит, что нужно отформатировать все прочие ссылки на *странице* таким же образом. Нужно как бы сообщить браузерам (с помощью CSS-кода): «Отформатируйте с помощью этих стилей *только* ссылки на навигационной панели», не применяя класс к каждой из этих ссылок. Другими словами, нам нужна возможность форматирования одинаковых HTML-элементов по-разному, в зависимости от того, где они располагаются, а это как раз то, что предоставляют селекторы потомков (рис. 18.5).

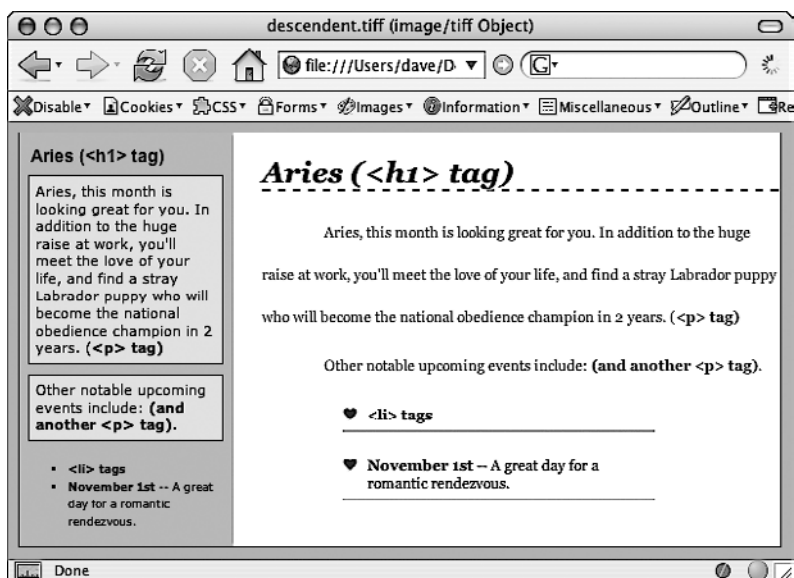


Рис. 18.5. Один и тот же HTML-код был вставлен в левую боковую панель и в большую область справа. При использовании селекторов потомков идентичные HTML-элементы (`h1`, `p`, `ul` и `li`) форматированы по-разному, в зависимости от их расположения на странице

Разделение контента

Одним из самых больших подспорий в эффективном использовании селекторов потомков является использование элемента `div`. Поскольку этот элемент позволяет создавать логические *разделы* на странице, вы можете применять его для идентификации

различных типов контента, таких как баннер, боковая панель, колонка текста и т. д. Содержимое страницы можно организовать в блоки, назначая каждому из них HTML-элемент `div`.

Сгруппируйте заголовок текста и список ссылок для навигации по страницам:

```
<div>
  <h2>Мой сайт</h2>
  <ul>
    <li><a href="page1.html">страница 1</a></li>
    <li><a href="page2.html">страница 2</a></li>
    <li><a href="page3.html">страница 3</a></li>
  </ul>
</div>
```

После добавления контейнера `div` обозначьте его для каскадной таблицы стилей с помощью класса (`<div class = "pullQuote">`). Если вы хотите вставить один и тот же тип контента несколько раз на странице (например, несколько врезок в отдельной истории), используйте класс.

Предположим, что список ссылок в HTML-коде, приведенном выше, появляется дважды на странице: в начале и в конце. Вы добавляете к нему класс следующим образом:

```
<div class="storyNav">
  <h2>Мой сайт</h2>
  <ul>
    <li><a href="page1.html">страница 1</a></li>
    <li><a href="page2.html">страница 2</a></li>
    <li><a href="page3.html">страница 3</a></li>
  </ul>
</div>
```

СОВЕТ

Вам не всегда нужно добавлять элемент `div`, чтобы отформатировать группу элементов. Если бы код, приведенный выше, содержал только неупорядоченный список ссылок и не включал элемент `h2`, то вы могли бы легко пропустить элемент `<div>` и добавить класс к неупорядоченному списку следующим образом: `<ul class = "storyNav">`. Можно также заключить элемент `ul` в HTML5-контейнер `nav` и применить класс к этому контейнеру.

Как только вы идентифицируете каждый элемент `div` на странице, будет очень легко использовать селектор потомков, нацеленный на элементы внутри конкретного контейнера `div`. Скажем, вы хотите создать уникальный вид для каждой ссылки в приведенном коде. Вы создаете селектор потомков следующим образом:

```
.storyNav a {
  color: red;
  background-color: #ccc;
}
```

Теперь ссылки будут выделены красным цветом на светло-сером фоне, но *только* когда они находятся *внутри* другого элемента, к которому применяется класс `storyNav`. Если вы хотите добавить другую ссылку (например, на страницу `page4.html`) в этот список, вам не нужно обращаться к HTML-коду, чтобы отформатиро-

вать ее, как другие ссылки. Браузер все обрабатывает автоматически, когда применяет селектор потомков.

Форматирование других элементов внутри этого контейнера `div` — лишь вопрос создания селектора потомков, начинающегося с имени класса (`.storyNav`), за которым следует пробел и имя элемента, стиль которого вы хотите создать. Чтобы отформатировать элемент `h2`, который находится внутри контейнера `div`, создайте селектор потомков `.storyNav h2`.

Идентификация тела страницы

Поскольку селекторы потомков обеспечивают такое специфическое определение стилей, вы можете легко создавать стили, которые не только относятся к конкретной области страницы, но и применяются исключительно к определенным *типам* страниц на вашем сайте. Скажем, вы хотите отформатировать заголовок `h1` на главной странице не так, как для остальных страниц сайта. Простой способ выделения элементов `h1` на главной странице — присвоить класс к элементу `body` на ней следующим образом:

```
<body class="home">
```

Вы можете отформатировать элемент `h1` на главной странице, используя селектор потомков: `.home h1`. Таким же образом можно создавать форматирование любому элементу конкретной страницы вашего сайта (рис. 18.6). Один из подходов — идентифицировать раздел сайта, в котором находится каждая страница. Скажем, ваш сайт разделен на четыре раздела: новости, события, публикации и ссылки. На каждой странице внутри раздела назначьте класс или идентификатор элементу `body`. Так, каждая страница в разделе новостей могла бы содержать следующий HTML-код: `<body class="news">`, в то время как для страниц с событиями был бы использован код `<body class="events">`.

СОВЕТ

Вы также можете использовать класс, чтобы идентифицировать тип дизайна, который хотите установить для определенной страницы (например, с одной, двумя или тремя колонками).

Кроме всего прочего, идентификация раздела страницы применяется для выделения кнопки этого раздела на навигационной панели. Выделенные кнопки действуют наподобие индикаторов, указывающих, на какой странице находится пользователь, как продемонстрировано на рис. 18.6. Если страница находится в разделе новостей вашего сайта, вы можете выделить кнопку **Новости**, так что посетитель моментально сможет определить раздел, который просматривает.

Рассмотрим, как можно отформатировать навигационную кнопку в зависимости от того, в каком разделе сайта она находится.

1. Присвойте класс элементу `body` для обозначения раздела сайта, в котором находится страница.

Например, код может выглядеть следующим образом: `<body class = "home">`. Повторите то же самое для каждого раздела, чтобы у страниц в разделе новостей был следующий код: `<body class = "news">`.

- Добавьте навигационную панель на страницу.

Пошаговые инструкции по созданию навигационной панели вы найдете в разделе «Использование ролловеров» главы 9.

- Обозначьте каждую ссылку навигационной панели.

Для ссылки на главную страницу у вас может использоваться следующий код: `Главная`. Атрибут `class` позволяет обозначить ссылку как указывающую на главную страницу. Повторите это для других ссылок: `Новости` и т. д.

На данный момент в HTML-коде предоставлено достаточно информации, чтобы уникальным образом отформатировать ссылку каждого раздела, используя каскадные таблицы стилей. В этом примере ссылка на главную страницу вложена в элемент `body` с классом `home` *только* на главной странице.

- Создайте селектор потомков, который позволит отформатировать ссылку каждого раздела по отдельности. Это форматирование применяется, когда ссылка находится на странице соответствующего раздела.

Для главной страницы в этом примере селектор потомков должен выглядеть примерно так:

```
.home .homeLink
```

Селектор форматирует ссылку `.homeLink`, только если она находится внутри другого элемента с классом `.home`. В большинстве случаев нужно, чтобы выделенная кнопка выглядела одинаково для каждого раздела сайта, так что лучше использовать групповой селектор, чтобы группировать все селекторы потомков для каждой кнопки раздела. Таким образом, вы применяете одно и то же форматирование к каждой кнопке, не создавая для нее отдельные правила. Групповой селектор для выделения кнопки текущего раздела светло-желтым фоном может быть таким:

```
.home .homeLink,
.news .newsLink,
.articles .articlesLink,
.links .linksLink {
  background-color: #FBEF99;
}
```

СОВЕТ

При создании группового селектора, который включает в себя несколько селекторов потомков, указывайте каждый селектор на отдельной строке, как в этом примере. Так будет легче распознать селекторы в группе, если придется редактировать таблицу стилей.

Используя схожую методику, создайте дополнительные стили, определяющие различное форматирование для всех состояний ссылок: при наведении указателя мыши, щелчке кнопкой мыши и пр. (см. главу 9).

Приведенные примеры — лишь некоторые способы использования селекторов потомков. Эти селекторы могут немного усложнить ваши таблицы стилей. У вас будут такие стили, как `.home .navbar`, вместо, к примеру, простого класса `.navLink`.



Рис. 18.6. Используя селекторы потомков, можно выделить кнопку на навигационной панели, просто изменив класс, присвоенный элементу `body`

Но, однократно настроив стили, в дальнейшем вы будете вносить лишь небольшие правки в код. И теперь HTML-код, который вставляется в различные области страницы, автоматически форматируется совершенно по-разному. Почти как по волшебству.

ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ**Объектно-ориентированный CSS-код**

Объектно-ориентированный CSS-код представляет еще один подход к организации и использованию CSS. Этот термин был придуман знатоком каскадных таблиц стилей, Николь Салливан, а подход был разработан в ответ на сложность поддержки крупных сайтов с достаточным разнообразием HTML-структур. На больших сайтах может быть очень разный HTML-код, содержащий сходные типы информации.

Например, на одной странице может использоваться маркированный список для отображения имен, контактной информации и фотографий списка контактов. А на другой — элемент `article` для каждой карточки контакта. Предположим, что нужно отформатировать эти элементы одинаково, поскольку это информация действительно одинакового типа. При использовании стилей, зависящих от этого HTML-кода, можно столкнуться либо с дублированием стилей, либо с групповыми селекторами наподобие следующих:

```
article img, li img {
  /* сюда помещается форматирование */
}
```

Объектно-ориентированный CSS рекомендует отказаться от привязки CSS-кода к реальным HTML-элементам, и использовать вместо них классы. То есть вы по всему HTML-коду применяете классы, а затем используете для форматирования стили классов.

При этом неважно, какие HTML-элементы используются. Если они применяют одни и те же классы, у них будет одинаковый внешний вид.

Возможно, это похоже на проблему «тяги к классификации», и по большому счету это так и есть. Объектно-ориентированный CSS-код требует добавления большого количества атрибутов `class` в HTML-код. Если используется такая система управления контентом, как WordPress, то объем работы может быть не столь существенным — имена классов можно добавлять к файлам шаблонов. Но если каждая страница создается вручную, такой подход может добавить большой объем работы.

Краткое введение в объектно-ориентированный CSS-код доступно по адресу tinyurl.com/qzo9jik. Проект OOCSS можно найти по адресу tinyurl.com/kfz84cn.

Еще один подобный подход, разработанный Джонатаном Снуком (Jonathon Snook), который называется *масштабируемой и модульной архитектурой для CSS* (SMACSS, Scalable and Modular Architecture for CSS), является простым руководством для создания повторно используемых CSS-компонентов. Дополнительные сведения об этом подходе можно получить по адресу smacss.com. По адресу tinyurl.com/pl38ake можно найти множество видеороликов, касающихся SMACSS.

19 Профессиональный дизайн с помощью Sass

К этому моменту вы узнали много нового о каскадных таблицах стилей. В предыдущей главе вы изучили методы улучшения таблиц стилей. А в этой главе мы рассмотрим очень популярный инструмент, который существенно упрощает написание CSS-кода.

Понятие Sass

Sass — это *препроцессор CSS*. Это означает, что сначала вы пишете Sass-код, который затем компилируется в CSS-код для браузера. Вы можете воспринимать метаязык Sass (расшифровывается как Syntactically Awesome Stylesheets — «синтаксически привлекательные таблицы стилей») как нечто вроде сокращенного варианта записи CSS-кода, который позволяет писать код меньшего объема.

Метаязык Sass работает с файлами двух типов: Sass-файлы, которые имеют расширение `.scss`, и CSS-файлы, имена которых (как вы знаете) оканчиваются символами `.css`. На самом деле вы редактируете только файлы `.scss`: препроцессор Sass автоматически преобразует Sass-файл в привычный файл `.css`.

Может показаться, что для того, чтобы создать CSS-код, о котором вы узнали в этой книге, необходимо проделать дополнительную работу. Однако, поскольку ваши сайты и их таблицы стилей станут больше и сложнее, Sass вносит ряд преимуществ.

- Он организует ваш CSS-код в файлы меньшего размера. В настоящее время веб-дизайнеры пытаются использовать по возможности меньше CSS-файлов. Чем больше файлов приходится загружать браузеру, тем дольше посетителю сайта нужно ждать передачи файлов от веб-сервера. Поэтому некоторые веб-дизайнеры добавляют сотни или даже тысячи стилей в одну таблицу стилей. Хотя это может быть более эффективным для посетителей сайта, но для вас попытка найти определенный стиль, который, к примеру, форматирует заголовок `h2` на боковой панели страницы **О нас**, будет достаточно сложной. Sass позволяет логически группировать все стили вашего сайта в файлы меньшего размера. К примеру, все стили веб-форм вашего сайта могут находиться в одном файле Sass, в то время как правила оформления шрифтов — в другом. Когда Sass

выполняет предпроцессорную обработку, он может автоматически объединить эти несколько файлов в один для быстрой загрузки CSS-файла. Что еще лучше, инструмент Sass может сжать конечный CSS-файл таким образом, что его размер будет намного меньше, чем у файла, созданного вручную.

- Значения свойств легко обновлять. В процессе работы вы заметите, что используете одни и те же значения в свойствах стилей снова и снова. Например, ваша компания может применять несколько определенных корпоративных цветов, которые вы постоянно используете в веб-дизайне для форматирования шрифта, фона, границ и т. д. А если ваш отдел маркетинга решит, что цвета нужно изменить? При использовании обычного CSS-файла вам понадобится найти и изменить *каждый* экземпляр измененного цвета. Благодаря Sass вы можете определить значение цвета в одном месте и использовать его во всех стилях. Измените цвет и обновите его в одном месте Sass-файла, и инструмент Sass автоматически заменит цвет на новый.
- Вам понадобится писать меньше кода. Sass содержит некоторые действительно мощные инструменты, которые позволяют писать меньше кода. Помните вендорные префиксы? Вам приходилось писать несколько строк кода только для того, чтобы использовать одно и то же свойство в различных браузерах. *Примеси* позволяют сделать эту нудную работу за вас. Вы можете написать только одну строку кода и указать Sass преобразовать ее для различных версий браузеров с необходимым вендорным префиксом. Кроме того, если вы используете один и тот же набор свойств CSS в различных стилях, можно указать Sass добавлять этот код автоматически. Меньшее время написания кода означает более быстрое время верстки и меньшую вероятность сделать ошибку.

У Sass есть много других преимуществ, о которых вы узнаете в этой главе, но суть в том, что данный метаязык повысит вашу эффективность в качестве веб-дизайнера: вы сможете работать быстрее и тратить больше времени на обдумывание дизайна вместо верстки кода.

ПРИМЕЧАНИЕ

Подробную информацию об инструменте Sass можно найти на официальном сайте sass-lang.com (рис. 19.1).

Однако у инструмента Sass есть несколько недостатков. Во-первых, для написания CSS-кода он вводит несколько новых концепций, а также новый синтаксис. Вы узнаете о них в этой главе. Во-вторых, чтобы использовать Sass, необходимо установить дополнительное программное обеспечение на ваш компьютер. Метаязык Sass основан на языке программирования Ruby, поэтому вам потребуется установить среду разработки Ruby, а также основные файлы командной строки Sass. Кроме того, Sass — не обычная программа, запускаемая двойным щелчком кнопкой мыши. Чтобы начать работать с Sass, необходимо использовать командную строку (приложение `cmd` в операционной системе Windows или Терминал (Terminal) — в OS X). Командная строка пугает многих веб-дизайнеров, но бояться ее не стоит. На самом деле все не так страшно, как вы и увидите в следующем разделе.

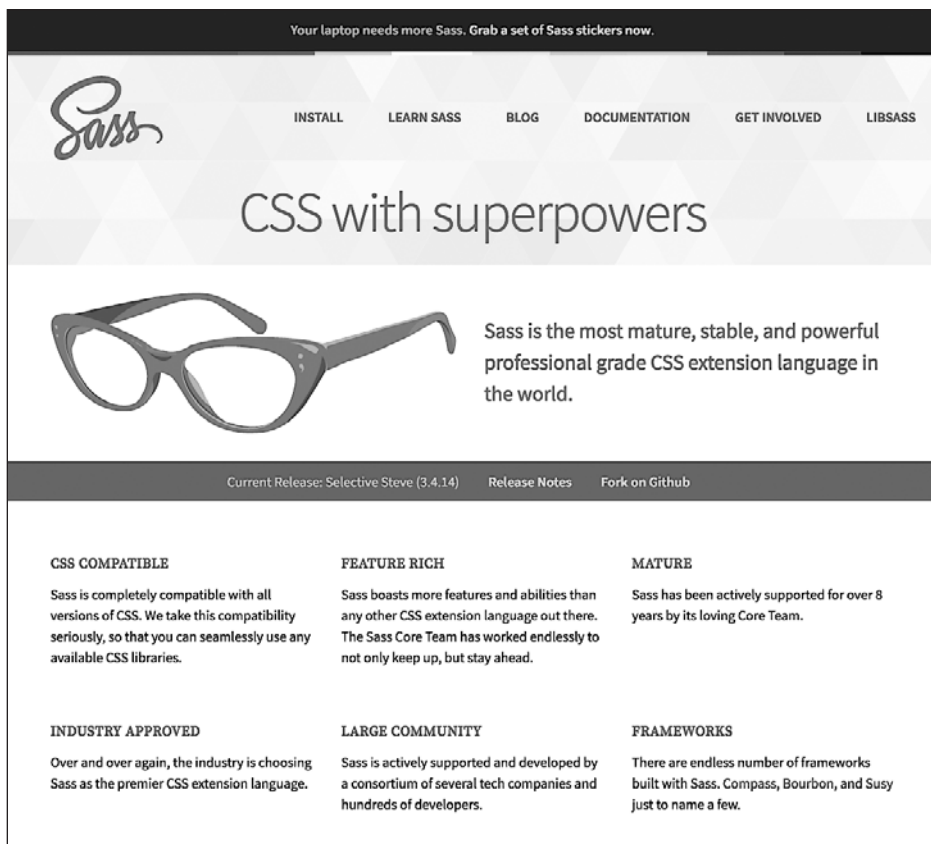


Рис. 19.1. Официальный сайт Sass — отличный способ узнать подробнее о преимуществах этого метаязыка

Интеграция Sass

Первоначально расширение Sass было создано с использованием языка программирования Ruby. Этот же язык применялся во фреймворке Ruby On Rails при создании многих сайтов, таких как GitHub, Square, Airbnb и Twitter. Перед установкой Sass необходимо установить среду разработки Ruby. В операционной системе Windows сначала понадобится скачать среду разработки Ruby, а затем установить ее. Операционная система OS X уже содержит среду разработки Ruby, поэтому ее установка занимает пару шагов (пользователи компьютеров Mac могут сразу перейти к подразделу «Установка Sass в операционной системе OS X»).

ПРИМЕЧАНИЕ

Оригинальная версия Sass работает только вместе со средой Ruby. Однако в настоящее время доступна версия libSass, которая использует языки программирования C/C++. Версия libSass поддерживает многие языки программирования и часто применяется с платформой Node.js в сочетании с JavaScript в качестве таких инструментов, как Grunt (gruntjs.com) и Gulp (gulpjs.com).

Установка Sass в операционной системе Windows

Установка Sass в операционной системе Windows требует выполнения двух простых шагов. Во-первых, вам необходимо установить среду разработки Ruby и программу Sass (написанную на языке программирования Ruby).

1. Загрузите установочный пакет Ruby с сайта rubyinstaller.org/downloads/.

Скачайте последнюю версию среды разработки Ruby, перейдя по одной из ссылок загрузки. Если на вашем компьютере установлена 64-разрядная версия операционной системы Windows, выберите 64-разрядную версию. Установщик представляет собой обычный исполняемый файл Windows с расширением .exe.

ПРИМЕЧАНИЕ

В книге описана версия установщика 2.2.2. Иногда более новая версия Ruby не поддерживает Sass. В этом случае установите более старую версию среды разработки Ruby.

2. Запустите установочный файл.

Появится окно мастера установки, в котором следует выбрать язык интерфейса мастера и принять условия лицензионного соглашения. Далее вы увидите окно выбора пути и компонентов установки (рис. 19.2).

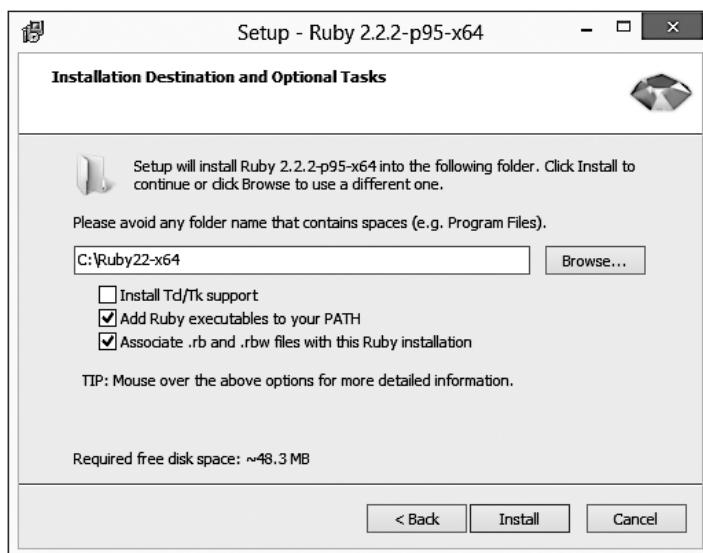


Рис. 19.2. Окно выбора пути и компонентов установки

3. В диалоговом окне Installation Destination and Optional Tasks (Путь установки и выбор компонентов) проверьте путь установки и установите флажки Add Ruby executables to your PATH (Добавить в системные пути исполняемые файлы Ruby) и Associate files with this Ruby installation (Ассоциировать файлы .rb и .rbw с Ruby). Важно установить первый флажок. В этом случае операционная система Windows «узнает», где установлена среда разработки Ruby. Затем, когда

вы будете использовать командную строку, операционная система Windows будет знать, где находятся необходимые файлы среды Ruby, позволяющие выполнить команду. Если вы не установите этот флажок, запустить Sass будет проблематично.

Среда разработки Ruby будет установлена на компьютер. Процесс займет пару минут.

4. После завершения установки нажмите кнопку **Finish** (Готово) в последнем диалоговом окне.

Теперь среда разработки Ruby установлена в операционной системе. Можно приступить к установке компонентов Sass с помощью приложения командной строки.

5. Запустите приложение командной строки.

Вы можете использовать приведенный ниже быстрый способ.

- Нажмите сочетание клавиш **Win+R**, чтобы открыть диалоговое окно **Выполнить** (Run). Это диалоговое окно позволяет запускать приложения путем ввода их имени.
- Введите `cmd` и нажмите кнопку **OK**. При этом запустится программа `cmd.exe` и откроется окно командной строки (рис. 19.3).

СОВЕТ

Чтобы узнать другие способы запуска приложения командной строки в операционной системе Windows, посетите сайт tinyurl.com/km2yp86.

Приложение командной строки позволяет вводить команды и запускать приложения непосредственно с помощью консоли. Это способ, который применяют некоторые пользователи компьютеров с 70-х годов прошлого века.

ПРИМЕЧАНИЕ

Если приложение командной строки было запущено в тот момент, когда вы устанавливали среду разработки Ruby, необходимо его перезапустить. В противном случае путь к файлам Ruby не будет обнаружен.

6. Введите команду `gem install sass` и нажмите клавишу **Enter**.

Команда `gem` относится к Ruby и используется для установки различных программ и библиотек. Процесс установки может занять несколько минут, системе потребуется загрузить и установить необходимые файлы.

Если на вашем компьютере установлено какое-либо антивирусное программное обеспечение, возможно, вам потребуется подтвердить установку программы Sass.

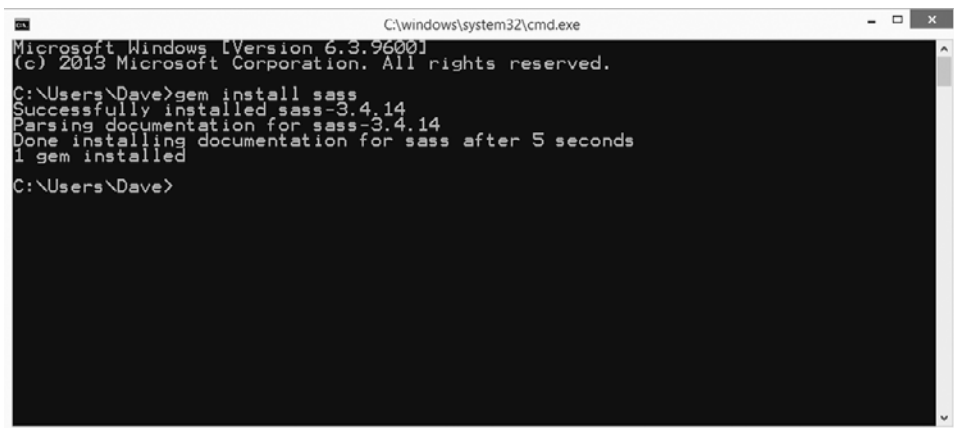
Кроме того, если вы не установили флажок **Add Ruby executables to your PATH** (Добавить в системные пути исполняемые файлы Ruby) при установке среды разработки Ruby (см. рис. 19.2), то получите сообщение об ошибке вида «`gem` is not recognized as an internal or external command, operable program or batch file». Оно означает, что операционная система Windows не может обнаружить файлы Ruby и выполнить команду `gem`. В этом случае переустановите пакет Ruby, ориентируясь

на указанные выше шаги, и убедитесь, что флажок **Add Ruby executables to your PATH** (Добавить в системные пути исполняемые файлы Ruby) в этот раз установлен.

Если вы все сделали правильно, командная строка должна выглядеть так, как показано на рис. 19.3. Вы готовы использовать Sass. О том, как это сделать, написано в следующем разделе.

ПРИМЕЧАНИЕ

Чтобы обновить компоненты Sass до последней версии, выполните команду `gem update sass` в командной строке.



```
C:\windows\system32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Dave>gem install sass
Successfully installed sass-3.4.14
Parsing documentation for sass-3.4.14
Done installing documentation for sass after 5 seconds
1 gem installed

C:\Users\Dave>
```

Рис. 19.3. Командная строка Windows выглядит не очень привлекательно, но ее использование — простой способ установки и запуска компонентов Sass. С ее помощью можно увидеть, что только что была установлена версия 3.4.14. Чтобы увидеть, какая версия установлена у вас, введите команду `sass -v`

Установка Sass в операционной системе OS X

Поскольку среда разработки Ruby предустановлена в операционной системе OS X, процесс установки компонентов Sass предельно прост.

1. Запустите приложение **Терминал** (Terminal).

Для этого перейдите в папку **Приложения/Утилиты** (Applications/Utilities). Окно приложения **Терминал** (Terminal) понадобится вам в дальнейшем, поэтому добавьте его значок на панель Dock, перетащив его мышью.

2. В окне приложения **Терминал** (Terminal) введите команду `gem install sass` (рис. 19.4).

Команда `gem` относится к Ruby и используется для установки различных программ и библиотек. Процесс установки может занять несколько минут, системе потребуется загрузить и установить необходимые файлы.

Кроме того, в зависимости от настроек прав доступа в вашей операционной системе OS X вам, возможно, потребуется выполнить эту команду от имени администратора. Для этого необходимо использовать команду `sudo`:

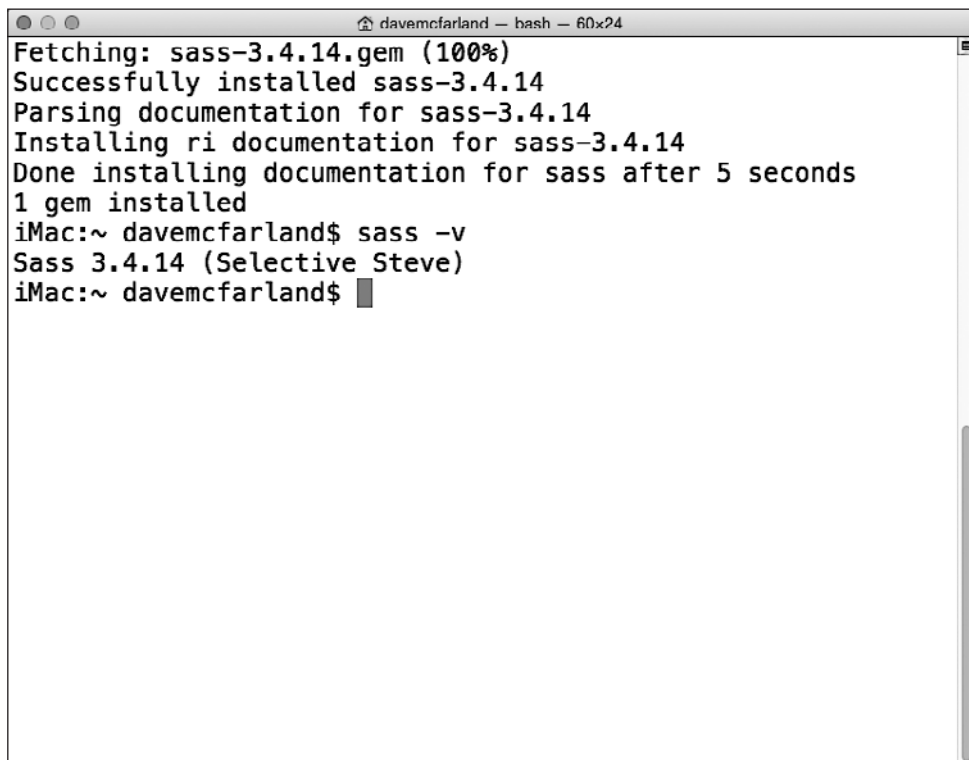
```
sudo gem install sass
```

Затем для установки компонентов Sass необходимо ввести имя и пароль администратора.

Если вы сделали все правильно, окно приложения Терминал (Terminal) будет выглядеть так, как показано на рис. 19.4. Вы готовы к использованию Sass. О том, как это сделать, написано в следующем разделе.

ПРИМЕЧАНИЕ

Чтобы обновить компоненты Sass до последней версии, выполните команду `gem update sass` в окне приложения Терминал (Terminal). Если у вас нет прав администратора, используйте команду `sudo gem update sass`, а затем введите имя и пароль администратора.



```
davemcfarland — bash — 60x24
Fetching: sass-3.4.14.gem (100%)
Successfully installed sass-3.4.14
Parsing documentation for sass-3.4.14
Installing ri documentation for sass-3.4.14
Done installing documentation for sass after 5 seconds
1 gem installed
iMac:~ davemcfarland$ sass -v
Sass 3.4.14 (Selective Steve)
iMac:~ davemcfarland$
```

Рис. 19.4. Программа Терминал (Terminal) — утилита командной строки. Вы будете использовать ее все время при работе с Sass и многими другими инструментами веб-разработки. Чтобы увидеть, какая версия установлена у вас, введите команду `sass -v`

Основы Sass

Sass — это одновременно и программный *инструмент*, который создает CSS-файлы, и *язык*, который добавляет функции, которые недоступны в настоящее время в языке CSS. Но не волнуйтесь, после изучения языка CSS вам не нужно тратить еще месяц на обучение новому языку. Sass полностью совместим с CSS. Другими словами, вы можете начать с обычной таблицы стилей (как те, что вы создавали на

протяжении всей книги) и постепенно улучшать ее, применяя экономящие время возможности языка Sass.

В этой главе вы узнаете, как добавить новые функции стилям. Но главное, что необходимо помнить, это то, что при создании стилей с помощью Sass, вы можете применять все свои с трудом накопленные знания о языке CSS. В самом деле вы можете взять обычную таблицу стилей, например `styles.css`, и превратить ее в таблицу стилей Sass просто путем замены расширения `.css` на `.scss` — `styles.scss`.

В КУРС ДЕЛА!

Sass и командная строка

Если одна только мысль о командной строке в операционной системе Windows или окне приложения Терминал (Terminal) в OS X погружает вас в панику, есть и другие способы использовать всю мощь языка Sass.

Отдельные текстовые редакторы, такие как Sublime и WebStorm, позволяют установить плагины, которые могут компилировать Sass в CSS-код. Кроме того, существует несколько бесплатных и платных программ, которые могут скомпилировать Sass-файлы в CSS без установки среды разработки Ruby и компонентов Sass.

Для этих приложений необходимо указать, в каких папках находятся Sass-файлы и в какую папку должны сохраняться готовые, скомпилированные CSS-файлы. Они могут также обнаруживать изменения, внесенные в Sass-файлы, и автоматически генерировать новые CSS-файлы, так что вы сможете мгновенно просматривать внесенные изменения. Некоторые программы даже автоматически обновляют окно браузера, чтобы перезагрузить обновленный CSS-файл.

- Scout (tinyurl.com/ksvxl79) — популярное бесплатное приложение, предназначенное как для операционной системы Windows, так и для OS X.
- Приложение Koala (koala-app.com) работает в операционных системах Windows, OS X и Linux. Оно не только компилирует Sass-файлы, но также может преобразовывать файлы Less (еще один препроцессорный язык CSS) и CoffeeScript (препроцессорный язык JavaScript).
- LiveReload (livereload.com) — платное решение для операционной системы OS X (\$10). Версия для операционной системы Windows доступна в виде альфа-версии и предлагается для тестирования. Программа позволяет компилировать код на языках Sass, Less, Stylus, CoffeeScript, Jade и многих других, используемых веб-разработчиками.
- CodeKit (tinyurl.com/3ff6bo) — мощное приложение для операционной системы OS X (\$32), которое может обрабатывать не только Sass-код и множество других языков, но и автоматически обновляет браузер, что позволяет мгновенно просматривать изменения, внесенные в Sass-файлы.

Организация файлов

При использовании Sass вы будете работать с двумя различными типами файлов: теми, которые вы создаете (Sass-файлы), и теми, которые сгенерирует препроцессор Sass (обычные CSS-файлы, которые использует браузер). Sass-файлы заканчиваются расширением `.scss`, в них вы будете добавлять CSS- и Sass-код.

Затем вы будете использовать программное обеспечение Sass, чтобы преобразовать Sass-файлы в CSS-файлы, заканчивающиеся расширением `.css`. При использовании Sass вы *никогда* не редактируете непосредственно CSS-файлы; вы работаете только с файлами с расширением `.scss`.

Поскольку используется два различных типа файлов, наиболее распространенный способ организовать их заключается в создании двух отдельных каталогов в корне вашего сайта: папка `css` используется для хранения финальных CSS-файлов, скомпилированных с помощью Sass, и другая папка — для рабочих Sass-файлов. Вы можете назвать ее как угодно, но многие веб-дизайнеры используют имена `scss` или `sass`. Внутри этой папки создаются Sass-файл с именами в соответствии с тем, как должны называться финальные CSS-файлы. Например, если вы привыкли использовать имя `styles.css`, то присвойте Sass-файлу имя `styles.scss`. Когда препроцессор Sass обработает файл `styles.scss`, он автоматически создаст файл `styles.css`.

Преобразование текущего сайта в Sass

Поскольку Sass отлично поддерживает обычный CSS-код, вы можете начать использовать его с уже существующим файлом, выполнив несколько простых действий, описанных ниже.

1. Добавьте папку с именем `sass` в корневой каталог сайта.

Убедитесь, что в корневом каталоге вашего сайта находится папка `css` и в ней расположена таблица стилей сайта.

2. Переместите таблицу стилей сайта из папки `css` в папку `sass`.

Ничего страшного, если на вашем сайте нет папки `css`. Возможно, вы использовали имя папки `styles` или подобное. В этом случае вы будете применять имя этой папки при выполнении команды `sass`, как будет описано в следующем разделе.

Кроме того, если в папке находится более одного файла таблицы стилей, например файлы `reset.css` и `styles.css`, также переместите их в папку `sass`. Как вы увидите далее в этой главе, в проекте вы будете часто использовать несколько Sass-файлов.

3. Переименуйте файл с таблицей стилей в папке `sass` таким образом, чтобы он имел расширение `.scss`.

Например, если файл называется `site.css`, измените его расширение: `site.scss`. Если вы используете более одного CSS-файла, к примеру, еще файл с именем `reset.css`, измените расширение на `.scss` и у него.

Конечно, перемещение CSS-файлов в другую папку на вашем сайте означает, что веб-страницы не смогут обнаружить их и браузер не будет применять какое-либо форматирование к сайту. Но это временно. Когда вы запустите команду `sass`, вы получите новую таблицу стилей в файле `.css`.

Запуск препроцессора Sass

Чтобы преобразовать Sass-документ в CSS-файл, вам нужно выполнить команду `sass`, используя командную строку (приложение `cmd` в операционной системе Windows или Терминал (Terminal) в OS X). Команда `sass` содержит одну очень интересную функцию, `watch`, которая постоянно отслеживает любые изменения, внесенные в файл `.scss`. Если Sass обнаруживает изменение, она автоматически генерирует новый CSS-файл. Эта функция особенно полезна в процессе создания

вашего сайта. Браузеры не поддерживают Sass-файлы, поэтому, просто изменив код в Sass-файле, вы не увидите никаких изменений на странице. Тем не менее, когда используется аргумент `--watch` команды `sass`, вы можете сразу увидеть любые изменения, внесенные в Sass-файл. Для этого просто обновите окно браузера, чтобы загрузить обновленную таблицу стилей.

Чтобы запустить Sass, выполните следующие действия.

1. Запустите приложение командной строки в операционной системе Windows или приложение **Терминал (Terminal)** в OS X.

Чтобы вспомнить, как это сделать, вернитесь к шагу 5 в подразделе «Установка Sass в операционной системе Windows» для пользователей Windows или к шагу 1 в подразделе «Установка Sass в операционной системе OS X» для пользователей OS X.

2. Перейдите к корневому каталогу вашего сайта с помощью командной строки.

Чтобы начать выполнять команды Sass, необходимо перейти к корневой папке, в которой находятся каталоги `sass` и `css`. Вам нужно использовать следующую команду: `C:\Users\Vasya\Desktop\site` или `cd ~/Documents/site`. Имена путей у вас будут иными.

Если вы не знаете, как перемещаться по каталогам с помощью командной строки, используйте следующий способ, который применим как в операционной системе Windows, так и в OS X:

- 1) запустите приложение командной строки или **Терминал (Terminal)**;
- 2) в программе **Проводник (Explorer)** или **Finder** откройте каталог с вашим сайтом.

Вы должны видеть окна командной строки и папки сайта одновременно (рис. 19.5);

- 3) в командной строке введите `cd` и нажмите клавишу **Пробел**.

`cd` — это команда, заимствованная из операционной системы Unix. Она означает смену каталога — **Change Directory** — и используется для перехода из одной папки в другую с помощью командной строки. Не забудьте поставить пробел, который отделяет команду `cd` от пути, который вы добавите на следующем шаге;

ПРИМЕЧАНИЕ

Чтобы узнать, как использовать командную строку для навигации по каталогам файловой системы, посетите сайт tinyurl.com/bgsslfk.

- 4) перетащите значок папки из окна программы **Проводник (Explorer)** или **Finder** в окно командной строки (см. рис. 19.5, 1).

Путь — маршрут от верхнего уровня файловой системы на жестком диске вашего компьютера к папке с сайтом — будет добавлен в командную строку автоматически;

ПРИМЕЧАНИЕ

Чтобы узнать, как использовать окно программы **Терминал (Terminal)** для навигации по файловой системе, посетите сайт tinyurl.com/ne3h2t6.

5) нажмите клавишу **Enter**.

Окно командной строки изменится, отобразив каталог с сайтом. Теперь вы находитесь в папке с сайтом. Любые команды, которые вы будете выполнять в командной строке, с этого момента будут использовать эту папку в качестве расположения.

3. В командной строке введите команду `sass --watch sass:css` и нажмите клавишу **Enter**.

Она запустит препроцессор Sass. Он проанализирует содержимое папки `sass` на предмет любых файлов, заканчивающихся расширением `.scss`, преобразует их содержимое в обычный CSS-код, а затем создаст соответствующие CSS-файлы в папке `css`. На изображении 2 на рис. 19.5 показано, что у вас должно получиться.

Поскольку препроцессор Sass отслеживает изменения в Sass-файлах, то, если вы модифицируете файл `.scss` внутри папки `sass`, в папке `css` будет автоматически создан новый файл с расширением `.css`.

Если для Sass-файлов вы используете папку с именем, отличным от `sass`, указывайте в команде ее имя вместо `sass`. Например, если ваша папка называется `scss`, используйте команду `sass --watch scss:css`.

СОВЕТ

Препроцессор Sass также может оптимизировать финальный CSS-файл, удалив комментарии и лишние пробелы, а также внести другие изменения, которые позволят уменьшить размер CSS-файла. В результате файл таблицы стилей у посетителей вашего сайта будет загружаться максимально быстро. Чтобы сжать CSS-файлы с помощью Sass, введите команду `sass --watch scss:css --type compressed`.

После запуска команды `sass` с аргументом `--watch` она будет работать непрерывно. Если вы хотите прекратить отслеживание изменений, нажмите сочетание клавиш **Ctrl+C** в окне приложения командной строки или Терминал (Terminal). Если вы закроете окно командной строки или перезагрузите компьютер, то команду `sass` придется выполнить снова, следуя инструкциям, приведенным выше.

Как вы могли заметить, чтобы начать использовать препроцессор Sass, необходимо выполнить ряд действий. Однако, как только он заработает, начнется самое интересное.

Организация стилей с помощью фрагментов Sass

Красивые сайты используют огромное количество каскадных таблиц стилей. И чем больше и сложнее сайт, тем сложнее CSS-код. Нетрудно создать сотни или даже тысячи стилей для сайта, но они сделают CSS-файл огромным. Некоторые веб-дизайнеры разделяют CSS-код на меньшие, управляемые файлы, но чем больше файлов вы добавляете на сайт, тем больше времени посетителям придется потратить на запросы и обработку файлов. Многие профессиональные веб-дизайнеры рекомендуют по возможности использовать несколько CSS-файлов.

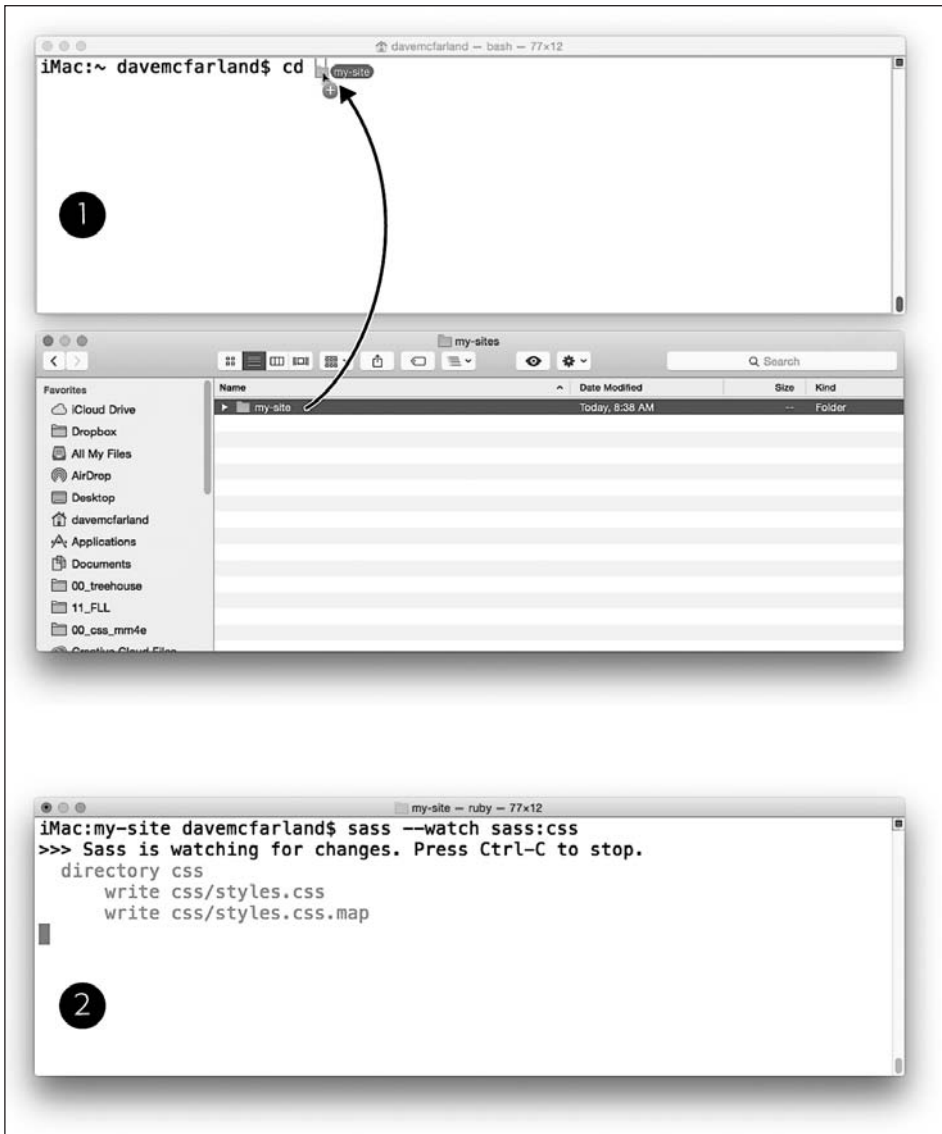


Рис. 19.5. Чтобы препроцессор Sass преобразовывал Sass-файлы в CSS-файлы, необходимо через командную строку перейти в каталог, где находятся файлы вашего сайта (1). Чтобы преобразовать файлы .scss в .css (2), используйте команду `sass`

Sass позволяет применять преимущества обоих подходов. Вы можете разделить правила Sass на отдельные файлы, логически организованные. Например, для правил сброса использовать один файл, для правил модульной верстки — второй, для правил типографики — третий и т. д. Вы можете указать препроцессору Sass собрать все эти файлы в единый CSS-файл или даже сжать его, чтобы он занимал меньший объем и загружался быстрее.

В самом деле, даже если вы больше не хотите изучать Sass, уже эта его особенность является достаточной причиной, чтобы использовать Sass-стили. Возможность организовать CSS-код в отдельные файлы означает, что вы сможете легко найти нужный файл для редактирования: «Я хочу изменить внешний вид этой веб-формы — правила, которые отвечают за него, находятся в файле `forms.scss`». Помните, вы можете разместить правила CSS в любом Sass-файле (см. раздел «Основы Sass» выше), что позволяет использовать Sass как способ разделения и организации каскадов правил CSS на управляемые файлы.

Создание и использование фрагментов Sass

Вы можете разделить CSS/Sass-правила на несколько файлов, а затем объединить их в один CSS-файл. В Sass эти файлы называются *фрагментами*. Существует специальный способ присвоения им имен и их использования. Чтобы сообщить препроцессору Sass, что вы *не* хотите преобразовывать эти фрагменты в отдельные CSS-файлы, их имена должны начинаться с символа подчеркивания (`_`).

Например, можно разделить код стилей на четыре разных файла, чтобы было проще создавать, искать и обновлять стили. В этом случае имя каждого файла должно начинаться с символа подчеркивания, например `_reset.scss`, `_grid.scss`, `_layout.scss` и `_forms.scss`. Теперь, если вы выполните команду `sass`, приложение Sass не преобразует эти четыре файла в CSS-файлы `reset.css`, `grid.css`, `layout.css` и `forms.css` и не поместит их в папку `css`. На самом деле препроцессор Sass будет игнорировать их до тех пор, пока вы явно не укажете их в команде.

Конечно, вам не нужно, чтобы эти фрагменты просто лежали в каталоге. Они были созданы для окончательного CSS-файла, который сгенерирует препроцессор Sass. В этом случае используйте инструкцию `@import`, чтобы скомпилировать в итоговый файл определенный фрагмент, например:

```
@import '_reset.scss';
```

Наиболее распространенный способ использования фрагментов заключается в том, чтобы подготовить один финальный Sass-файл, который будет преобразован в законченный CSS-файл и который не содержит ничего, кроме инструкций импорта. Например, требуется создать одиночный CSS-файл с именем `styles.css`. Для этого можно использовать Sass-файл с именем `styles.scss` и поместить его в папку `sass`. Содержимое Sass-файла может выглядеть следующим образом:

```
@import '_reset.scss';  
@import '_grid.scss';  
@import '_layout.scss';  
@import '_forms.scss';
```

Поскольку имя файла `styles.scss` *не* начинается с символа подчеркивания, препроцессор Sass обработает его и создаст файл `styles.css` в папке `css`. В нем нет ничего особенного: Sass просто скопирует код из каждого фрагмента и добавит его в единый CSS-файл. Порядок, в котором перечислены фрагменты, определяет приоритет их добавления в CSS-файл, поэтому в приведенном выше примере CSS-файл вначале будет содержать правила сброса, затем модульной сетки, затем компоновки

макета, и в самом конце — веб-форм. Не забывайте, что порядок, в котором стили располагаются в файле, может повлиять на работу принципа каскадности (см. главу 5). Указывая инструкции `@import`, следует помнить об этом.

Препроцессор Sass довольно умен по отношению к фрагментам, поэтому при их импорте в другой Sass-файл для обозначения вы можете использовать сокращения: символ подчеркивания (`_`) и расширение `.scss` можно опустить. При этом указанный выше код может быть переписан следующим образом:

```
@import 'reset';
@import 'grid';
@import 'layout';
@import 'forms';
```

Директива `@import` — один из моментов вашего знакомства с синтаксисом Sass. Хотя она и является частью языка CSS, в данном примере используется исключительно как элемент Sass. Если вы попытаетесь использовать инструкцию `@import` в этом виде в обычном CSS-файле, браузер не будет знать, что с ней делать, и, конечно, не сможет объединить все файлы в один CSS-файл.

Организация файлов фрагментов Sass

Многие веб-дизайнеры, которые используют Sass, хранят свои файлы в определенной структуре. Используя несколько Sass-файлов, они помещают их в подпапки, группируя по функционалу. Существует множество способов организации файлов. Ниже представлена простая диаграмма базовой структуры папок с Sass-файлами:

```
sass/
|
|-- helpers/           # специальные Sass-файлы
|   |-- _variables.scss
|   |-- _mixins.scss
|
|-- base/             # базовые файлы для проекта
|   |-- _reset.scss
|   |-- _grid.scss
|   |-- _typography.scss
|
|-- layout/           # файлы форматирования элементов страницы
|   |-- _header.scss
|   |-- _footer.scss
|   |-- _sidebar.scss
|   |-- _forms.scss
|
|-- components/       # файлы для компонентов интерфейса
|   |-- _buttons.scss
|   |-- _dropdown.scss
|   |-- _navigation.scss
|
|-- styles.scss       # основной Sass-файл
```

Внутри основной папки `sass` находится `styles.scss` — основной файл, который препроцессор Sass преобразует в CSS-файл. Кроме того, есть четыре вложенные папки для хранения файлов, которые выполняют особые функции в дизайне сайта, а также несколько папок для специфических Sass-файлов, о которых вы узнаете чуть позже в этой главе. Чтобы использовать все эти фрагменты, вы должны импортировать их в файл `styles.scss`. Чтобы импортировать в документ `styles.scss` этот конкретный набор файлов, необходимо использовать следующий код:

```
@import 'helpers/variables';
@import 'helpers/mixins';
@import 'base/reset';
@import 'base/grid';
@import 'base/typography';
@import 'layout/header';
@import 'layout/footer';
@import 'layout/sidebar';
@import 'layout/forms';
@import 'components/buttons';
@import 'components/dropdown';
@import 'components/navigation';
```

Обратите внимание, что вы должны указывать имена папок (чтобы препроцессор Sass обнаружил нужные файлы), но можно опустить подчеркивания и расширение `.scss` в именах файлов.

ПРИМЕЧАНИЕ

Дополнительную информацию о способах организации Sass-файлов можно найти на сайтах tinypurl.com/ctemo55 и tinypurl.com/zc5c3or.

Переменные Sass

Вы, вероятно, сталкивались со случаями, когда использовали одно и то же значение снова и снова в CSS-коде. Например, основным цветом логотипа вашей компании является синий: `#083B91`. Отдел маркетинга требует, чтобы вы применяли его для всех элементов, поэтому вы используете его в качестве цвета фона панели навигации, цвета шрифта заголовков `h1` и цвета границ боковых панелей.

Таким образом, вы создаете несколько десятков правил каскадных таблиц стилей, которые используют этот цвет. Но что произойдет, если отдел маркетинга решит изменить значение цвета на одну цифру, например `#083B92`? Конечно, чтобы перейти к новому цвету, вы можете воспользоваться поиском с заменой, но разве не было бы замечательно, если бы можно было изменить одно значение в одном файле, а цвет обновился бы во всем CSS-коде?

Вы можете реализовать это с помощью переменных Sass. *Переменная* — это атрибут, которому присвоено значение и которое может изменяться — прямо как цвет логотипа вашей компании. Вы сохраняете значение в переменной один раз, а затем используете ее там, где это необходимо. Если значение, сохраненное в переменной, будет обновлено, оно обновится везде, где вы использовали переменную

в Sass-файлах. Другими словами, вы вносите изменения в один файл, а препроцессор Sass обновляет остальные файлы автоматически.

Чтобы использовать переменную Sass, необходимо выполнить два шага. Во-первых, вы должны определить переменную, то есть присвоить ей имя, а затем — присвоить ей значение. В Sass имена переменных начинаются с символа доллара, а значения присваиваются с помощью двоеточия (:). Например, чтобы создать переменную для цвета логотипа вашей компании, добавьте следующий код в Sass-файл:

```
$logo_color: #083B91;
```

Обратите внимание, что этот тип записи очень похож на объявление в языке CSS, за исключением того, что `$logo_color` не является допустимым свойством каскадных таблиц стилей. Это синтаксис Sass, и он означает, что препроцессор Sass идентифицирует эту конкретную строку кода в качестве своей переменной. Так же как Sass преобразует Sass-файлы в CSS-файл, препроцессор заменяет имя переменной ее значением, то есть в данном случае значением цвета.

Но, чтобы заставить этот трюк работать, эту переменную необходимо использовать в правиле CSS. Переменную можно использовать в любой позиции, где бы вы вставили обычное значение CSS. Например, чтобы назначить с помощью переменной цвет элементу `h1`, используйте в Sass-файле следующую запись:

```
h1 {  
  color: $logo_color;  
}
```

После выполнения препроцессором Sass *компиляции* (то есть преобразования) кода в финальном CSS-файле появится следующее правило:

```
h1 {  
  color: #083B91;  
}
```

Организация переменных Sass

Вначале вы будете использовать несколько переменных, возможно, для нескольких часто используемых цветов. Однако по мере работы с Sass вы найдете десятки возможностей, чтобы использовать переменные. Чтобы хранить все переменные в одном месте таким образом, чтобы все другие Sass-файлы имели к ним доступ, мы рекомендуем создать отдельный файл специально для переменных.

Один из способов сделать это — создать файл с именем `_variables.scss` и поместить его в папку со специальными Sass-файлами. В подразделе «Организация файлов фрагментов Sass» предыдущего раздела приведен пример одного из способов организации Sass-файлов; папка `helpers` прекрасно подойдет.

Помните, что Sass-файлы, имена которых начинаются с символа `_`, являются фрагментами Sass. Препроцессор Sass создает конечный CSS-файл путем объединения нескольких файлов фрагментов. Порядок, в котором фрагменты перечислены в основном Sass-файле, определяет приоритет их считывания и обработки препроцессором. Допустим, что у вас есть файл с именем `styles.scss`, а внутри него приведены инструкции импорта трех фрагментов:


```
@import 'base/reset';  
@import 'base/grid';  
@import 'base/layout';
```

Когда вы выполните команду `sass`, сначала будет считано содержимое файла `_reset.scss`. Оно будет добавлено в начале нового файла — `styles.css`. Остальная часть этого CSS-файла будет содержать код из файлов фрагментов `_grid.scss` и `_layout.scss` в соответствующем порядке.

Смысл использования переменных в Sass заключается в том, чтобы однократно присваивать им определенные значения и использовать затем в Sass-файлах всякий раз, когда это необходимо. Поскольку другие файлы должны знать присвоенное вами значение переменной, вы должны убедиться, что переменные, которые вы создаете, загружаются в *первую* очередь. Другими словами, рекомендуется импортировать переменные перед любыми другими файлами фрагментов:

```
@import 'helpers/variables';  
@import 'base/reset';  
@import 'base/grid';  
@import 'base/layout';
```

Основные способы использования переменных Sass

Переменные Sass обычно применяются для хранения значений цвета. Основываясь на выборе отдела маркетинга вашей компании или вашем собственном мнении, вы можете с завтрашнего дня полностью изменить цвета, используемые на вашем сайте.

Простой подход заключается в определении цвета для различных элементов на странице, таких как шрифт текста, баннер, границы и фон:

```
$text-color: #333333;  
$headline-color: #ff0000;  
$border-color: #0032ff;  
$background-color: #FFEE99;
```

Добавив эти стили в файл `_variables.scss`, вы сможете использовать их в других фрагментах Sass. Многие дизайнеры рекомендуют для переменных цвета применять двухэтапный подход: сначала создать набор переменных, который определяет палитру цветов, а затем назначить эти переменные другим переменным, определяющим особенности использования цвета.

Скажем, цветовая палитра вашей компании состоит из пяти цветов. Для начала вы могли бы назначить переменные для этих цветов. Затем назначить эти переменные цвета с функциональными именами конкретным элементам страницы или свойствам CSS, например:

```
/* корпоративные цвета */  
$primary-dark: #06888A;  
$primary-light: #FFEE99;  
$primary-mid: #DC664A;
```

```
$secondary-dark: #5A3928;
$secondary-light: #FDC149;

/* функциональные переменные */
$page-background: $primary-light;
$headline-color: $primary-dark;
$text-color: $primary-dark;
$border-light: $secondary-light;
$border-dark: $primary-dark;
```

Такой подход позволяет легко использовать один и тот же цвет для нескольких компонентов. Например, в этом коде текст и заголовок имеют одинаковый цвет: #06888A. Но вместо того, чтобы назначать этот цвет два раза:

```
$headline-color: #06888A;
$text-color: #06888A;
```

вы присвоили его один раз переменной `$primary-dark`. Если вы задумаете изменить цветовую палитру своего сайта, просто измените значение переменной `$primary-dark`, и цвета текста и заголовка изменятся. Кроме того, если позже вы решите задействовать другой цвет для текста, это легко сделать, изменив значение одной из вторых переменных, которые вы создали для вашей цветовой палитры:

```
$headline-color: $primary-dark;
$text-color: $secondary-dark;
```

Переменные также могут быть использованы для создания *стека шрифтов* — списка шрифтов, которые вы хотите использовать для различных элементов страницы, таких как заголовки и основной текст (см. раздел «Использование шрифтов» главы 6). Например, простой метод заключается в том, чтобы назначить один стек шрифта переменной, используемой для заголовка текста, а другой — для основного текста, например:

```
$headline-font: 'Varela Round', Helvetica, Arial, sans-serif;
$body-font: 'Palatino Linotype', Baskerville, serif;
```

После этого можно использовать эти переменные в других фрагментах Sass. Допустим, что у вас есть файл фрагмента с именем `_typography.scss`. Вы можете использовать переменные стека шрифта в других стилях:

```
body {
  font-family: $body-font;
}
h1, h2, h3 {
  font-family: $headline-font;
}
```

Если позже вы решите изменить шрифты в стеках, просто обновите переменную `$headline-font` или переменную `$body-font`.

Начав использовать переменные Sass, вы обнаружите множество применений для них. Например, если вы хотите скруглить углы, используйте переменную, ко-

торая определяет степень скругления углов элементов на вашей странице, таких как боковые панели, изображения и т. д.:

```
$border-radius: 6px;
```

Если вам покажется, что радиус скругления слишком мал или велик, измените переменную. Все стили будут обновлены, а элементы страницы будут использовать новое значение радиуса.

ДЛЯ ОПЫТНЫХ ПОЛЬЗОВАТЕЛЕЙ

Позвольте Sass делать математические расчеты

Препроцессор Sass может даже выполнять базовые математические расчеты. Нет, он не заменит калькулятор, но вы можете использовать его для записи новых значений, основанных на математических расчетах. Обычно вы будете делать это с помощью переменных Sass.

Допустим, вы хотите установить стандартные поля для заголовков: нижнее поле всегда будет составлять половину от верхнего. Чтобы получить некоторую гибкость, вы можете настроить эти поля с помощью переменной и начать с определения переменной для верхнего поля:

```
$margin-top: 20px;
```

Затем, вместо того чтобы применить простое число, например 10px, в качестве значения нижнего поля, создайте новую переменную и используйте математический расчет:

```
$margin-bottom: $margin-top / 2;
```

С помощью этого способа вы обеспечиваете значительную гибкость. Если 20 пикселей недостаточно, измените значение переменной `$margin-top`. Значение

переменной `$margin-bottom` изменится автоматически. Пропорции 1:2 сохранятся.

При сложении и вычитании необходимо использовать одни и те же единицы измерений. Например, присвоив значение 20px переменной `$margin-top`, вы не можете выполнить следующее математическое действие:

```
$margin-bottom: $margin-top + 1em;
```

Препроцессор Sass не знает, как добавить пиксели к единицам `em` и отобразит ошибку при попытке компиляции Sass-кода. Однако, используя одинаковые величины, можно складывать и вычитать:

```
$margin-bottom: $margin-top - 5px;
```

Существует много примеров математических действий в Sass. Вы изучите один из них в разделе «Использование медиазапросов» далее, когда вы будете использовать Sass для создания медиазапроса. Чтобы узнать подробную информацию об использовании математических расчетов в Sass, посетите сайты tinyurl.com/qdquo65, tinyurl.com/npxfy8t и русскоязычный ресурс tinyurl.com/perlp4h.

Вложенные селекторы

Ранее в этой книге вы познакомились с селекторами потомков и некоторыми ситуациями, когда они могут быть полезны (см. раздел «Использование селекторов потомков» главы 18). Как вы помните, селектор потомков состоит из нескольких селекторов, определяющих вложенность элементов. Например, селектор `.nav a` расшифровывается как «выделить все элементы `a`, которые находятся внутри другого элемента с классом `nav`».

Часто вы будете использовать селекторы потомков, чтобы определить вид различных элементов внутри более крупного компонента. Допустим, вы применяете следующий HTML-код для создания панели навигации:

```
<ul class="nav">
  <li><a href="index.html">Главная</a></li>
  <li><a href="about.html">О нас</a></li>
  <li><a href="contact.html">Контакты</a></li>
</ul>
```

Вы могли бы создать ряд селекторов для форматирования различных частей этой панели навигации. К примеру, в этом коде используется три элемента, поэтому селекторы могут выглядеть следующим образом:

```
.nav {
  display: flex;
  justify-content: space-around;
  list-style-type: none;
}
.nav li {
  width: 30%;
  background-color: #FFEED5;
  padding: 5px;
  text-align: center;
}
.nav a {
  text-decoration: none;
}
```

Эти стили применяют метод flexbox-верстки (см. главу 17), позволяющий разместить три кнопки рядом друг с другом. Не волнуйтесь о перечисленных свойствах: в данном случае мы акцентируем внимание на селекторах. Селекторы потомков гарантируют, что не каждый элемент списка (`li`) или элемент привязки (`a`) будет отформатирован с использованием этих правил; только те из них, которые находятся на панели навигации.

Метаязык Sass позволяет сохранить стили, наподобие этих, сгруппированными вместе, в то же время уменьшая количество кода, который необходимо написать, чтобы получить нужный результат. Чтобы сделать это, вы примените *вложенный селектор* Sass. Во-первых, обратите внимание, что все три селектора в примере выше начинаются со значения `.nav`. Это будет *базовый*, или основной, селектор:

```
.nav {
  display: flex;
  justify-content: space-around;
  list-style-type: none;
}
```

Затем вместо того, чтобы создавать два новых стиля с селекторами `.nav li` и `.nav a`, вы просто вложите элемент `li` и селекторы внутрь, как это сделано, например, в следующем коде:

```
.nav {
  display: flex;
  justify-content: space-around;
  list-style-type: none;

  li {
    width: 30%;
    background-color: #FFEEED5;
    padding: 5px;
    text-align: center;
  }

  a {
    text-decoration: none;
  }
}
```

Обратите внимание, что CSS-код для элементов `li` и `a` расположен внутри скобок `{ }` стиля `.nav` — это *вложение*. Это не CSS-, а Sass-код, поэтому после компиляции окончательный CSS-код будет выглядеть следующим образом:

```
.nav {
  display: flex;
  justify-content: space-around;
  list-style-type: none;
}
.nav li {
  width: 30%;
  background-color: #FFEEED5;
  padding: 5px;
  text-align: center;
}
.nav a {
  text-decoration: none;
}
```

Это тот же код, который вы видели выше, но он немного короче и содержит стили, сгруппированные во внешнем контейнере `.nav`.

Ссылка на родительский селектор

В предыдущем примере вы могли заметить, что препроцессор Sass при компиляции окончательного CSS-файла автоматически добавляет имя внешнего селектора, который оборачивается вокруг вложенных селекторов. Например, селектор `.nav` добавляется к `li` в Sass-коде, чтобы создать селектор `.nav li` в CSS-коде. По умолчанию препроцессор Sass указывает имя родительского селектора, затем пробел, а затем имя вложенного селектора.

Но бывают случаи, когда не требуется этот пробел. Допустим, вы хотите использовать вложенные селекторы Sass для создания псевдоклассов, применяемых

для различных состояний ссылок. Вы можете попробовать использовать следующий Sass-код:

```
a {
  color: blue;
  :hover {
    color: green;
  }
}
```

Финальный CSS-код будет выглядеть следующим образом:

```
a {
  color: blue;
}
a :hover {
  color: green;
}
```

На первый взгляд код выглядит хорошо, но обратите внимание на промежуток между `a` и `:hover`. Благодаря ему текст изменит цвет *внутри* элемента `a`, а не самого элемента привязки и только при установке указателя мыши поверх вложенного элемента. То, что вам действительно нужно, — `:hover` без пробела. Чтобы сделать это, препроцессор Sass использует символ `&` (амперсанд). Когда команда `sass` встречает символ `&`, она замещает его именем родительского селектора. Таким образом, чтобы получить правильный CSS-код для приведенного выше примера, необходимо использовать следующий Sass-код:

```
a {
  color: blue;
  &:hover {
    color: green;
  }
}
```

Между символом `&` и `:hover` нет пустого пространства, поэтому в результате вы получите правильный псевдокласс `a:hover`. Если вы хотите отформатировать все четыре псевдокласса для ссылок, вы могли бы использовать следующий шаблон Sass-кода вложенного селектора:

```
a {
  &:link {

  }
  &:visited {

  }
  &:hover {

  }
  &:active {

  }
}
```

Точно так же, если вы примените класс `button`, например в некоторых ссылках, и захотите использовать псевдоклассы, такие как `.button:link`, `.button:visited`, `.button:hover` и `.button:active`, используйте следующий код:

```
.button {
  &:link {

  }
  &:visited {

  }
  &:hover {

  }
  &:active {

  }
}
```

Поскольку препроцессор Sass буквально заменяет символ `&` именем родительского селектора, вы можете использовать принцип вложенности, чтобы создать группу имен классов, которые зависят от иерархии селекторов потомков. Например, некоторые дизайнеры не беспокоятся о точном вложении или вводе HTML-элементов, использующихся для создания элементов страницы. Они просто применяют имена классов к каждому (или большинству) HTML-элементов. Например, вместо следующего HTML-кода:

```
<div class="main">
  <h1>Заголовок</h1>
  <div>

  </div>
</div>
```

и опираясь на CSS-селекторы, такие как `.main`, `.main h1`, `.main div`, они используют HTML-код наподобие показанного ниже:

```
<div class="main">
  <h1 class="main-title">Заголовок</h1>
  <div class="main-content">
  </div>
</div>
```

И создают стили, такие как `.main`, `.main-title` и `.main-content`. Благодаря принципу вложенности и символу `&` препроцессор Sass позволяет легко это сделать.

```
.main {
  &-title {

  }

  &-content {

  }
}
```

Многоуровневая вложенность

Препроцессор Sass позволяет вкладывать селекторы почти бесконечно. В конце концов, это всего лишь компьютерная программа, и она может определить, что делать с 50 уровнями вложенных селекторов. Но для простых людей лучше ограничить глубину вложения селекторов — в большинстве ситуаций одного или двух уровней будет вполне достаточно. Более того, при значительном количестве вложений достаточно легко заблудиться в селекторах.

СОВЕТ

Большая глубина вложенности также означает, что вы слишком углубились с вложением HTML-элементов в коде веб-страницы. Это делает ваш CSS-код «хрупким», поскольку простое изменение структуры HTML-кода может привести к тому, что сложная система селекторов потомков перестанет функционировать.

Чтобы вложить несколько уровней селекторов, поместите селектор внутри уже вложенного селектора. Скажем, в примере, рассмотренном выше, вы хотели бы добавить псевдокласс `:hover` к каждой ссылке на панели навигации. Вы можете вложить стиль `:hover` в селектор, вложенный, в свою очередь, в селектор `.nav`:

```
.nav {
  display: flex;
  justify-content: space-around;
  list-style-type: none;

  li {
    width: 30%;
    background-color: #FFEEED5;
    padding: 5px;
    text-align: center;
  }

  a {
    text-decoration: none;

    &:hover {
      text-decoration: underline;
    }
  }
}
```

Обратите внимание, что вам необходимо использовать символ `&`, чтобы сослаться на родительский селектор, и что код `&:hover` вложен в селектор.

В дополнение к стилям, приведенным ранее, препроцессор Sass скомпилирует приведенный выше код, чтобы создать следующее многоуровневое вложение:

```
.nav a:hover {
  text-decoration: underline;
}
```


Как вы, вероятно, заметили, используя многоуровневое вложение селекторов в Sass, можно очень быстро запутаться. Но осторожное применение вложенных селекторов позволяет уменьшить объем CSS-кода, а также организовать и сгруппировать их по предназначению.

Наследование (или расширение) свойств

В главе 3 вы узнали, как групповые селекторы позволяют применять один и тот же набор свойств к нескольким элементам страницы. Например, чтобы назначить один и тот же шрифт и цвет для всех заголовков, можно использовать групповой селектор наподобие следующего:

```
h1, h2, h3, h4, h5, h6 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
```

Применение групповых селекторов означает, что вам не нужно повторять один и тот же набор свойств снова и снова. Без группового селектора одиночный стиль, представленный выше, превратится в шесть отдельных стилей:

```
h1 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h2 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h3 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h4 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h5 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h6 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
```

Дело не только в объеме кода. Если вы захотите изменить цвет шрифта для всех шести заголовков, вам придется обновить шесть строк кода вместо одной. Метаязык Sass обеспечивает способ, который позволяет стилю наследовать свойства другого стиля. В приведенном выше примере основные шесть заголовков,

начиная с `h1` и заканчивая `h6`, используют один и тот же набор свойств. Другими словами, они наследуют, или, как говорят программисты, *расширяют*, базовый набор свойств.

Метаязык `Sass` позволяет применить свойства текущего стиля к другим стилям, расширяя оригинальный стиль. Вы видели простой пример, касающийся стилей заголовков, выше. Предположим, вы хотите применить одно и то же значение свойств `font-family` и `color` к заголовкам `h1`, `h2` и `h3` (приведенные здесь заголовки взяты только в качестве примера, вы можете использовать любые другие). Вы могли бы начать со стиля `h1` и с помощью инструкции `@extend` `Sass` применить его к другим заголовкам следующим образом:

```
h1 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h2 {
  @extend h1
}
h3 {
  @extend h1
}
```

Сейчас вы, наверное, спросите: «Не проще ли использовать групповой селектор?» Вся мощь инструкции `@extend` проявляется тогда, когда вы добавляете дополнительные свойства к каждому селектору таким образом, что конкретные стили имеют только *некоторые* общие свойства, в то время как другие свойства отличаются. Допустим, вы хотите добавить границу над заголовками `h2` и отступ к заголовкам `h3`. В то же время эти заголовки должны использовать одинаковые шрифт и цвет. В этом случае `Sass`-код может выглядеть следующим образом:

```
h1 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h2 {
  @extend h1;
  border-top: 1px solid #444;
}
h3 {
  @extend h1;
  margin-left: 20px;
}
```

В процессе компиляции препроцессор `Sass` преобразует код, представленный выше, в следующий:

```
h1, h2, h3 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h2 {
```

```
border-top: 1px solid #444;
}
h3 {
margin-left: 20px;
}
```

Расширение стилей имеет несколько преимуществ: во-первых, количество необходимых селекторов уменьшается. При использовании группового селектора вы получаете два различных стиля для одного и того же элемента. Например, в приведенном выше CSS-коде селектор `h2` используется как в групповом селекторе `h1`, `h2`, `h3`, так и в отдельном селекторе `h2`.

Во-вторых, расширение стилей позволяет хранить все свойства CSS-элемента в одном селекторе. Это означает, что, если вы хотите изменить внешний вид элементов `h2`, вам не нужно искать два стиля — групповой и индивидуальный селекторы. Просто найдите селектор `h2` и, если хотите изменить шрифт и цвет, удалите инструкцию `@extend`. Или, если хотите изменить цвет всех заголовков, найдите наследованный селектор (в данном примере это `h1`) и измените его.

ПРИМЕЧАНИЕ

Директиву `@extend` можно поместить в любой позиции в пределах стиля, например на первой или последней строке. Тем не менее большинство дизайнеров помещает инструкцию `@extend` перед какими-либо другими свойствами. Такой подход позволяет гораздо проще определять, что данный стиль основан на другом стиле.

Расширение с селекторами-заполнителями

Метаязык Sass довольно агрессивен в отношении расширения селекторов. Расширение будет распространяться не только на тот селектор, который вы указываете, но и на другие стили, на которые он ссылается. Это обычно приводит к нежелательным последствиям. Предположим, что в файле `.scss` находится следующий код:

```
#main h1 {
background-color: blue;
}
h1 {
font-family: "Raleway", Helvetica, Arial, sans-serif;
color: #222;
}
h2 {
@extend h1;
border-top: 1px solid #444;
}
h3 {
@extend h1;
margin-left: 20px;
}
```

Первый стиль (`#main h1`) добавляет синий фон к любому заголовку `h1` внутри элемента с идентификатором `main`. К сожалению, когда стили `h2` и `h3` будут расширены

стилем `h1`, они также распространятся и на стиль `#main h1`. В результате CSS-код будет выглядеть следующим образом:

```
#main h1, #main h2, #main h3 {
  background-color: blue;
}
h1, h2, h3 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h2 {
  border-top: 1px solid #444;
}
h3 {
  margin-left: 20px;
}
```

Обратите внимание, что препроцессор Sass создает групповой селектор для всех заголовков `h1`, `h2` и `h3` внутри основного элемента. Ситуация становится еще хуже, если у вас есть другие стили, которые содержат селектор `h1`, например `h1 span` или `h1 a`. Препроцессор Sass создаст групповые селекторы и для них тоже! Чтобы можно было обойти эту проблему, Sass содержит инструмент под названием *селектор-заполнитель*. Его имя начинается с символа `%`. Вы не можете использовать этот символ в имени обычного CSS-селектора, но в языке Sass этот символ указывает на стиль, который вы хотите применять только в качестве основы для других стилей.

Например, чтобы избежать проблемы *излишнего* расширения селектора `h1` в приведенном выше примере, вы могли бы создать селектор-заполнитель и расширить его следующим образом:

```
#main h1 {
  background-color: blue;
}
%headline {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h1 {
  @extend %headline;
}
h2 {
  @extend %headline;
  border-top: 1px solid #444;
}
h3 {
  @extend %headline;
  margin-left: 20px;
}
```

На этот раз запись `%headline` не является настоящим CSS-селектором. Это только инструмент Sass, поэтому при создании финального CSS-кода в CSS-файле не будет стиля `%headline`. Поскольку на селектор `h1` не распространяются другие сти-

ли, верхний стиль (`#main h1`) не расширится и конечный CSS-код будет выглядеть так, как нам нужно:

```
#main h1 {
  background-color: blue;
}
h1, h2, h3 {
  font-family: "Raleway", Helvetica, Arial, sans-serif;
  color: #222;
}
h2 {
  border-top: 1px solid #444;
}
h3 {
  margin-left: 20px;
}
```

Расширение только связанных элементов

Если вы не будете достаточно осторожны, то расширение стилей с помощью Sass быстро может выйти из-под контроля. Вы могли бы попытаться придумать заполнитель для стиля конкретной границы (скажем, синяя пунктирная линия шириной 1 пиксел, с радиусом скругления 5 пикселов), а затем расширить его на каждый стиль, который должен иметь такую границу. Это может показаться хорошей идеей, но быстро приведет к чрезмерному увеличению кода таблиц стилей благодаря групповым селекторам несвязанных элементов.

Рекомендуется расширять элементы со схожим функционалом. В примере заголовка в начале раздела разумно применять директиву `@extend`. Кроме того, если у вас есть конкретный элемент пользовательского интерфейса, например кнопки или диалоговое окно, который обладает различными состояниями, создайте базовый стиль и расширьте его.

ПРИМЕЧАНИЕ

Примеси (будут рассмотрены позже) обеспечивают альтернативу директиве `@extend`. Чтобы узнать ключевые отличия между этими двумя понятиями и использовать их, обратитесь к сайту tinyurl.com/kf3yzfu.

Например, вы можете создать стиль, который будет применяться ко всем кнопкам на вашем сайте, таким как **Заказать**, **Удалить** или **Отмена**. Все они имеют некоторые сходства — шрифт, форму и т. д., но могут обладать уникальными свойствами, например, кнопка **Заказать** имеет зеленый фон, а кнопка **Удалить** — красный. Вы можете легко создать базовый стиль для всех кнопок и расширить его на другие стили с дополнительными свойствами:

```
%btn {
  display: inline-block;
  padding: 1em;
  border-radius: 3px;
}
```

```
.btn-order {
  @extend %btn;
  background-color: green;
  color: white;
}
.btn-delete {
  @extend %btn;
  background-color: red;
  color: white;
}
.btn-cancel {
  @extend %btn;
  background-color: #888;
  color: black;
}
```

Препроцессор Sass преобразует его в следующий код:

```
.btn, .btn-order, .btn-delete, .btn-cancel {
  display: inline-block;
  padding: 1em;
  border-radius: 3px;
}
.btn-order {
  background-color: green;
  color: white;
}
.btn-delete {
  background-color: red;
  color: white;
}
.btn-cancel {
  background-color: #888;
  color: black;
}
```

Использование и расширение заполнителей позволяет легко обновить внешний вид такого элемента интерфейса, как кнопка. В данном примере, если вы хотите удалить скругленные углы всех кнопок, можете удалить свойство `border-radius` селектора-заполнителя, и все кнопки на вашем сайте будут изменены.

Существует много других элементов пользовательского интерфейса, применение которых вкупе с заполнителями и директивой `@extend` приносит пользу. Например, диалоговые окна, предупреждающие пользователя, указывающие об ошибке, подтверждающие заказ или проведение транзакции.

Примеси

Метаязык Sass содержит еще один мощный инструмент под названием *примеси* (также называемый *миксинами*), который позволяет упорядочить CSS-код и сэкономить время. Примеси — это мини-программы, которые выполняют часть работы

за вас (своеобразный аналог макросов в программах Excel или Word). В языке Sass примеси представляют собой ссылки на группу объявлений CSS, которые вы хотели бы использовать многократно. Другими словами, примеси могут написать CSS-код для вас, экономя вам время.

Создание примесей

В главе 17 вы познакомились с методом flexbox-верстки и узнали о том, как можно превратить любой элемент страницы в flex-контейнер, присвоив свойству `display` значение `flex`. Чтобы использовать аналогичный прием в браузере Safari версии 8 и ниже, необходим вендорный префикс, поэтому, чтобы превратить элемент `div` с классом `container` в flex-контейнер, понадобится добавить следующие две строки CSS-кода:

```
.container {
  display: -webkit-flex;
  display: flex;
}
```

Другими словами, вам необходимо написать код, объем которого в два раза больше. Примеси в метаязыке Sass упрощают эту рутинную работу, позволяя писать одну строку кода вместо двух. Чтобы создать примесь, сначала добавьте инструкцию `@mixin`, а затем имя, которое вы хотите присвоить создаваемой примеси, и фигурные скобки:

```
@mixin flex {
}
```

`@mixin` — это ключевое слово в языке Sass, а имя (`flex` в данном примере) предназначено для вас. Имя `flex` подходит идеально, поскольку примесь создаст CSS-код, который, в свою очередь, предназначен для создания flex-контейнера.

Затем внутри фигурных скобок можно добавить код, который препроцессор Sass должен добавить в финальную таблицу стилей. В нашем случае это две почти одинаковые строки CSS-кода:

```
@mixin flex {
  display: -webkit-flex;
  display: flex;
}
```

Примеси — это многократно используемые фрагменты кода. Они сродни функциям в языках программирования, таких как JavaScript: вы можете применять примеси снова и снова, чтобы добавлять CSS-код в финальную каскадную таблицу стилей. Поскольку примеси можно использовать так же, как и переменные Sass, рекомендуется сохранять их в отдельный файл фрагмента Sass с именем `_mixins.scss` и импортировать его в приоритетной очереди в основном Sass-файле (в том, который добавляет все остальные фрагменты). Существует одно правило: примеси рекомендуется импортировать только *после* переменных (поскольку в примесях вы будете часто использовать переменные), но *перед* остальными фрагментами Sass, в которых вы могли бы задействовать примеси. Например:

```
@import 'helpers/variables';
@import 'helpers/mixins';
@import 'base/reset';
@import 'base/grid';
@import 'base/layout';
```

Создав примеси, вы можете применять их в любых стилях.

Использование примесей

Для этого необходимо использовать Sass-директиву `@include`, а после нее указывать имя примеси. Например, чтобы включить элемент страницы с классом `container` в flex-контейнер, вы могли бы задать примесь, подобную следующей:

```
.container {
  @include flex;
}
```

После компиляции кода препроцессором Sass код получившегося CSS-файла будет выглядеть следующим образом:

```
.container {
  display: -webkit-flex;
  display: flex;
}
```

Конечно, после использования примеси вы можете добавить и другие свойства. Например, если вы хотите назначить фоновый цвет и границу контейнеру, добавьте следующий CSS-код после примеси:

```
.container {
  @include flex;
  background-color: #84F;
  border: 1px solid #444;
}
```

После компиляции CSS-код будет выглядеть следующим образом:

```
.container {
  display: -webkit-flex;
  display: flex;
  background-color: #84F;
  border: 1px solid #444;
}
```

Как вы могли заметить, использовать примеси удобно для любых свойств, которые требуют вендорные префиксы, таких как flex-свойств для браузера Safari 8 и ниже, или CSS-анимаций и преобразований, описанных в главе 10.

ПРИМЕЧАНИЕ

Примеси Sass очень полезны, но могут запутать вас. Чтобы получить подробную информацию о примесях Sass, посетите сайт tinycloud.com/perl4h.

Передача данных примесям

Примеси Sass могут также принимать данные и использовать их для управления стилями. Как вы только что видели, примеси очень удобны для свойств, которые содержат вендорные префиксы. Свойство `transform` требует наличия двух вендорных префиксов, если нужно обеспечить поддержку в браузерах Safari 8 и Internet Explorer 9. Например, чтобы повернуть элемент на 10° , вам понадобятся три строки CSS-кода:

```
-webkit-transform: rotate(10deg);  
-ms-transform: rotate(10deg);  
transform: rotate(10deg);
```

Этот пример прекрасно подходит для использования примеси, но, обратите внимание, угол поворота весьма специфичен — 10° . Вам может потребоваться повернуть отдельный элемент на 5, 45 или даже -30° ; точные значения, как правило, различаются. Например, было бы здорово, если бы после вложения примеси можно было указать, на сколько градусов элемент должен повернуться.

К счастью, вы можете передать в примесь данные, которые она будет использовать при выводе CSS-кода. В данном случае вы можете создать примесь, которая принимает значение в градусах. Начнем с директивы `@mixin`, имени примеси, пары скобок с именем переменной внутри них и фигурных скобок:

```
@mixin rotate($deg) {  
  
}
```

Запись `$deg` в приведенном коде выглядит как переменная Sass, которые мы обсуждали ранее в этой главе. Она начинается с символа доллара, после которого следует еще несколько символов. Здесь запись `$deg` внутри скобок называется *параметром* — это переменная, но она пуста, пока вы не используете примесь и не *передаете* ей (то есть присвоите) значение. Через минуту вы увидите, как это сделать, но сначала нужно завершить код примеси, добавив следующий Sass-код:

```
@mixin rotate($deg) {  
  -webkit-transform: rotate($deg);  
  -ms-transform: rotate($deg);  
  transform: rotate($deg);  
}
```

Обратите внимание, что параметр `$deg` используется так, как если бы речь шла об обычной переменной Sass. В этом случае вы применяете его для указания определенного угла поворота. Вы добавите код, приведенный выше, в файл фрагмента `_mixins.scss`, а затем используете (или *вызовете*) примесь в стиле. Допустим, вы хотели бы повернуть каждый элемент `h1` на 3° , а каждый контейнер `div` — на 7° . Вы можете выполнить это с помощью следующего стиля:

```
h1 {  
  @include rotate(3deg);  
}
```

```
div {
  @include rotate(7deg);
}
```

В результате компиляции итоговый CSS-код будет выглядеть следующим образом:

```
h1 {
  -webkit-transform: rotate(3deg);
  -ms-transform: rotate(3deg);
  transform: rotate(3deg);
}
div {
  -webkit-transform: rotate(7deg);
  -ms-transform: rotate(7deg);
  transform: rotate(7deg);
}
```

В КУРС ДЕЛА!

Не изобретайте примеси

Примеси Sass экономят время и укорачивают CSS-код. Почему бы не сэкономить еще больше времени и не использовать примеси, уже написанные другими людьми? Самые сложные из них очень похожи на настоящие компьютерные программы и могут использовать переменные, циклы, условные операторы и выполнять некоторые, казалось бы, удивительные вещи.

К счастью, существует много библиотек с примесями Sass.

- **Bourbon** (bourbon.io) — небольшая библиотека примесей, которая добавляет вендорные префиксы и выполняет другие интересные действия для многих CSS-свойств, таких как анимации, переходы, `font-face` и т. д.
- **Neat** (neat.bourbon.io) — библиотека примесей, которая упрощает создание CSS-сеток, разработанная создателями библиотеки Bourbon.
- **Susy** (susy.oddbird.net) — еще один набор примесей для создания CSS-сеток. Эта библиотека очень популярна среди пользователей Sass.
- **Breakpoint** (breakpoint-sass.com) — библиотека примесей, созданная с одной только целью — упростить и ускорить написание медиазапросов.
- **Compass** (compass-style.org) — нечто большее, чем просто библиотека примесей. Она содержит полезные примеси, а также помогает создавать CSS-спрайты. Библиотека Compass настолько велика и предлагает так много возможностей, что создатели называют ее *фреймворком для CSS-верстки*.

Добавление переменных в текст

В предыдущем разделе вы создали примесь `rotate`. Каждый раз, когда вы захотите повернуть элемент, вы просто используете примесь и передаете ему значение, такое как `5deg`, `7deg` или `-45deg`. Поскольку примеси призваны сократить объемы набираемого кода, было бы удобнее указать просто число: например, вместо значения `5deg` ввести число `5`. В конце концов, вы всегда можете добавить запись `deg` в конечном CSS-коде, так почему бы не позволить препроцессору Sass сделать это за вас?

Другими словами, вы могли бы описать вращение следующим образом:

```
h1 {
  @include rotate(3);
}
```

Затем добавьте единицу измерения deg к результату работы примеси. Можно использовать что-то вроде следующего кода:

```
@mixin rotate($deg) {
  -webkit-transform: rotate($deg deg);
  -ms-transform: rotate($deg deg);
  transform: rotate($deg deg);
}
```

К сожалению, препроцессор Sass поставит пробел между значением, которое вы введете, и единицей измерения deg. Результат будет выглядеть примерно так:

```
h1 {
  -webkit-transform: rotate(3 deg);
  -ms-transform: rotate(3 deg);
  transform: rotate(3 deg);
}
```

Пробела в этой записи быть не должно, иначе браузеры не смогут повернуть элемент. Но вы не можете просто исключить пробел из примеси:

```
@mixin rotate($deg) {
  -webkit-transform: rotate($degdeg);
  -ms-transform: rotate($degdeg);
  transform: rotate($degdeg);
}
```

Если вы сделаете это, препроцессор Sass не создаст CSS-код. Вместо этого он отобразит ошибку Undefined variable: "\$degdeg". Это обусловлено тем, что без пробела препроцессор будет искать переменную \$degdeg, которой на самом деле не существует.

Чтобы объединять переменные Sass с текстом, вы должны использовать специальные символы, которые сообщают препроцессору Sass, где находится действительная переменная. Эта техника называется *интерполяцией*, и смысл ее заключается в фразе: «Эй, Sass, вот переменная; используй ее значение». Для того чтобы сделать это, вы должны обернуть переменную некоторыми специальными символами, начиная с #, после чего следует имя переменной и фигурная скобка закрывается — }. Например, чтобы приведенный выше пример стал работоспособным, препроцессор Sass должен интерполировать переменную \$deg, поэтому вы должны использовать следующий код:

```
@mixin rotate($deg) {
  -webkit-transform: rotate(#{ $deg }deg);
  -ms-transform: rotate(#{ $deg }deg);
  transform: rotate(#{ $deg }deg);
}
```

Теперь вы можете добавить примесь, показанную выше, отправить числовое значение, и препроцессор Sass корректно подключит число с текстом deg без слияния, поэтому запись `@include rotate(3);` создаст код:

```
-webkit-transform: rotate(3deg);  
-ms-transform: rotate(3deg);  
transform: rotate(3deg);
```

Передача дополнительных данных и значений примесям

Примеси используются не только для добавления вендорных префиксов или значений. Они могут быть полезны, если необходимо писать аналогичный CSS-код снова и снова. Например, при назначении типографических свойств заголовкам, абзацам, информации об авторских правах или любой части текста обычно используются следующие значения: `font-size`, `line-height`, `font-weight` и `color`.

Вы можете создать примесь, которая позволяет из одной строки Sass-кода получить четыре строки CSS-кода. Хитрость заключается в том, чтобы передать всю необходимую информацию примеси. К счастью, препроцессор Sass позволяет передать более чем одну часть информации (называемую *аргументом*) примеси. Укажите имя каждой переменной (или *параметра*) в круглых скобках через запятую после имени примеси. Например, чтобы создать примесь, которая принимает четыре типографических свойства и создает четыре строки CSS-кода, вы могли бы использовать следующий код:

```
@mixin text($size, $line-height, $weight, $color) {  
  font-size: $size;  
  line-height: $line-height;  
  font-weight: $weight;  
  color: $color;  
}
```

В этом примере добавляется примесь с именем `text`. Чтобы использовать примесь, нужно вложить ее и ввести четыре значения:

```
h1 {  
  @include text(1.25em, 1.2, bold, #FF0000);  
}
```

CSS-код, созданный препроцессором Sass, будет выглядеть следующим образом:

```
h1 {  
  font-size: 2em;  
  line-height: 1.2;  
  font-weight: bold;  
  color: #FF0000;  
}
```

Примеси могут действительно сэкономить ваше время и уменьшить объем набираемого кода. Но как поступить, если в этом примере вы не хотите передавать цвет или вес шрифта? Все, что вам нужно сделать, — только указать размера шриф-

та и высоту строки. В этом случае можно было бы просто не вводить дополнительные свойства:

```
h1 {
  @include text(1.25em, 1.2);
}
```

Поскольку эта примесь ожидает получить четыре значения, а вы ввели только два, препроцессор Sass отобразит ошибку и Sass-файлы не будут скомпилированы в CSS-файл. К счастью, вы можете сообщить препроцессору Sass, что значения примеси опциональны. Допустим, вы хотите сделать свойства `line-height`, `font-weight` и `color` опциональными. Для этого необходимо с помощью примеси присвоить значение `null` каждому из этих параметров:

```
@mixin text($size, $line-height: null, $weight: null, $color: null) {
  font-size: $size;
  line-height: $line-height;
  font-weight: $weight;
  color: $color;
}
```

Значение `null` сообщает препроцессору Sass, что эти значения могут быть пусты. Другими словами, используя примесь, вы можете не указывать эти значения и все эти стили будут компилироваться без ошибок:

```
h1 {
  @include text(2em);
}
h2 {
  @include text(1.25em, 1.2);
}
p {
  @include text(1em, 1.2, normal);
}
```

Кроме того, вместо значения `null` для параметров примеси вы можете задавать значения по умолчанию, и препроцессор Sass будет их использовать, если вы не укажете иные. Например, требуется, чтобы стандартное значение параметра `line-height` для большинства элементов было равно `1.2`, а свойства `weight` — `normal`. В примеси вы можете назначить эти значения по умолчанию:

```
@mixin text($size, $line-height: 1.2, $weight: normal, $color: null) {
  font-size: $size;
  line-height: $line-height;
  font-weight: $weight;
  color: $color;
}
```

Теперь, если вы укажете размер шрифта при использовании примеси, вы получите код, похожий на следующий:

```
h1 {
  @include text(2em);
}
```

Препроцессор Sass создаст CSS-код, используя значения по умолчанию для любых свойств, которые вы опустите:

```
h1 {  
  font-size: 2em;  
  line-height: 1.2;  
  font-weight: normal;  
}
```

Чтобы применить несколько параметров, осталось выполнить одну небольшую доработку. Как поступить, если вы хотите указать размер и цвет шрифта, но не изменять два других параметра? Когда вы вкладываете примесь в стиль, препроцессор Sass ожидает значения для примеси в том же порядке, в котором они перечислены в ней. Например, взгляните на пример использования примеси, представленной выше, для правила h2:

```
h2 {  
  @include text(2em, red);  
}
```

Поскольку препроцессор Sass ожидает второй параметр, который должен содержать значение свойства `line-height`, она попытается присвоить значение `red` свойству `line-height`. Этот CSS-код ошибочен, ведь вы, вероятно, не этого хотите. Тем не менее вы можете явно указать препроцессору Sass, какие значения хотите назначить определенным параметрам. Для этого укажите имя параметра в примеси, затем двоеточие, а после него — значение, которое нужно присвоить параметру:

```
h2 {  
  @include text(2em, $color:red);  
}
```

Примеси Sass могут сэкономить уйму времени, поэтому я рекомендую потратить время на их изучение.

Использование медиазапросов

Как вы узнали из главы 15, медиазапросы важны при создании резиновых дизайнов, которые хорошо выглядят и функционируют на экранах различных устройств. Медиазапросы позволяют сделать шрифт крупнее для экранов с высоким разрешением или устранить нежелательные поля и отступы на маленьких экранах телефонов. Метаязык Sass обеспечивает встроенную поддержку медиазапросов в рамках селекторов. Например, может потребоваться, чтобы шрифт всех заголовков `h1` по умолчанию имел размер `2em`, но увеличивался до `3em`, когда экранное разрешение по горизонтали превышает `1200` пикселей. С помощью Sass этого можно добиться следующим способом:

```
h1 {  
  font-size: 2em;  
  @media (min-width: 1200px) {
```

```
    font-size: 3em;
  }
}
```

Это немного похоже на вложение селекторов. Однако финальный CSS-код выглядит совершенно иначе. В этом случае препроцессор Sass создает один стиль `h1`, а затем генерирует медиазапрос с другим стилем `h1`. Финальный CSS-код выглядит следующим образом:

```
h1 {
  font-size: 2em;
}
@media (min-width: 1200px) {
  h1 {
    font-size: 3em;
  }
}
```

Преимущество этого подхода заключается в том, что вы можете сгруппировать все свои стили `h1` в одном месте. В разделе «Медиазапросы» главы 15 говорится, что, как правило, при работе с медиазапросами, таблицу стилей приходится разделять на несколько медиазапросов и дублировать селекторы в каждом из них. Так, в зависимости от того, сколько медиазапросов вы добавили в CSS-файл, селекторы `h1` могут встречаться в двух, трех или даже пяти разных частях файла. Используя Sass, вы можете группировать все медиазапросы с селектором и таким образом видеть, как форматируется элемент `h1` в той или иной точке останова.

ПРИМЕЧАНИЕ

Недостатком такого подхода является то, что препроцессор Sass создает отдельные медиазапросы для каждого селектора, вследствие чего таблица стилей может быть загромождена дополнительным кодом медиазапросов. На этот счет ведется много дискуссий, но многие дизайнеры считают, что подход препроцессора Sass упрощает процесс управления стилями и на самом деле добавляет не так уж и много кода. Хорошая статья, посвященная преимуществам и недостаткам строчных медиазапросов (которые использовались в данном примере), доступна по адресу tinyurl.com/hkwmyfq.

Примеси медиазапросов

В то время как вложение медиазапросов в селектор помогает сгруппировать свойства элемента в одном месте Sass-файла, использование примесей повышает их полезность. Как вы можете увидеть в коде на предыдущей странице, описание медиазапроса присутствует непосредственно в коде: `min-width: 1200px`. Но как поступить, если вы решите, что 1200 пикселей — слишком большое число или слишком малое? Если вы добавили запрос к сотням других элементов страницы, то теперь придется выполнить множество обновлений в Sass-файлах.

Вы можете объединить несколько методов Sass — переменные, примеси и строчные медиазапросы, — чтобы создать гибкий настраиваемый метод для добавления медиазапросов к стилям. Существует много различных способов создания примесей медиазапросов в Sass и не меньше способов использования медиазапросов.

В этом разделе вы увидите одну технику, основанную на стратегии Mobile First. Ее суть заключается в том, чтобы начать со стилей, которые применимы при любой ширине окна браузера. То есть основные стили (без медиазапросов) представляют версию для мобильных устройств, а также устанавливают общие свойства, распространяющиеся на экраны с любым экранным разрешением по горизонтали. К этим свойствам, к примеру, относятся шрифты, используемые для заголовков и абзацев текста или цвета, применяемые в качестве фона.

Затем добавляются медиазапросы для различных точек останова, в которых изменяются предыдущие стили, подстраиваясь под различные размеры окна браузера/разрешение экрана. Например, по мере того, как окно браузера становится шире, можно использовать точки останова, чтобы изменить размер заголовков или добавить пространство по периметру фотографии. Другими словами, каждый медиазапрос изменяет оригинальную конструкцию, добавляя правила, которые улучшают внешний вид страницы при новой ширине окна браузера/разрешении экрана.

В этом примере вы будете использовать три точки останова: для экрана с малым, средним и высоким экранным разрешением по горизонтали. Первая из них предназначена для таких устройств, как смартфоны с экраном с большой диагональю и высоким разрешением, например Samsung Galaxy S6. Вторая точка останова предназначена для большинства планшетов, а третья будет работать с компьютерными мониторами и экранами ноутбуков. Кроме того, вы создадите медиазапросы, которые работают *только* в определенных точках останова, и те, которые применяются при конкретной точке останова и выше.

Чтобы сделать это, вы создадите переменные Sass для точек останова, создадите примеси для генерации медиазапросов на основе этих точек останова и используете примеси в коде с помощью директивы @include.

Установка точек останова переменных

Ваш дизайн может адаптироваться по-разному для экранов с разным разрешением по горизонтали. Например, четыре колонки могут превращаться в три, если разрешение экрана по горизонтали менее 760 пикселей. Или если менее 780. Для начала необходимо создать переменные Sass и определить их значения, которые позже при необходимости можно будет изменить:

```
$screen-small : 400px;  
$screen-medium : 760px;  
$screen-larger: 1000px;
```

Создание примесей медиазапросов

Примеси будут генерировать необходимый код медиазапроса с использованием переменных для точек останова. Чтобы обеспечить большую гибкость, вы можете создать два различных медиазапроса для первой и второй точек останова. Первый медиазапрос будет применяться, если разрешение экрана по горизонтали *не ниже* величины, указанной в точке останова, а также выше, чем значение в следующей точке останова. Вы будете использовать этот медиазапрос, если понадобится, к при-

меру, добавить стиль, который будет работать как на планшетах, так и на экранах компьютерных мониторов.

Код примеси для этого медиазапроса будет выглядеть следующим образом:

```
@mixin mq-small-up {  
  @media (min-width: $screen-small) {  
    @content;  
  }  
}
```

Обратите внимание, что появилась новая директива Sass — @content. Она предоставляет код, который помещается внутри фигурных скобок после директивы @include. Например, чтобы использовать примесь, показанную выше, для увеличения размера шрифта текста абзацев, когда разрешение экрана по горизонтали выше, чем указано в наименьшей точке останова, вы могли бы добавить следующий код:

```
p {  
  font-size: 1.5em;  
  @include mq-small-up {  
    font-size: 1.75em;  
    margin-top: 10px;  
  }  
}
```

В результате работы примеси код, заключенный в фигурные скобки (font-size: 1.75em; margin-top: 10px;), заменит директиву @content. CSS-код, скомпилированный препроцессором Sass из приведенного выше примера, выглядит следующим образом:

```
p {  
  font-size: 1.5em;  
}  
@media (min-width: 500px) {  
  p {  
    font-size: 1.75em;  
    margin-top: 10px;  
  }  
}
```

Второй медиазапрос для первой точки останова применяется *только* тогда, когда разрешение экрана по горизонтали выше, чем указано в этой точке, а также ниже, чем в следующей точке останова. Таким образом, вы можете создавать стили, которые работают только в определенном диапазоне размеров экрана.

Примесь «только точка останова» для первой точки останова будет выглядеть следующим образом:

```
@mixin mq-small {  
  @media (min-width: $screen-small) and (max-width: $screen-medium - 1px) {  
    @content;  
  }  
}
```

Этот код будет генерировать медиазапрос, который выглядит следующим образом:

```
@media (min-width: 400px) and (max-width: 759px) {
}
```

Обратите внимание, что в данном случае присутствует математическое действие: `$screen-medium - 1px`. Препроцессор Sass запрограммирован на выполнение основных математических операций (см. врезку в разделе «Переменные Saas»). В этом случае он вычитает 1 пиксел из значения переменной `$screen-medium`, генерируя значение 759px. Поскольку этот медиазапрос не должен применяться при следующей точке останова (760px), значение свойства `max-width` должно быть на 1 пиксел меньше, чем указано в ней.

Две похожие примеси созданы для двух первых точек останова, поэтому, придерживаясь примера трех точек останова, вы могли бы использовать следующие три примеси:

```
@mixin mq-small {
  @media (min-width: $screen-small) and (max-width: $screen-medium - 1px) {
    @content;
  }
}
@mixin mq-small-up {
  @media (min-width: $screen-small) {
    @content;
  }
}
@mixin mq-medium {
  @media (min-width: $screen-medium) and (max-width: $screen-large - 1px) {
    @content;
  }
}
@mixin mq-medium-up {
  @media (min-width: $screen-medium) {
    @content;
  }
}
@mixin mq-large-up {
  @media (min-width: $screen-large) {
    @content;
  }
}
```

Использование примесей медиазапросов

Наконец, добавив переменные и примеси с помощью директивы `@include`, вы можете начать использовать их в своих стилях. Предположим, вы хотите создать отступы внутри боковой панели, увеличивая ее при достижении каждой точки останова и добавляя пространство на экранах с высоким разрешением. Кроме того, вы

хотите, чтобы ширина верхнего и нижнего поля боковой панели была равна 0 для первой точки останова, а затем, когда разрешение экрана по горизонтали достигнет значения второй точки останова, увеличивалась до 20 пикселей. Таким образом, если боковой панели присвоен класс с именем `sidebar`, вы можете использовать следующий Sass-код для достижения цели:

```
.sidebar {
  padding: 0px;
  margin: 0px;
  @include mq-small {
    padding: 3px;
  }
  @include mq-medium {
    padding: 10px;
  }
  @include mq-medium-up {
    margin: 10px 0;
  }
  @include mq-large-up {
    padding: 15px;
  }
}
```

Хотя этот код далек от тех традиционных медиазапросов, которые мы рассматривали в главе 15, как только вы привыкнете к данному подходу, обнаружите преимущества от лаконичности создаваемого кода. А также увидите, что хранение всех медиазапросов рядом с их элементами упрощает обновление CSS-кода.

ПРИМЕЧАНИЕ

Существует много других способов, с помощью которых использование медиазапросов в Sass можно упростить. О них вы можете прочитать в следующих статьях: tinyurl.com/ztcn289, tinyurl.com/pqwj7tr, tinyurl.com/7ntbgl3 и tinyurl.com/18r.

Поиск и устранение ошибок с помощью карт CSS-кода

Как вы узнали из глав 5 и 6, каскадные таблицы стилей могут взаимодействовать различными необычными и нежелательными способами. Врезка «Инструменты в помощь» в разделе «Особенности каскадности: какие стили имеют преимущество» главы 5 рассказывает об одном решении, которое позволяет определить, как различные стили влияют на элемент страницы. Большинство браузеров позволяют инспектировать элементы страницы. Рассмотрим пример работы в браузере Firefox. Для анализа элемента необходимо щелкнуть правой кнопкой мыши на элементе на странице и выбрать пункт **Исследовать элемент** (Inspect Element) (выделен на рис. 19.6) в открывшемся контекстном меню. Откроется панель (обычно в нижней части веб-страницы), отображающая исходный код страницы с выбранным HTML-элементом. HTML-код обычно отображается в левой области, а CSS-стили — в правой (см. рис. 19.6, *внизу*).

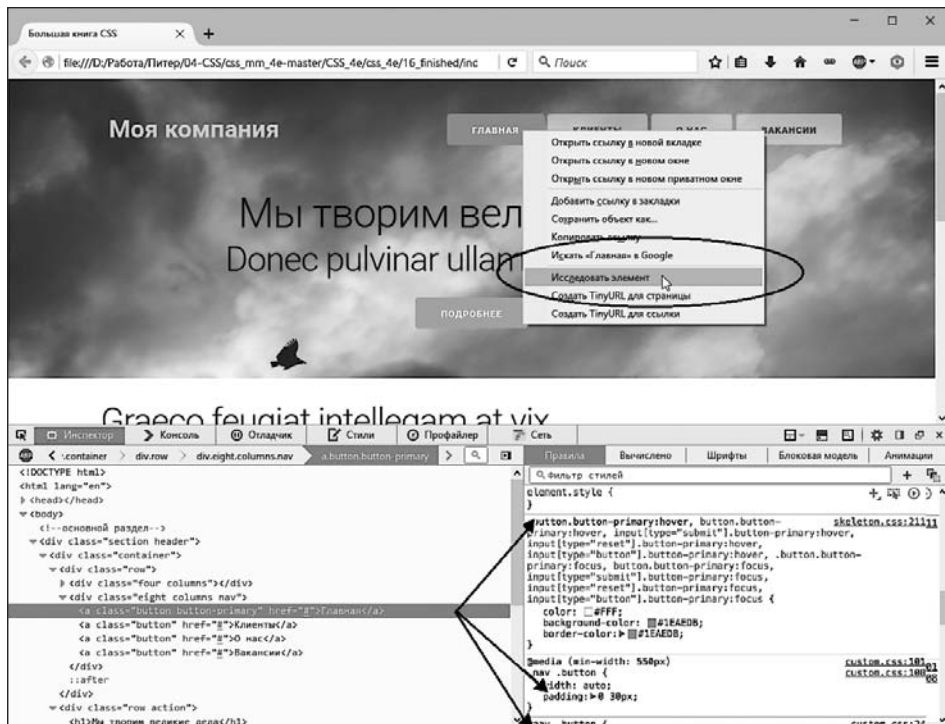


Рис. 19.6. Используя инструмент исследования элемента в браузере, вы можете узнать много нового о том, какой CSS-код применяется для форматирования конкретного элемента страницы

Правая область отображает все стили (в том числе и медиазапросы), которые влияют на текущий элемент. С ее помощью вы можете увидеть, есть ли какой-либо стиль, который делает то, что вам не нужно, а затем исправить его в редакторе HTML-кода. Кроме того, в правой области перечислены все стили, которые содержит таблица стилей. Например, на рис. 19.6 первый стиль (начиная с `.button.button-primary: hover`) находится в таблице стилей `skeleton.css`, в то время как следующие два стиля — в файле `custom.css`. Эта информация позволяет узнать, к какому файлу обратиться в тех случаях, когда вы хотите изменить стиль.

Однако при использовании Sass таблицы стилей, которые браузер применяет для форматирования HTML-кода, не являются теми же файлами, в которых вы добавляете и редактируете свои стили. Знать, что цвет фона кнопки определяется файлом `styles.css`, не принесет особой пользы, поскольку на самом деле, чтобы изменить этот цвет, вам необходимо перейти к файлу фрагмента `_buttons.scss`. К счастью, большинство браузеров поддерживают специальную функцию «карты кода», которые могут помочь вам найти необходимый Sass-файл.

Карта кода — это схема, которой может следовать браузер, чтобы перейти от стиля к его источнику. То есть она сообщает браузеру, как от стиля в конечном CSS-файле перейти к исходному стилю в Sass-файле. Препроцессор Sass автоматически генерирует карты кода, поэтому, когда вы выполняете команду `sass --watch`, она создает как конечный CSS-файл, так и его карту кода.

ПРИМЕЧАНИЕ

В старых версиях Sass приходилось включать функцию формирования карт кода вручную с помощью следующего кода:

```
sass --watch --sourcemap sass:scss
```

Файлы карты кода имеют расширение `.map`, например `styles.css.map`. Если вы не видите файла с таким расширением в папке, в которой препроцессор Sass сохраняет финальный скомпилированный CSS-файл, попробуйте выполнить команду `sass` с параметром `sourcemap`, как показано выше.

Браузеры Internet Explorer 11, Firefox, Chrome, Safari поддерживают карты кода, поэтому исследование элементов страницы, использующей CSS-файл, созданный с помощью Sass, на самом деле не сильно отличается от анализа обычного CSS-файла. Щелкните правой кнопкой мыши на элементе, который вы хотите проверить, а затем выберите соответствующий элемент в открывшемся контекстном меню. В окне вашего браузера откроется та же панель инспектора элементов. Тем не менее теперь вы увидите имена файлов фрагментов Sass, содержащих правила, участвующих в форматировании элемента (выделено на рис. 19.7).

На рис. 19.7 вы можете увидеть, что медиазапрос был создан примесью из файла `_mixins.scss`, а фактический стиль заголовка находится в файле `_layout.scss`. Чтобы изменить этот медиазапрос, необходимо открыть файл `_mixins.scss`; чтобы изменить значение свойства `font-size` заголовка или добавить дополнительные стили, необходимо открыть и изменить файл `_layout.scss`.

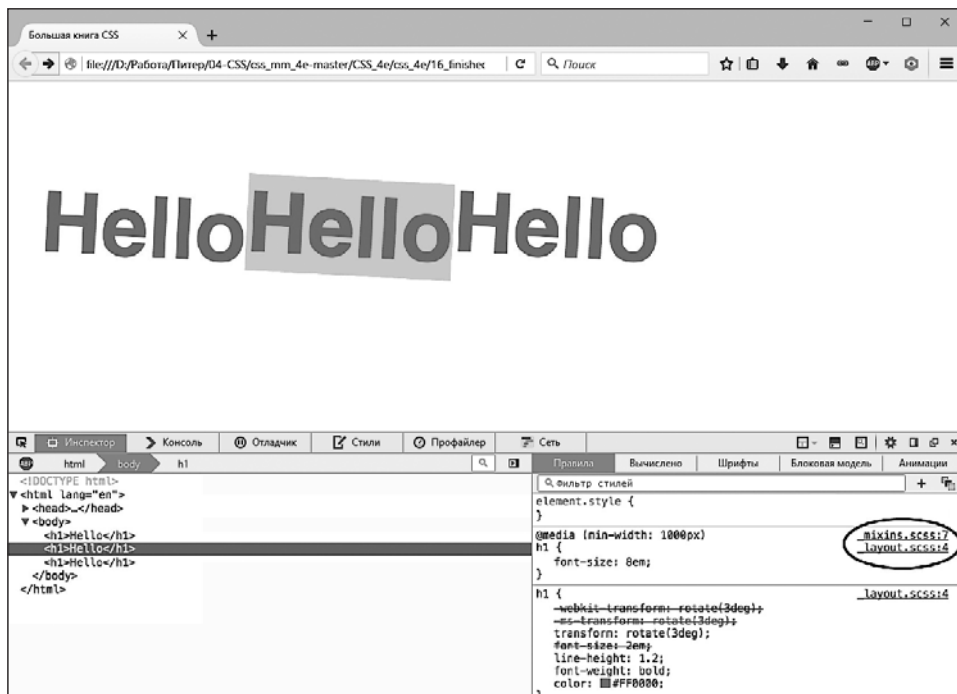


Рис. 19.7. Карта CSS-кода позволяет легко найти файлы фрагментов, содержащих правила Sass, которые форматировуют элементы на веб-странице

Метаязык Sass открывает много возможностей для веб-дизайнера. Он позволяет легко организовать CSS-код в небольшие, легко редактируемые файлы гораздо меньших размеров. Кроме того, Sass позволяет автоматически добавлять вендорные префиксы. И в целом он дает возможность выполнять CSS-верстку быстрее и эффективнее.

ПРИМЕЧАНИЕ

Чтобы создать сжатый финальный CSS-файл, см. совет в конце раздела «Основы Sass».

ЧАСТЬ V

Приложения

Приложение 1. Справочник свойств CSS

Приложение 2. Информационные ресурсы, посвященные CSS

Приложение 1.

Справочник свойств CSS

Владение на профессиональном уровне каскадными таблицами стилей (CSS) означает знание того, как использовать огромное количество свойств CSS, которые управляют внешним видом текста, изображений, таблиц и форм. Чтобы помочь вам в ваших поисках, я собрал в этом приложении свойства и их значения, которые вы будете использовать для создания собственных стилей. Этот список охватывает практически все стандартные свойства CSS 2.1 — те, которые поддерживает большинство браузеров, а также ряд наиболее полезных и поддерживаемых браузерами свойств CSS3.

ПРИМЕЧАНИЕ

Самая последняя спецификация CSS имеет довольно большой объем. Фактически, чтобы справиться с разрастанием CSS, Консорциум W3C разбил спецификацию CSS на множество модулей — каждый из них описывает конкретное свойство или набор родственных свойств. Чтобы получить полную информацию из уст Консорциума W3C, посетите страницу по адресу tinyurl.com/n8xhw. Полный список свойств каскадных таблиц стилей опубликован на сайте tinyurl.com/ke6h5dt.

Значения свойств CSS

У каждого свойства каскадных таблиц стилей есть соответствующее ему значение. Свойство `color`, которое определяет цвет шрифта, требует присвоения значения, позволяющего указать, какой цвет вы хотите использовать. К примеру, свойство `color: #FFFFFF`; задает белый цвет. Разные свойства требуют указания различных значений, каждое из которых относится к одной из четырех основных категорий: цвета, размеры, ключевые слова и URL-адреса.

Цвета

Вы можете назначать цвета многим характеристикам элемента, включая шрифт, фон и границы. Каскадные таблицы стилей предоставляют несколько различных способов указания цвета.

Имена

Ключевое слово, обозначающее цвет, — это имя цвета, например `white` или `black` (белый или черный соответственно). В настоящее время существует 17 одобренных имен, обозначающих цвета: `aqua`, `black`, `blue`, `fuchsia`, `gray`, `green`, `lime`, `maroon`, `navy`, `olive`, `orange`, `purple`, `red`, `silver`, `teal`, `white` и `yellow`. Некоторые браузеры поддержи-

вают больше имен, а спецификация CSS3 допускает в будущем применение еще большего количества цветов. См., например, рассмотренную в главе 6 цветовую модель RGBA. Подробнее о ключевых словах можно прочитать по адресу tinyurl.com/p2ga5.

Значения по модели RGB

Мониторы компьютеров создают оттенки, используя красный, зеленый и синий цвета. Эти RGB-значения могут создать почти весь спектр цветов. Практически любой графический редактор, редактор HTML-файлов или программа предпечатной подготовки позволяют определить цвета, используя модель RGB, поэтому легко можно выбрать цвет в одной из таких программ и указать соответствующее значение в свойстве CSS. Каскадные таблицы стилей позволяют указывать значения цвета по модели RGB несколькими способами.

- **Шестнадцатеричные значения.** Этот метод наиболее часто используется во Всемирной паутине для определения цвета. Шестнадцатеричные значения цветов состоят из трех двузначных чисел в шестнадцатеричной (то есть с основанием 16) системе счисления. #FF0033 представляет RGB-значение, составленное из красного (FF, что равняется 255 в десятичной системе счисления), зеленого (00) и синего (33). Символ # указывает CSS, что впереди следует ожидать шестнадцатеричное число, и является обязательным. Если вы пропустите символ #, браузер не отобразит нужный цвет.

СОВЕТ

Если во всех трех значениях цифры повторяются, можно сократить шестнадцатеричное значение, используя только первое число каждой пары. Например, #361 означает то же самое, что и #336611.

- **Проценты.** Вы также можете указать цвет, используя значения в процентах, например `rgb(100%, 0%, 33%)`. Вы можете узнать эти значения в графическом редакторе или программе предпечатной подготовки, которые позволяют указывать цвета, используя проценты.
- **Целочисленные значения.** И наконец, для назначения цвета по модели RGB можно использовать целочисленные значения. Формат такой же, что и для процентных значений, но для указания каждого цвета используется число от 0 до 255: `rgb(255, 0, 33)`.
- **RGBA.** Модель RGBA добавляет к смешиваемым цветам уровень прозрачности. То есть можно назначить цвет, сквозь который просматривается расположенный позади фоновый цвет, изображение или текст. В конце значения свойства добавляется значение в диапазоне от 0 (полная прозрачность) до 1 (полная непрозрачность). Для задания насыщенности компонентов цвета можно указывать либо процентные, либо десятичные значения, а для прозрачности можно использовать только десятичные значения. Например, оба следующих объявления создают светло-серый цвет с 50%-ной прозрачностью:

```
rgba(50%,50%,50%,.5)
rgba(122,122,122,.5)
```

- **HSL.** Аббревиатура HSL расшифровывается как Hue — «тон», Saturation — «насыщенность» и Lightness — «светлота» (иногда также используется термин *luminance* — «яркость»). Это еще один способ задания цвета. Он не поддерживается в браузере Internet Explorer 8 и более ранних версиях, но работает во всех остальных браузерах. Если вы привыкли использовать RGB- или шестнадцатеричные значения цвета, синтаксис HSL может показаться немного непривычным. Вот так выглядит значение свойства, назначающего ярко-красный цвет:

```
hsl(0, 100%, 50%);
```

Первое число является значением угла в диапазоне от 0 до 360° на цветовом круге. Если вы помните очередность следования цветов в радуге — красный, оранжевый, желтый, зеленый, голубой, синий и фиолетовый, то поймете основную идею значения, через которое выражен цвет. Красный — это 0 (это также и 360, поскольку это один полный оборот по кругу), желтый — это приблизительно 50, оранжевый — приблизительно 100, зеленый — 150 и т. д. Каждый цвет занимает примерно 51°.

Вторым значением указывается насыщенность, или то, насколько чистым является оттенок цвета. Насыщенность указывается в диапазоне значений от 0% до 100%. Значение 0% означает полное отсутствие насыщенности, или тусклый серый оттенок. Фактически неважно, какой тон задан, 0% всегда даст один и тот же серый тон. Значение 100% задает чистый цвет, яркий и живой. Третье значение определяет степень светлоты, указанную в процентах от 0% (полностью черный) до 100% (полностью белый). Если нужно получить чистый цвет, следует воспользоваться значением 50%.

Как и у RGB-цвета, существует версия HSL, поддерживающая прозрачность. Она называется HSLA: `hsla(0, 100%, 50%, .5)`.

Неважно, какой метод вы используете, — все они допустимы. Для согласованности вы должны выбрать один определенный способ обозначения цвета и придерживаться этого способа. Но проще и удобнее, может быть, применять имена цветов, например `white` и `black`. В операционных системах Windows и OS X доступны инструменты выбора цвета, которые позволяют подобрать идеально подходящий цвет из всей палитры цветов, а также отображают его RGB-значение. В качестве альтернативы можно использовать инструмент выбора цвета на сайте tinyurl.com/zn2d или более совершенное средство управления палитрами цветов на сайте kuler.adobe.com.

Размеры

Каскадные таблицы стилей предоставляют множество различных способов указать размер шрифта, ширину поля или толщину границы. Чтобы указать размер шрифта, вы можете использовать дюймы, цитеро, точки, сантиметры, миллиметры, единицы измерения `em` и `ex`, пиксели и проценты.

Однако при всем многообразии единиц измерения многие из них не относятся к миру экранного отображения по причинам, обсуждаемым в разделе «Изменение размера шрифта» главы 6. На самом деле вам стоит задуматься только о трех: пикселах, `em` и процентах.

Пикселы

Пиксел — это одна точка на экране компьютера. Пикселы предоставляют последовательный метод обозначения размеров от компьютера к компьютеру: 72 пиксела на одном мониторе составляют 72 пиксела на другом мониторе. Тем не менее это не означает, что фактический размер в реальном мире будет идентичен. Поскольку люди устанавливают для своих мониторов различные разрешения — 800×600 , 1024×768 , 1600×1200 или еще какое-нибудь, 72 пиксела могут выглядеть как 5 сантиметров на одном мониторе и всего 3 сантиметра — на другом. Однако пикселы предоставляют наиболее последовательный контроль над отображением контента.

ПРИМЕЧАНИЕ

Есть только один недостаток в использовании пикселов: люди, которые пользуются браузером Internet Explorer версии 6 или ниже, не могут изменить размеры элемента, если они заданы в пикселах. Если ваш текст окажется слишком мелким, то посетитель будет не в состоянии увеличить его, чтобы сделать более приемлемым для чтения (см. раздел «Изменение размера шрифта» главы 6).

Единица em

В типографике em — это единица измерения, которая представляет высоту прописной буквы «М» определенного шрифта. В веб-дизайне 1 em — это высота базового шрифта в браузере, которая обычно составляет 16 пикселов. Однако пользователь может изменить базовое значение размера, поэтому 1 em может равняться 16 пикселям в одном браузере и 24 пикселям — в другом. Другими словами, em — это относительная единица измерения.

Вдобавок к исходной установке размера шрифта в браузере em может унаследовать информацию о размере от элементов. Размер .9em установит высоту шрифта текста приблизительно равной 14 пикселям в большинстве браузеров с базовым размером 16 пикселов. Но если у вас есть элемент p с размером шрифта .9em, а также элемент strong с размером шрифта .9em внутри этого абзаца p, то размер текста strong составит не 14, а 12 пикселов ($16 \times 0,9 \times 0,9$). Поэтому не забывайте о наследовании, если используете значения в единицах em.

Проценты

Каскадные таблицы стилей используют проценты в *различных* целях, например при установке размера шрифта, определении ширины или высоты элемента, позиционирования фонового изображения и т. д. Для размеров шрифта проценты вычисляются, основываясь на унаследованном значении текста. Скажем, общий размер шрифта абзаца — 16 пикселов. Если вы создали стиль для отдельного абзаца и установили размер его шрифта равным 200 %, то этот текст отображается высотой 32 пиксела. Однако при применении к ширине проценты вычисляются на основе ширины страницы или другого родительского элемента с заданной шириной. Процентное значение указывается в виде числа и следующего за ним символа процентов: 100%.

Единица rem

Единица измерения rem является относительной величиной и зависит от размера шрифта корневого элемента — размера шрифта, применяемого вами для элемента html. В документе это значение остается неизменным, поэтому, в отличие от еди-

ницы em, значение rem не меняется в зависимости от величины размера шрифта, унаследованного от родительских элементов.

Единица vh

Единица измерения vh обозначает *высоту области просмотра*. Она эквивалентна 1/100 высоты экрана или окна браузера. Так, значение 100vh равно высоте окна браузера. Если посетитель сайта изменяет высоту окна браузера, то значение vh изменяется соответствующим образом.

Единица vw

Единица измерения vw обозначает *ширину области просмотра*. Она эквивалентна 1/100 ширины экрана или окна браузера. Так, значение 100vw равно ширине окна браузера. Если посетитель сайта изменяет ширину окна браузера, то значение vw соответственно изменяется.

Единица vmin

Эквивалентна минимальному из значений величин vh или vw. Другими словами, если высота окна браузера больше ширины, то значение 1 vmin равно 1 vw. Однако если ширина окна браузера больше высоты, то 1 vmin равно 1 vh.

Единица vmax

Равна максимальному из значений величин vh или vw. Другими словами, если высота окна браузера меньше ширины, то значение 1 vmax равно 1 vw. Однако если ширина окна браузера меньше высоты, то 1 vmax равно 1 vh.

Ключевые слова

У многих свойств есть собственные специфические значения, которые влияют на то, как проявляют себя свойства. Они представлены ключевыми словами. Свойство text-align, выравнивающее текст на экране, позволяет присвоить одно из четырех ключевых слов: right, left, center и justify. Поскольку ключевые слова различны в зависимости от свойства, читайте описания свойств далее, чтобы узнать, какие ключевые слова им соответствуют.

Есть одно ключевое слово, которое используется всеми свойствами, — inherit. Оно заставляет стиль наследовать значение от родительского элемента. Ключевое слово inherit дает возможность заставить стили унаследовать свойства, которые обычно не наследуются от родительских элементов. Например, вы используете свойство border, чтобы добавить границу по периметру абзаца. Остальные элементы, такие как em и strong внутри абзаца p, не наследуют это значение, но вы можете указать им сделать это с помощью ключевого слова inherit:

```
em, strong {  
  border: inherit;  
}
```

Таким образом, в этом случае элементы `em` и `strong` окружены такой же границей, как и их родительский элемент `p`. Элементы `em` и `strong` абзаца получают свои собственные границы, как и весь текст абзаца. Если бы они изначально наследовали значение свойства `border`, то у вас получились бы одни границы внутри других (а это серьезное основание, чтобы *не* наследовать данное свойство). Если вы измените значение `border` элемента `p` на *другой цвет или толщину линии*, элементы `em` и `strong` также унаследуют это изменение и отобразят соответствующее форматирование.

URL-адреса

Значения URL-ссылки позволяют указывать на другой файл во Всемирной паутине. Например, свойство `background-image` принимает URL-адрес — путь к файлу во Всемирной паутине — в качестве значения, которое позволяет вам использовать графический файл в качестве фона элемента страницы. Эта методика удобна при добавлении мозаичного изображения в фон страницы или при использовании собственных маркеров для неупорядоченных списков (см. раздел «Добавление фоновых изображений» главы 8).

В каскадных таблицах стилей URL-адреса указываются следующим образом: `url(images/tile.gif)`. Код стиля, который добавляет изображение с именем файла `tile.gif` в качестве фона страницы, выглядел бы следующим образом:

```
body { background-image: url(images/tile.gif); }
```

В отличие от языка HTML, в CSS-коде URL-адрес не обязательно заключать в кавычки, поэтому код `url("images/tile.gif")`, `url('images/tile.gif')` и `url(images/tile.gif)` идентичен.

ПРИМЕЧАНИЕ

URL-адреса в CSS-коде формируются так же, как и в случае HTML-атрибута `href`, используемого для указания ссылок. Это означает, что вы можете использовать как абсолютный адрес `http://piter.com/images/tile.gif`, так и корневой относительный `/images/tile.gif`, а также относительный от документа `../images/tile.gif`. Прочитайте врезку в разделе «Добавление фоновых изображений» главы 8 для получения полной информации об этих типах адресов.

Свойства текста

Описанные далее свойства определяют форматирование текста на веб-странице. Поскольку большинство свойств из этой категории наследуются, вам не обязательно применять их к элементам, специально предназначенным для текста (таким как элемент `p`). Вы можете применить эти свойства к элементу `body`, чтобы остальные элементы унаследовали и использовали те же самые свойства. Использование этой методики — быстрый способ применить общий шрифт, цвет и так далее на всей странице или во всем разделе.

color (наследуется)

Определяет цвет текста. Поскольку это свойство наследуется, то, если, к примеру, вы зададите красный цвет для элемента `body`, у всего текста на странице — и у всех других элементов внутри `body` — также будет красный цвет.

Значения: любое допустимое значение цвета.

Пример: `color: #FFFF33;`

ПРИМЕЧАНИЕ

Предустановленный в браузере цвет ссылок (элемента `a`) отменяет наследование цвета. В вышеупомянутом примере любые ссылки в элементе `body` будут стандартного синего цвета. Чтобы узнать, как изменить предустановленный цвет ссылок, читайте раздел «Выборка форматируемых ссылок» главы 9.

font (наследуется)

Используя это свойство, вы можете в виде сокращенной записи указать следующие характеристики текста в одном стиле: `font-style`, `font-variant`, `font-weight`, `font-size`, `line-height` и `font-family`.

В этом свойстве между значениями нужно указывать пробелы и включить по крайней мере свойства `font-size` и `font-family`, причем они должны быть последними двумя элементами в объявлении. Остальные свойства указываются при необходимости. Если вы не установите свойство, браузер будет использовать предустановленное значение, замещая унаследованные свойства.

Значения: любое значение, допустимое для соответствующего свойства шрифта. Если вы настраиваете значение свойства `line-height`, то сначала задается высота линии, а затем, через слеш, указывается размер шрифта, например: `1.25em/150%`.

Пример: `font: italic small-caps bold 1.25em/150% Arial, Helvetica, sans-serif;`

font-family (наследуется)

Позволяет указать шрифт, который браузер должен использовать при отображении текста. Шрифты обычно указываются наборами из трех-четырех пунктов, с учетом того, что указанный шрифт может быть не установлен на компьютере посетителя (см. раздел «Использование шрифтов» главы 6).

Значения: имена шрифтов, разделенные запятыми. Если имя шрифта состоит из нескольких слов, его нужно заключить в кавычки. Последний пункт в значении свойства — общий тип шрифта, позволяющий браузерам выбрать наиболее подходящий шрифт автоматически, если остальные указанные шрифты недоступны: `serif`, `sans-serif`, `monotype`, `fantasy` или `cursive`.

Пример: `font-family: "Lucida Grande", Arial, sans-serif;`

ПРИМЕЧАНИЕ

Все браузеры позволяют указывать дополнительные шрифты, которые либо хранятся на вашем веб-сервере, либо загружаются браузерами посетителей с сервера службы веб-шрифтов. Подробное описание веб-шрифтов представлено в главе 6, в разделе «Использование веб-шрифтов».

font-size (наследуется)

Устанавливает размер текста. Это свойство наследуется, что может привести к некоторому неожиданному эффекту при использовании относительных единиц измерения, таких как проценты и `em`.

Значения: любая допустимая в языке CSS единица измерения (см. подраздел «Размеры» предыдущего раздела этого приложения), а также следующие ключевые слова: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`, `larger` и `smaller`. `medium` — представляет обычное, предустановленное значение размера шрифта браузера, а остальные размеры — кратны ему. Каждое из них увеличивает или уменьшает текст в определенной степени. Несмотря на то что все изменения цвета, как предполагается, должны быть последовательными увеличениями или уменьшениями от предыдущего значения, на самом деле это не так. По существу, значение `xx-small` эквивалентно 9 пикселям (с расчетом, что вы не изменяли базовый размер шрифта браузера); `x-small` — 10 пикселям, `small` — 13, `large` — 18, `x-large` — 24 и `xx-large` — 32 пикселям. Из-за неопределенности в том, как каждый браузер обрабатывает эти ключевые слова, многие дизайнеры используют вместо них пиксели, единицы `em` или проценты.

Пример: `font-size: 1.25em;`

font-style (наследуется)

Применяет к тексту курсивное начертание. Если свойство применено к курсивному тексту, то возвращает его вновь к обычному. Значения `italic` и `oblique` функционально одинаковы.

Значения: `italic`, `oblique`, `normal`.

Пример: `font-style: italic;`

font-variant (наследуется)

Применяет к тексту начертание капителью, например: СПЕЦИАЛЬНАЯ ПРЕЗЕНТАЦИЯ. Значение `normal` возвращает текст с начертанием капителью к обычному виду.

Значения: `small-caps`, `normal`.

Пример: `font-variant: small-caps;`

font-weight (наследуется)

Форматирует текст полужирным начертанием или отменяет его для текста, который уже был отформатирован таким образом.

Значения: каскадные таблицы стилей на самом деле предоставляют 14 различных ключевых слов для свойства `font-weight`, но только два из них поддерживаются современными браузерами и компьютерными системами: `bold` и `normal`.

Пример: `font-weight: bold;`

letter-spacing (наследуется)

Позволяет настроить интервал между буквами, позволяя растянуть слова (добавляя расстояние между буквами) или сжать их (удаляя расстояние).

Значения: любая допустимая единица измерения языка CSS, хотя em и пикселы распространены больше всего. Проценты для этого свойства не работают в большинстве браузеров. Используйте положительное значение, чтобы увеличить интервал между буквами, и отрицательное значение, чтобы уменьшить. Значение `normal` сбрасывает интервал до стандартных значений в браузере.

Примеры:

```
letter-spacing: -1px;
```

```
letter-spacing: 2em;
```

line-height (наследуется)

Позволяет настроить интервал между строками текста в абзаце (в текстовых редакторах часто называется *межстрочным интервалом*). Стандартное значение составляет 120 % размера шрифта текста (см. раздел «Форматирование абзацев» главы 6).

Значения: большинство допустимых единиц измерения языка CSS (см. раздел «Размеры» этого приложения), хотя em, пикселы и проценты наиболее распространены.

Пример: `line-height: 200%;`

text-align (наследуется)

Выравнивает блок текста по левому, правому краю или по центру страницы либо элемента-контейнера.

Значения: `left`, `center`, `right`, `justify` (значение `justify` часто затрудняет чтение текста на мониторах).

Пример: `text-align: center;`

text-decoration (наследуется)

Добавляет линии над или под текстом, а также поверх текста. Подчеркивание привычно при оформлении ссылок, поэтому *не* рекомендуется подчеркивать текст, который не является ссылкой. Цвет линии такой же, как и цвет шрифта элемента, к которому применен стиль. Свойство также поддерживает значение `blink`, которое делает текст мерцающим (но большинство браузеров игнорируют значение `blink`).

Значения: `underline`, `overline`, `line-through`, `blink`, `none`. Значение `none` сбрасывает форматирование. Используйте его, чтобы скрыть линию подчеркивания под ссылками. Вы также можете добавить несколько линий в одном стиле, перечисляя значения (кроме `none`) через пробел.

Пример: `text-decoration: underline overline line-through;`

text-indent (наследуется)

Устанавливает размер отступа для первой строки абзаца текста. Первая строка может иметь отступ (как во многих печатных книгах) или выступать над левым краем остального текста.

Значения: любая допустимая единица измерения языка CSS. Пиксели и `em` наиболее распространены; проценты ведут себя иначе, чем в случае свойства `font-size`. Здесь проценты основываются на ширине области, содержащей текст, которая может быть шириной всего окна браузера. Таким образом, значение 50% сделает отступ первой строки на половину окна браузера. Чтобы первая строка выступала за левым краем, используйте отрицательное значение. Эта методика привлекательна в связке с положительным значением свойства `margin-left`, которое добавляет отступ с левой стороны других строк текста на установленную величину.

Пример: `text-indent: 3em;`

text-shadow (наследуется)

Позволяет добавить к тексту тень.

Значения: два значения (в `em` или пикселях) смещения тени по горизонтали и по вертикали, значение, определяющее степень размытости тени, и значение цвета тени. Отрицательное значение смещения по горизонтали помещает тень слева от текста, а положительное значение — справа. Отрицательное значение смещения по вертикали помещает тень сверху от текста, а положительное значение — снизу. Каждое значение отделяется от предыдущего пробелом. Можно также добавить несколько теней, добавляя настройки каждой тени через запятую.

Примеры:

```
text-shadow: -4px 4px 3px #999999;
```

```
text-shadow: -4px 4px 3px #999999, 2px 3px 10px #000;
```

text-transform (наследуется)

Изменяет регистр букв в тексте так, что все буквы в тексте становятся прописными, строчными или только первая буква каждого слова прописной.

Значения: `uppercase`, `lowercase`, `capitalize`, `none`. Значение `none` возвращает текст к фактическому регистру из HTML-кода. Допустим, *аБВГде* — буквы, которые именно так набраны в HTML-коде. Значение `none` отменит другие унаследованные свойства форматирования от элемента-предка и отобразит на экране *аБВГде*.

Пример: `text-transform: uppercase;`

vertical-align

Устанавливает базовую линию внутреннего элемента относительно базовой линии окружающего контента. Вы можете сделать так, чтобы символ появился немного выше или ниже окружающего текста. Используйте его для создания символов верхнего индекса, таких как [™], [®] или [©]. При применении к ячейке таблицы значения

top, middle, bottom и baseline управляют выравниванием по вертикали контента внутри ячейки (см. подраздел «Настройка горизонтального и вертикального выравнивания» раздела «Форматирование таблиц» главы 11).

Значения: baseline, sub, super, top, text-top, middle, bottom, text-bottom, процентное или абсолютное значение (например, пиксели или единицы em). Проценты вычисляются на основании значения свойства line-height элемента.

Примеры:

```
vertical-align: top;  
vertical-align: -5px;  
vertical-align: 75%;
```

white-space

Управляет тем, как браузер отображает пробельные символы из HTML-кода. Обычно, если вы добавляете более одного пробела между словами, например «Привет, Москва!», браузер отображает только один пробел — «Привет, Москва!». При использовании значения pre вы можете отобразить пробельные символы точно так же, как они набраны в HTML-коде. Это значение функционирует так же, как и HTML-элемент pre. Кроме того, браузер выполнит перенос текста в позиции пробела, если строка не будет соответствовать ширине окна. Чтобы препятствовать обтеканию текста, используйте значение nowrap. Однако это значение отображает *весь* текст абзаца в виде одной строки, так что не используйте его с длинными абзацами текста (если вам, конечно, не хочется заставить посетителей бесконечно прокручивать страницу вправо).

Значения: nowrap, pre, normal. Еще два значения — pre-line и pre-wrap — не поддерживаются в большинстве браузеров.

Пример: white-space: pre;

word-spacing (наследуется)

Схоже со свойством letter-spacing, но регулирует интервал между словами, а не буквами.

Значения: любая допустимая единица измерения языка CSS, но единицы em и пиксели наиболее распространены, а проценты не работают в большинстве браузеров. Используйте положительное значение, чтобы увеличить интервал между словами, и отрицательное значение, чтобы убрать его (сжать слова). Значение normal устанавливает стандартный интервал между словами, принимаемый в браузерах за 0.

Примеры:

```
word-spacing: -1px;  
word-spacing: 2em;
```

Свойства списков

Описанные далее свойства касаются форматирования маркированных (ul) и нумерованных списков (ol).

list-style (наследуется)

Применение этого свойства — сокращенный способ определения трех свойств, перечисленных далее. Вы можете указать значение для одного или нескольких этих свойств, разделяя каждое пробелом. Вы можете даже использовать это свойство, чтобы сэкономить время при наборе кода: `list-style: outside` вместо `list-style-position: outside`. Если вы укажете и тип списка, и изображение маркера, браузер отобразит стандартный маркер (кружок, квадрат и т. д.), *только* если не обнаружит указанное изображение. Таким образом, если изображение маркера недоступно, маркированный список все равно отобразится с маркерами.

Значения: любое подходящее значение для `list-style-type`, `list-style-image` и/или `list-style-position`.

Пример: `list-style: disc url(images/bullet.gif) inside;`

list-style-image (наследуется)

Позволяет указать ссылку на изображение, которое используется для маркера в маркированном списке.

Значения: URL-адрес или значение `none`.

Пример: `list-style-image: url(images/bullet.gif);`

СОВЕТ

Свойство `background-image` также позволяет определить для маркера изображение и предоставляет больше возможностей управления (см. раздел «Добавление фоновых изображений» главы 8).

list-style-position (наследуется)

Позиционирует маркеры или числа списка. Маркеры или числа могут отобразиться за пределами текста, выступая за левый край, или внутри текста (где обычно начинается первая буква первой строки). Значение `outside` определяет стандартное отображения маркеров и чисел.

Значения: `inside`, `outside`.

Пример: `list-style: inside;`

list-style-type (наследуется)

Устанавливает тип маркера для списка: круг, квадрат, римские цифры и т. д. Вы можете даже превратить неупорядоченный (маркированный) список в упорядоченный (нумерованный), изменяя значение свойства `list-style-type`. Используйте значение `none`, чтобы полностью удалить маркеры или числа из списка.

Значения: `disc`, `circle`, `square`, `decimal`, `decimal-leading-zero`, `upper-alpha`, `lower-alpha`, `upper-roman`, `lower-roman`, `lower-greek`, `none`.

Пример: `list-style-type: square;`

Отступы, границы и поля

Описанные далее свойства управляют пустым пространством по периметру элемента и позволяют добавлять границы.

box-shadow

Добавляет тень по периметру блочного элемента.

Значения: необязательное значение `inset`, добавляющее тень внутрь элемента, за которым следуют четыре числовых значения характеристик тени и цвета. Первые два значения (в единицах `em` или пикселах) задают смещение тени по горизонтали и по вертикали, третье — степень размытия тени, четвертое, необязательное, задает «расширение» тени, делая ее шире. Отрицательное значение для смещения по горизонтали помещает тень слева от блока, а положительное значение — справа. Отрицательное значение для смещения по вертикали помещает тень сверху от блока, а положительное значение — снизу. Каждое значение отделяется от предыдущего пробелом. Можно также добавить несколько теней, добавляя значения дополнительной тени через запятую.

Примеры:

```
box-shadow: -4px 4px 3px #999999;  
box-shadow: inset 4px 4px 3px 5px #999999;  
box-shadow: inset 4px 4px 3px 5px #999999, 2px 2px 5px black;
```

border

Создает линию по краям четырех сторон элемента.

Значения: толщина границы задается с помощью любой допустимой единицы измерения CSS (кроме процентов).

Вы также можете определить стиль для линии: `solid`, `dotted`, `dashed`, `double`, `groove`, `ridge`, `inset`, `outset`, `none` и `hidden`. Значения `none` и `hidden` делают одно и то же — удаляют любую границу.

Наконец, вы можете определить цвет, используя любое допустимое значение цвета в CSS (к примеру, имя: `green` — или шестнадцатеричное значение, такое как `#33fc44`).

Пример: `border: 2px solid #f33;`

border-radius

Скругляет углы элемента. Визуальный эффект проявляется только в том случае, если у элемента есть граница, цвет фона или изображение.

Значения: одно, два, три или четыре значения (в пикселах, единицах `em` или процентах), задающих размер радиуса окружности, создаваемой в углах блока элемента. Если указано только одно значение, один и тот же размер скругленного угла применяется ко всем четырем углам, если указано два значения, первое определяет радиус скругления верхнего левого и нижнего правого углов, а второе — верх-

него правого и нижнего левого углов. Если используются три значения, первое определяет радиус скругления верхнего левого угла, второе — верхнего правого и нижнего левого углов, а третье — нижнего правого угла. Если используются четыре значения, то каждое из них применяется по порядку к верхнему левому, верхнему правому, нижнему правому и нижнему левому углам. Для создания эллипсoidalных углов можно добавить символ /, после которого указать второе значение радиуса.

Примеры:

```
border-radius: .5em;  
border-radius: 15px 10px 25px 5px;  
border-radius: 15px / 5px;
```

border-top, border-right, border-bottom, border-left

Добавляют границу к соответствующему краю. Например, `border-top` добавляет границу к верхнему краю элемента.

Значения: те же, что и для `border`.

Пример: `border-left: 1em dashed red;`

border-color

Определяет цвет, используемый для всех четырех границ.

Значения: любое допустимое значение цвета в CSS (имя, например `green`, или шестнадцатеричное значение, такое как `#33fc44`).

Пример: `border-color: rgb(255,34,100);`

У этого свойства есть еще и сокращенная запись, позволяющая присваивать различные цвета каждой из четырех границ.

Значения: любое допустимое значение цвета в CSS для каждой границы: верхней, правой, нижней и левой. Если вы укажете только два значения, то первое будет присвоено верхней и нижней, а второе — правой и левой границам.

Пример: `border-color: #000 #F33 #030 #438F3C;`

border-top-color, border-right-color, border-bottom-color, border-left-color

Функционируют так же, как свойство `border-color`, но устанавливают цвет только для одной соответствующей границы. Используйте эти свойства, чтобы отменить цвет, установленный свойством `border`. Таким образом, вы можете изменить цвет определенной одной границы, применив ранее более общее свойство `border`, формирующее все четыре границы.

Значения: как и для свойства `border-color`.

Пример: `border-left-color: #333;`

border-style

Определяет стиль всех четырех границ.

Значения: одно из ключевых слов: `solid`, `dotted`, `dashed`, `double`, `groove`, `ridge`, `inset`, `outset`, `none` и `hidden` (см. рис. 7.7, где показаны примеры различных стилей). Значения `none` и `hidden` действуют одинаково — удаляют любую границу.

Пример: `border-style: inset;`

У этого свойства есть сокращенная запись, позволяющая присваивать различные стили для каждой из четырех границ: верхней, правой, нижней и левой.

Значения: одно из ключевых слов, упомянутых выше, для каждой из четырех границ. Если вы укажете только два значения, первое будет присвоено верхней и нижней, а второе — правой и левой границам.

Пример: `border-style: solid dotted dashed double;`

border-top-style, border-right-style, border-bottom-style, border-left-style

Функционируют точно так же, как свойство `border-style`, но применяются только к одному соответствующему краю элемента.

Значения: те же, что и для свойства `border-style`.

Пример: `border-top-style: none;`

border-width

Определяет толщину линии, используемой в качестве всех четырех границ.

Значения: любая допустимая в языке CSS единица измерения, кроме процентов. Наиболее часто используются единицы `em` и пиксели.

Пример: `border-width: 1px;`

У этого свойства есть еще и сокращенная запись, позволяющая присваивать различную толщину линии для каждой из четырех границ.

Значения: любая допустимая в языке CSS единица измерения, кроме процентов, для каждой из четырех границ. Если вы укажете только два значения, первое будет присвоено верхней и нижней, а второе — правой и левой границам.

Пример: `border-width: 3em 1em 2em 3.5em;`

border-top-width, border-right-width, border-bottom-width, border-left-width

Функционируют точно так же, как свойство `border-width`, но применяются только к одному соответствующему краю.

Значения: как и для свойства `border-width`.

Пример: `border-bottom-width: 3em;`

box-sizing

Предписывает браузеру порядок измерения высоты и ширины элемента. Обычно браузеры для определения объема экранного пространства, занимаемого элемен-

том, объединяют значения границ, отступов и ширины элемента. Такая система обычно запутывает, так как, к примеру, если ширина элемента равна 300 пикселям и у элемента также есть отступы и границы, браузер отображает элемент на экране в пространстве, превышающем по ширине 300 пикселей.

Значения: `content-box`, `padding-box` или `border-box`. Вариант `content-box` задает обычный порядок визуализации элемента браузером. Значение `padding-box` предписывает браузеру включать в расчет наряду со значениями ширины и высоты элемента еще и размер отступов. Вариант `border-box` заставляет браузер включать в расчет еще и размер границ. Например, если у вас есть элемент `div`, для которого установлена ширина 300 пикселей, отступы в 20 пикселей и граница толщиной 2 пикселя, браузер, как правило, отобразит этот `div`-элемент на 344 пикселях экранного пространства ($300 + 20 + 20 + 2 + 2$). Если свойству `box-sizing` присвоено значение `padding-box`, браузер отобразит этот `div`-элемент на 304 пикселях экранного пространства ($300 + 2 + 2$), потому что отступы считаются частью этих 300 пикселей, а если присвоено значение `border-box`, то браузеру отобразит `div`-элемент шириной 300 пикселей. Вариант `border-box` широко используется для удобства отслеживания ширины элемента.

Пример: `box-sizing: border-box;`

outline

Применение этого свойства — лаконичный способ объединить характеристики `outline-color`, `outline-style` и `outline-width` (перечислены далее). Контур, задаваемый этим свойством, работает точно так же, как граница, за исключением того, что он не влияет на размер элемента (то есть не увеличивает ширину или высоту элемента) и относится ко всем четырем краям. Оно больше применяется как средство выделения контента на странице, чем как элемент дизайна.

Значения: те же самые, что относятся к `border`, с одним исключением (см. описание свойства `outline-color` далее).

Пример: `outline: 3px solid #F33;`

outline-color

Определяет цвет для контура (см. описание свойства `outline`).

Значения: любое допустимое значение цвета в CSS плюс значение `invert`, которое изменяет цвет контура (цвет, на котором расположен контур) на противоположный. Если контур нарисован на белом фоне, значение `invert` сделает его черным. Работает точно так же, как свойство `border-color`.

Пример: `outline-color: invert;`

outline-style

Определяет тип линии для контура: пунктирная, сплошная, штриховая и т. д.

Значения: те же самые, что и для свойства `border-style`, описанного выше.

Пример: `outline-style: dashed;`

outline-width

Определяет толщину контура. Работает так же, как и свойство `border-width`.

Значения: любая допустимая в языке CSS единица измерения, кроме процентов. Наиболее распространенные — единицы `em` и пиксели.

Пример: `outline-width: 3px;`

margin

Устанавливает размер пространства между границей элемента и полем других элементов (см. рис. 7.1). Свойство позволяет добавлять пустое пространство между двумя элементами: между двумя изображениями или между боковой панелью и областью основного контента страницы.

ПРИМЕЧАНИЕ

Вертикальные поля между элементами могут схлопываться. Иными словами, браузеры используют только верхнее или только нижнее поле и игнорируют остальные, создавая меньший промежуток, чем ожидается (см. практикум главы 7).

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели или `em`. Значения в процентах основаны на ширине элемента-контейнера. Заголовок, который является потомком дочернего элемента `body`, использует ширину окна браузера, чтобы вычислить значение в процентах, так что поле шириной 10 % добавляет 10 % ширины окна к краям заголовка. Если посетитель изменяет размер окна браузера, изменяется и размер поля. Как и в случае с отступом, вы определяете поля со всех четырех краев элемента, используя одно значение, или определяете поля по отдельности в следующем порядке: `top`, `right`, `bottom`, `left`.

Примеры:

`margin: 20px;`

`margin: 2em 3em 2.5em 0;`

margin-top

Работает точно так же, как свойство `margin`, но устанавливает поле только для верхнего края элемента.

Пример: `margin-top: 20px;`

margin-right

Работает точно так же, как свойство `margin`, но устанавливает поле лишь для правого края элемента.

Пример: `margin-right: 20px;`

margin-bottom

Работает точно так же, как свойство `margin`, но устанавливает поле только для нижнего края элемента.

Пример: `margin-bottom: 20px;`

margin-left

Работает точно так же, как свойство `margin`, но устанавливает поле лишь для левого края элемента.

Пример: `margin-left: 20px;`

padding

Устанавливает размер области между контентом, границей и краем фона. Используйте его, чтобы добавить пустое пространство вокруг текста, изображений или другого контента (см. рис. 7.1 для наглядной демонстрации).

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели или `em`. Значения в процентах основываются на ширине элемента-контейнера. Заголовок, являющийся потомком элемента `body`, использует ширину окна браузера, чтобы вычислить значение в процентах, так что отступ шириной 20 % добавляет 20 % ширины окна. Если посетитель изменяет размер окна браузера, размер отступа изменяется пропорционально. Вы можете определить отступ для всех четырех краев, используя одно значение, или установить размеры отступа по отдельности для каждого края в таком порядке: `top, right, bottom, left`.

Примеры:

`padding: 20px;`

`padding: 2em 3em 2.5em 0;`

padding-top

Работает точно так же, как свойство `padding`, но устанавливает отступ только для верхнего края элемента.

Пример: `padding-top: 20px;`

padding-right

Работает точно так же, как свойство `padding`, но устанавливает отступ лишь для правого края элемента.

Пример: `padding-right: 20px;`

padding-bottom

Работает точно так же, как свойство `padding`, но устанавливает отступ только для нижнего края элемента.

Пример:

`padding-bottom: 20px;`

padding-left

Работает точно так же, как свойство `padding`, но устанавливает отступ лишь для левого края элемента.

Пример: `padding-left: 20px;`

Фоны

Каскадные таблицы стилей предоставляют несколько свойств для управления фоновым элементом, включая изменение его цвета, размещение изображения позади элемента и позиционирование этого фонового изображения.

background

Обеспечивает лаконичный способ определения свойств, которые проявляются в фоне элемента, таких как цвет, изображение и положение этого изображения. Оно объединяет пять свойств фона (описаны далее) в одну компактную строку так, что вы можете получить тот же самый результат с меньшим объемом кода. Однако если вы не установите одно из этих свойств, браузеры будут использовать стандартное значение. Например, если вы не определите, как должно повторяться фоновое изображение, браузеры будут повторять это изображение слева направо и сверху вниз (см. раздел «Управление повтором фоновых изображений» главы 8).

Значения: те же самые значения, что используются для свойств фона, перечисленных далее. Порядок свойств неважен (за исключением позиционирования, как описано ниже), но лучше указывать свойства в такой последовательности: `background-color`, `background-image`, `background-repeat`, `background-attachment`, `background-position`.

Пример: `background: #333 url(images/logo.gif) no-repeat fixed left top;`

background-attachment

Определяет поведение фонового изображения, когда ваш посетитель прокручивает страницу. Изображение либо прокручивается наряду с остальным контентом, либо остается на месте. К примеру, вы можете добавить логотип в левом верхнем углу очень длинной веб-страницы, присвоив значение `fixed` свойству `background-attachment`, и заставить это изображение тем самым находиться в левом верхнем углу, когда страница прокручивается.

Значения: `scroll` или `fixed`. `scroll` — это обычное поведение: изображение прокручивается наряду с контентом. `fixed` закрепляет изображение на месте.

Пример: `background-attachment: fixed;`

background-clip

Это свойство ограничивает область, в которой появляется фоновое изображение. Обычно фоновое изображение заполняет всю область элемента, включая его границы, отступы и контент. Но может потребоваться, чтобы изображение появлялось только позади отступов и не распространялось на границы, что может пригодиться в случае использования пунктирных границ, чтобы изображение не отображалась в промежутках между штрихами границы. Может также потребоваться исключить область отступов, чтобы мозаичное изображение фона появлялось только в области контента, а в области отступов использовался сплошной цвет. В Internet Explorer 8 и более ранних его версиях это свойство не поддерживается.

Значения: border-box, padding-box или content-box. Вариант border-box задает обычный метод, при котором изображение помещается позади границ, отступов и контента. Вариант padding-box располагает фоновое изображение только в области отступов и контента, и оно не отображается позади границы. Вариант content-box помещает фоновое изображение только в области контента, и оно не появляется ни позади области отступов, ни позади границы.

Примеры:

```
background-clip: content-box;  
background-clip: padding-box;
```

background-color

Добавляет цвет в качестве фона элемента. Фон располагается позади границы и фонового изображения — это нужно иметь в виду, если вы используете такие стили границы, как dashed или dotted. В этих случаях фоновый цвет виден сквозь промежутки между штрихами границы.

Значения: любое допустимое значение цвета в CSS (см. подраздел «Цвета» раздела «Значения свойств CSS» этого приложения).

Пример: background-color: #FFF;

background-image

Помещает изображение в качестве фона элемента. Остальные элементы страницы находятся поверх фонового изображения, так что убедитесь в том, что текст читабелен в области наложения на изображение. Вы также можете настроить отступы, чтобы сместить контент от изображения. Изображение повторяется слева направо и сверху вниз, если только вы не устанавливаете иное поведение с помощью свойства background-repeat. В CSS-коде допускается использование нескольких фоновых изображений.

Значения: URL-адрес изображения. Может также включать сгенерированный браузером линейный или радиальный градиент.

Примеры:

```
background-image: url(images/photo.jpg);  
background-image: url(http://www.example.org/photo.jpg);  
background-image: url(http://www.example.org/photo.jpg), url(images/photo.jpg);
```

background-origin

Инструктирует браузер, куда поместить фоновое изображение относительно границ, отступов и контента элемента. Больше подходит для неповторяющегося изображения, поскольку позволяет указать его позицию. Браузер Internet Explorer 8 и более ранние версии это свойство не поддерживают.

Значения: border-box, padding-box или content-box. Вариант border-box задает обычный метод — помещение изображения в верхний левый угол границы. Вариант padding-box задает начало фонового изображения только в области отступов,

и позади границы оно не отображается. Вариант `content-box` помещает фоновое изображение только в область контента, и оно не отображается в области отступов и позади границы. Но, если изображение размножено, вы все равно его увидите позади границ и в области отступов, поскольку это свойство управляет только тем, где начинается изображение. Чтобы размноженное изображение не появлялось позади границ и в области отступов, следует воспользоваться свойством `background-clip`.

Пример: `background-origin: content-box;`

background-position

Управляет размещением изображения в качестве фона элемента страницы. Если только вы не определите иначе, изображение начинается в верхнем левом углу элемента. Если изображение мостится мозаикой, свойство `background-position` определяет начальную точку изображения. Если вы позиционируете изображение в центре элемента, то браузер помещает его там, а затем мостит вверх и налево, а *также* вниз и направо. Во многих случаях точное позиционирование изображения не вызывает видимого различия при повторении фона, но позволяет вносить едва заметные изменения в позиционирование узора в фоне.

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели или `em`, а также ключевые слова или проценты. Значения указываются парами, где первое является позицией по горизонтали, а второе — по вертикали. Ключевые слова: `left`, `center` и `right` для позиционирования по горизонтали и `top`, `center` и `bottom` — по вертикали. Значения в пикселах и единицах `em` рассчитываются от верхнего левого угла элемента, поэтому, чтобы поместить изображение на расстоянии 5 пикселей от левого края и 10 пикселей от верхнего, нужно использовать значение `5px 10px`.

Значения в процентах сопоставляют одну точку изображения с одной точкой в фоне элемента на основе вычислений указанных процентных соотношений от левого и верхнего краев изображения и от левого и верхнего краев элемента. Значение `50% 50%` помещает изображение в центре элемента (см. раздел «Позиционирование фоновых изображений» главы 8). Вы можете использовать различные значения: по желанию, используйте значение в пикселах для выравнивания по горизонтали и значение в процентах — по вертикали.

Примеры:

```
background-position: left top;  
background-position: 1em 3em;  
background-position: 10px 50%;
```

background-repeat

Устанавливает, повторяется ли фоновое изображение, и если повторяется, то как. Обычно фоновое изображение повторяется от левого верхнего до правого нижнего края, заполняя весь фон элемента.

Значения: repeat, no-repeat, repeat-x, repeat-y. Значение repeat используется по умолчанию, оно повторяет изображение слева направо, сверху вниз. Значение no-repeat помещает изображение в фон один раз без какого-либо повторения. Значение repeat-x повторяет изображение только сверху вниз — это удобно при создании графической боковой панели. Значение repeat-y повторяет изображение только слева направо, благодаря чему вы можете добавить графическую полосу вверху, по центру или внизу элемента.

Пример: background-repeat: no-repeat;

background-size

Позволяет изменять размер фонового изображения, масштабируя его в разных направлениях или даже искажая его пропорции.

Значения: можно использовать точные значения в пикселах, единицах em или процентах или же воспользоваться одним из двух ключевых слов: contain или cover. Ключевое слово contain изменяет размеры изображения так, чтобы полностью уместить его в пределах элемента, сохраняя при этом пропорции. Ключевое слово cover изменяет ширину и высоту изображения для соответствия размерам элемента, что обычно искажает изображение, растягивая его или сжимая, чтобы вписать его в пределы элемента.

Примеры:

```
background-size: 200px 400px;
```

```
background-size: contain;
```

Компоновка макета

Следующие свойства управляют положением и размером элементов на веб-странице.

bottom

Это свойство применяется с абсолютным, относительным и фиксированным позиционированием. При использовании с абсолютным или фиксированным позиционированием свойство bottom определяет позицию нижнего края формируемого элемента относительно нижнего края ближайшего предка. Если формируемый элемент не находится внутри каких-либо позиционированных элементов, то он будет размещен относительно нижнего края окна браузера. Вы можете использовать это свойство, к примеру, чтобы поместить сноску в нижней части окна браузера. При использовании с относительным позиционированием положение элемента вычисляется от нижнего края элемента.

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели, em или проценты. Проценты вычисляются на основании ширины элемента-контейнера.

Пример: bottom: 5em;

clear

Препятствует тому, чтобы элемент обтекал выровненный элемент. Вместо этого форматруемый элемент опускается вниз — ниже выровненного элемента.

Значения: `left`, `right`, `both`, `none`. Значение `left` запрещает обтекание элементов, выровненных по левому краю, а `right` — элементов, выровненных по правому краю. Значение `both` предохраняет элемент от обтекания элементов, выровненных по *любому* краю. Значение `none` отменяет действие свойства, поэтому используйте это значение для замещения ранее установленного свойства `clear`. Этот прием используется при наличии у конкретного элемента стиля, вызывающего выпадение выровненного элемента, когда нужно, чтобы элемент обтекал его только в данном случае. Чтобы отменить обтекание только для определенного элемента, создайте более специфичный стиль.

Пример: `clear: both;`

clip

Создает прямоугольную область, отображающую часть элемента. Если у вас есть фотография вашего выпускного класса, на которой вы стоите крайним справа, вы могли бы создать в этом месте область, отсекающую всю остальную фотографию. Остальная фотография будет скрыта, а область отсечения отобразит только вас. Свойство `clip` наиболее эффективно, когда используется в связке с JavaScript-сценариями, анимирующими область отсечения. К примеру, вы можете начать с маленькой области отсечения и расширять ее, пока не будет показана вся фотография.

Значения: координаты прямоугольной области. Заключите координаты в круглые скобки и укажите перед ними ключевое слово `rect`, например: `rect(5px, 110px, 35px, 10px);`.

Рассмотрим, как действует порядок этих координат: первое число указывает смещение сверху — верхний край области отсечения. В этом примере смещение равно `5px`, так что будет скрыто все, что находится в четырех первых строках пикселей. Последнее число — смещение слева — левый край области отсечения. В этом примере смещение равно `10px`, так что будет скрыта область размером 9 пикселей слева. Второе число определяет ширину области отсечения плюс значение смещения слева (последнее значение); если левый край области отсечения составляет 10 пикселей и вы хотите, чтобы видимая область была шириной 100 пикселей, второе значение должно быть равно `110px`. Третье число — высота области отсечения плюс смещение сверху (первое значение). Так, в этом примере высота области отсечения равна 30 пикселям ($30 + 5 = 35$ пикселей).

Пример: `clip: rect(5px, 110px, 40px, 10px);`

СОВЕТ

Поскольку порядок указания координат здесь немного странный, большинству дизайнеров нравится начинать с первого и последнего аргументов, а затем рассчитывать два других.

display

Определяет вид области, используемой для отображения элемента страницы — блочного или строчного (см. раздел «Управление размерами полей и отступов» главы 7). Используйте его, чтобы изменить вариант отображения по умолчанию. Вы можете сделать так, чтобы абзац (блочный элемент) отображался без интервалов (разрывов строк) над и под ним — точно так же, как, скажем, ссылка (строчный элемент).

Значения: `block`, `inline`, `none`. Свойство `display` поддерживает 17 значений, большинство из которых не функционирует в современных браузерах. Значения `block`, `inline` и `none` тем не менее работают практически во всех браузерах. Значение `block` отображает интервал над и под элементом, точно так же, как и другие блочные элементы (например, абзацы и заголовки); `inline` отображает элемент на той же строке, что и окружающие элементы (точно так же, как текст внутри элемента `strong` появляется на той же строке, что и остальной текст); `none` скрывает элемент со страницы. Затем вы можете вновь отобразить его, используя JavaScript-сценарий или псевдокласс `:hover` (см. раздел «Псевдоклассы и псевдоэлементы» главы 3).

Пример: `display: block;`

float

Выравнивает элемент по левому или правому краю окна браузера или, если этот элемент вложен в другой, по левому или правому краю элемента-контейнера. Можно сказать, что свойство `float` делает элемент обтекаемым. Элементы, которые расположены в потоке после выровненного, перемещаются, чтобы заполнить область справа (для выровненных по левому краю элементов) или слева (для выровненных по правому краю элементов), а затем обтекают выровненный элемент. Используйте данное свойство для простых эффектов, таких как перемещение изображения к какой-либо стороне страницы, или для очень сложных дизайнов — таких, что были описаны в главе 12.

Значения: `left`, `right`, `none`. Значение `none` полностью сбрасывает выравнивание, что может оказаться полезным, если у формируемого элемента уже есть стиль, выравнивающий его по левому или по правому краю, и вы хотите создать более специфичный стиль, чтобы отменить выравнивание этого элемента.

Пример: `float: left;`

height

Устанавливает высоту *области контента* — пространство блока элемента, в котором содержатся, например, текст, изображения и др. Фактическая высота элемента на экране — это общая сумма высоты, верхнего и нижнего полей, верхнего и нижнего отступов, а также верхней и нижней границ.

Значения: любая допустимая в языке CSS единица измерения, например пиксели, `em` или проценты. Проценты вычисляются на основании высоты элемента-контейнера.

Пример: `height: 50%;`

ПРИМЕЧАНИЕ

Иногда контент превышает установленную высоту: к примеру, если вы вводите много текста или посетитель увеличивает размер шрифта в настройках браузера, контент выпадает за пределы области, фоновый цвет и границы. Свойство `overflow` определяет, что происходит в этом случае.

left

При использовании с абсолютным или фиксированным позиционированием (см. раздел «Принципы работы свойств позиционирования» главы 14) это свойство определяет позицию левого края формируемого элемента относительно левого края ближайшего предка. Если формируемый элемент не находится внутри каких-либо позиционированных элементов, то он будет размещаться относительно левого края окна браузера. Вы можете применять это свойство, чтобы поместить изображение на расстоянии 20 пикселей от левого края окна браузера. При использовании с относительным позиционированием положение элемента вычисляется от его левого края.

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели, `em` или проценты.

Пример: `left: 5em;`

max-height

Ограничивает максимальную допустимую высоту элемента. Таким образом, область элемента может быть ниже установленного значения, но не выше. Если контент элемента превышает значение свойства `max-height`, он выпадает за пределы области. Вы можете управлять поведением контента в такой ситуации с помощью свойства `overflow`. Браузер Internet Explorer версии 6 и ниже не поддерживает свойство `max-height`.

Значения: любая допустимая единица измерения CSS, такая как пиксели, `em` или проценты. Браузеры вычисляют проценты на основании высоты элемента-контейнера.

Пример: `max-height: 100px;`

max-width

Ограничивает максимальную допустимую ширину элемента. Таким образом, область элемента может быть уже установленного значения, но не шире. Если контент элемента превышает значение свойства `max-width`, он выпадает за пределы области. Вы можете управлять поведением контента в такой ситуации с помощью свойства `overflow`.

В основном свойство `max-width` применяется в резиновых дизайнах (см. раздел «Типы макетов веб-страниц» главы 12). Благодаря этому свойству разработчик может быть уверен — дизайн страницы на очень больших мониторах не станет настолько широким, что будет непригоден для чтения.

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели, em или проценты. Браузеры вычисляют проценты на основании ширины элемента-контейнера.

Пример: `max-width: 950px;`

min-height

Ограничивает минимальную допустимую высоту элемента. Таким образом, область элемента может быть выше установленного значения, но не ниже. Если контент элемента становится по высоте меньше, чем значение свойства `min-height`, высота элемента не уменьшается.

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели, em или проценты. Проценты рассчитываются на основе высоты элемента-контейнера.

Пример: `min-height: 20em;`

min-width

Ограничивает минимальную допустимую ширину элемента. Таким образом, область элемента может быть шире установленного значения, но не уже. Если контент элемента становится по ширине меньше, чем значение свойства `min-width`, ширина элемента не уменьшается.

Вы можете использовать свойство `min-width` в резиновых дизайнах. Благодаря этому свойству разработчик может быть уверен, что макет страницы не «рассыплется» при уменьшении ширины окна. Если окно браузера становится уже значения, присвоенного свойству `min-width`, добавляются горизонтальные полосы прокрутки.

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели, em или проценты. Проценты рассчитываются на основе ширины элемента-контейнера.

Пример: `min-width: 760px;`

ПРИМЕЧАНИЕ

Свойства `max-width` и `min-width` обычно применяются в связке при создании адаптированных дизайнов (см. главу 12).

overflow

Определяет поведение контента, который выпадает за пределы своей области содержимого. Например, в случае с фотографией, оказавшейся шире, чем установленное свойством `width` для области контента.

Значения: `visible`, `hidden`, `scroll`, `auto`. Значение `visible` расширяет контент за пределы области, накладывая его на границы и другие элементы страницы. Значение `hidden` скрывает контент за пределами отведенной ему области; `scroll` отображает полосы прокрутки к элементу, в результате чего посетитель может выполнить прокрутку, чтобы просмотреть неумещающиеся области контента. Значение `auto`

добавляет полосы прокрутки, *только* если они необходимы, чтобы показать не уместяющиеся области контента.

Пример: `overflow: hidden;`

position

Определяет тип позиционирования элементов браузером на странице.

Значения: `static`, `relative`, `absolute`, `fixed`. Значение `static` определяет обычный режим браузера — блочные элементы отображаются друг под другом, контент располагается от верхнего до нижнего края экрана. Значение `relative` позиционирует элемент относительно его положения, то есть присвоение этого значения смещает элемент с его текущей позиции. Значение `absolute` полностью извлекает элемент из потока контента. Другие элементы не «видят» абсолютно позиционированный элемент и отображаются позади него. Это значение используется для установки элемента в конкретное положение на странице или для его размещения в определенном месте относительно родительского элемента с абсолютным, относительным или фиксированным позиционированием. Значение `fixed` фиксирует элемент на странице, и если она прокручивается, фиксированный элемент остается на экране по аналогии с HTML-фреймами. Браузер Internet Explorer версии 6 и ниже игнорирует значение `fixed`.

Пример: `position: absolute;`

ПРИМЕЧАНИЕ

Чаще всего применяются значения `relative`, `absolute` и `fixed` совместно со свойствами `left`, `right`, `top` и `bottom`. Все подробности о позиционировании вы можете узнать в главе 13.

right

При использовании с абсолютным или фиксированным позиционированием это свойство определяет позицию правого края форматлируемого элемента относительно правого края ближайшего предка. Если форматлируемый элемент не находится внутри каких-либо позиционированных элементов, то он будет размещаться относительно правого края окна браузера. Вы можете использовать это свойство, чтобы поместить изображение на расстоянии 20 пикселей от правого края окна браузера. При использовании с относительным позиционированием положение элемента вычисляется от его правого края.

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели, `em` или проценты.

Пример: `right: 5em;`

ПРИМЕЧАНИЕ

В программе Internet Explorer версии 6 и ниже могут быть проблемы при позиционировании элементов с использованием свойства `right` (см. врезку в разделе «Принципы работы свойств позиционирования» главы 14).

top

Это свойство, противоположное свойству `bottom`, применяется с абсолютным, относительным и фиксированным позиционированием. При использовании с абсолютным или фиксированным позиционированием свойство `top` определяет позицию верхнего края форматируемого элемента относительно верхнего края ближайшего предка. Если форматируемый элемент не находится внутри каких-либо позиционированных элементов, то он будет помещен относительно верхнего края окна браузера. Вы можете использовать это свойство, к примеру, чтобы поместить логотип в верхней части окна браузера. При использовании с относительным позиционированием положение элемента вычисляется от верхнего края элемента.

Значения: любая допустимая в языке CSS единица измерения, такая как пиксели, `em` или проценты.

Пример: `top: 5em;`

visibility

Определяет, отображает ли браузер элемент. Используйте это свойство, чтобы скрыть некоторые объекты страницы, например абзац, заголовок или содержимое контейнера `div`. В отличие от значения `none` свойства `display`, установка которого скрывает элемент и удаляет его из потока страницы, значение `hidden` свойства `visibility` *не* удаляет элемент из потока страницы. Вместо этого остается пустое пространство в области, где должен отображаться элемент. По этой причине свойство `visibility` чаще применяется с абсолютно позиционированными элементами, которые уже были удалены из потока страницы.

Соккрытие элемента не принесет большой пользы, если вы не сможете показать его снова. JavaScript-сценарии — самый распространенный способ управления значением свойства `visibility` для отображения и сокрытия элементов на странице. Вы можете также использовать псевдокласс `:hover` (см. раздел «Псевдоклассы и псевдоэлементы» главы 3), чтобы изменить свойство `visibility` элемента, когда посетитель наводит указатель мыши на определенные области страницы.

Значения: `visible`, `hidden`. Кроме того, можно использовать значение `collapse`, чтобы скрыть строку или столбец в таблице.

Пример: `visibility: hidden;`

width

Устанавливает ширину *области контента* — пространство блока элемента, в котором содержатся, например, текст, изображения и др. Фактическая ширина элемента на экране — это сумма ширины, левого и правого полей, левого и правого отступов, а также левой и правой границ.

Значения: любая допустимая в языке CSS единица измерения, например пиксели, `em` или проценты. Проценты вычисляются на основании ширины элемента-контейнера.

Пример: `width: 250px;`

z-index

Управляет порядком наложения позиционированных элементов. Относится только к элементам, у которых свойству `position` присвоено значение `absolute`, `relative` или `fixed`. Свойство определяет, где отображается элемент по оси Z. Если два абсолютно позиционированных элемента накладываются друг на друга, тот, у которого более высокий индекс позиционирования, окажется поверх.

Значения: целочисленные значения, такие как 1, 2 или 10. Вы также можете использовать отрицательные значения, но различные браузеры обрабатывают их по-разному. Чем больше число, тем «выше» в стопке расположен элемент. Элемент со значением 20 свойства `z-index` будет расположен позади элемента со свойством `z-index: 100` (если эти два элемента наслаиваются) (см. рис. 15.6).

Пример: `z-index: 12;`

СОВЕТ

Значения не обязательно должны задаваться в строгом порядке. Если элементу А присвоено свойство `z-index: 1;`, вам не обязательно присваивать свойство `z-index: 2;` элементу Б, чтобы поместить его поверх. Вы можете использовать любое число — 5, 10 и т. д., чтобы получить тот же самый результат, главное, чтобы число было больше. Так, чтобы убедиться в том, что элемент всегда будет появляться поверх других элементов, присвойте ему очень большое значение, например 10000. Но помните, что максимальное значение, которое поддерживает браузер Firefox, равняется 2147483647, поэтому не превышайте его.

Свойства анимации, преобразований и переходов

В CSS доступны весьма интересные свойства для преобразования элементов путем масштабирования, вращения, скашивания и перемещения, а также возможность анимировать изменения от одного значения свойства к другому.

@keyframes

Основой анимации с помощью каскадных таблиц стилей является правило `@keyframes`. Оно позволяет присвоить анимации имя, которое затем можно будет применить к любому элементу страницы и создать набор ключевых кадров. Ключевые кадры являются позициями анимации, где происходят изменения свойств каскадных таблиц стилей. Например, чтобы постепенно превратить фон элемента из черного в белый, нужны два ключевых кадра: первый для установки свойства `background-color: black;`, а второй — для установки свойства `background-color: white;`. Ключевых кадров с различными свойствами может быть сколько угодно.

На момент написания данной книги к правилу `@keyframes` необходимо было добавлять вендорный префикс для поддержки анимации в браузере Safari. Кроме того, анимации не поддерживаются в браузере Internet Explorer 9 и более ранних версиях (см. главу 10).

Значения: правило `@keyframes` не похоже на любые другие свойства каскадных таблиц стилей, фактически это вообще не свойство, а *правило* и намного сложнее

обычного свойства. Анимации нужно присвоить имя (которое позже будет использоваться для применения анимации к элементу на странице), а затем указать группу фигурных скобок: { }. Внутри скобок находятся ключевые кадры, в качестве которых могут быть два ключевых слова — from и to — для обозначения первого и последнего ключевого кадра соответственно. Каждый ключевой кадр имеет собственный набор фигурных скобок, внутри которого помещаются анимируемые свойства: background-color, font-size, позиция на странице и т. д. Более подробную информацию о работе анимации можно найти в главе 10.

Пример:

```
@keyframes myAnimation {
  from {
    background-color: black;
  }

  to {
    background-color: white;
  }
}
```

animation

Это сокращенный метод применения анимации к элементу. Он является лаконичным способом применения следующих свойств в одном: animation-name, animation-duration, animation-timing-function, animation-iteration-count, animation-direction, animation-delay и animation-fill-mode. Все эти свойства рассматриваются далее в этом приложении. При использовании данных свойств в браузере Safari необходимо использовать вендорный префикс. Кроме того, они не поддерживаются в браузере Internet Explorer 9 и более ранних версиях.

Значения: список значений, разделенных пробелами, включающий свойства анимации, приведен выше. Конкретные типы значений, предоставляемые каждому из свойств, будут рассмотрены в соответствующих разделах приложения — animation-name, animation-duration и т. д. Обязательными являются только два свойства — animation-name и animation-duration. К одному и тому же элементу можно применить несколько именованных анимаций, предоставив список значений анимаций с запятой в качестве разделителя.

Примеры:

```
animation: myAnimation 2s;
animation: myAnimation 2s ease-in 2 alternate 5s forwards;
animation: fadeOut 2s ease-in-out 2 alternate 5s forwards,
  glow 5s;
```

animation-name

Это свойство используется для назначения анимации, созданной с помощью правила @keyframes. Это свойство добавляется в качестве части имени CSS-селектора, применяемого к одному или нескольким элементам страницы. Например, добавление

этого свойства к селектору тега `h1` сообщит браузеру, что нужно при загрузке страницы запустить указанную анимацию в отношении всех элементов `h1`. При этом нужно также назначить свойство `animation-duration`. При использовании этого свойства в браузере Safari необходимо использовать вендорный префикс. Кроме того, оно не поддерживается в браузере Internet Explorer 9 и более ранних версиях.

Значения: имя из правила `@keyframes`.

Пример: `animation-name: myAnimation;`

animation-duration

Определяет длительность анимации, обозначенной значением свойства `animation-name`.

Значения: значение в секундах — `1s` или в миллисекундах — `100ms`.

Пример: `animation-duration: 2s;`

animation-timing-function

Задаёт скорость анимации в течение выделенного ей периода. Например, установив длительность перехода 5 секунд, можно также управлять тем, как проигрывается переход в течение этих 5 секунд, например медленный старт и быстрый финиш. При использовании этого свойства в браузере Safari необходимо использовать вендорный префикс. Кроме того, оно не поддерживается в браузере Internet Explorer 9 и более ранних версиях.

Значения: одно из пяти ключевых слов: `linear`, `ease`, `ease-in`, `ease-out` и `ease-in-out`. Для настройки тайминга анимации вручную можно также применять значение кубической кривой Безье.

Примеры:

`animation-timing-function: ease-out;`

`animation-timing-function: cubic-bezier(.20, .96, .74, .07);`

animation-delay

Определяет время отсрочки начала воспроизведения анимации в секундах или миллисекундах. При использовании этого свойства в браузере Safari необходимо использовать вендорный префикс. Кроме того, оно не поддерживается в браузере Internet Explorer 9 и более ранних версиях.

Значения: значение в секундах — `1s` или в миллисекундах — `100ms`.

Пример: `animation-delay: 1.5s;`

animation-iteration-count

Определяет количество запусков анимации. Обычно анимация запускается один раз, а затем останавливается, но вы можете заставить анимацию запускаться 4, 5, 100 или бесконечное количество раз. При использовании этого свойства в браузере Safari необходимо использовать вендорный префикс. Кроме того, оно не поддерживается в браузере Internet Explorer 9 и более ранних версиях.

Значения: положительное целое число или ключевое слово `infinite`.

Примеры:

```
animation-iteration-count: 5;  
animation-iteration-count: infinite;
```

animation-direction

Если анимация должна воспроизводиться более одного раза, это свойство определяет направление воспроизведения для каждого последующего повтора. Обычно браузер заново проигрывает анимацию с самого начала. Но можно также воспроизвести анимацию в нормальном режиме, затем в обратном, а потом снова в нормальном. Например, можно создать анимацию, постепенно превращающую цвет фона из белого в черный, затем из черного в белый, затем вновь белый в черный и т. д. При использовании этого свойства в браузере Safari необходимо использовать вендорный префикс. Кроме того, оно не поддерживается в браузере Internet Explorer 9 и более ранних версиях.

Значения: ключевое слово `normal` или `alternate`. Обычно браузер проигрывает анимацию в режиме `normal`, поэтому данное свойство применяется только при необходимости использования ключевого слова `alternate`.

Пример: `animation-direction: alternate;`

animation-fill-mode

Определяет форматирование анимируемого элемента в начале и (или) в конце анимации. При использовании этого свойства в браузере Safari необходимо использовать вендорный префикс. Кроме того, оно не поддерживается в браузере Internet Explorer 9 и более ранних версиях.

Значения: одно из трех ключевых слов: `backwards`, `forwards` или `both`. Ключевое слово `forwards` применяется наиболее часто, поскольку оно оставляет элемент в форматировании, используемом в конце анимации вместо возвращения к стилю, который был до начала анимации.

Пример: `animation-fill-mode: backwards;`

animation-play-state

Управляет воспроизведением анимации. Это свойство можно использовать с псевдоклассом `:hover` для приостановки анимации при установке указателя мыши поверх элемента. При использовании этого свойства в браузере Safari необходимо указать вендорный префикс. Кроме того, оно не поддерживается в браузере Internet Explorer 9 и более ранних версиях.

Значения: одно из двух ключевых слов: `pause` или `running`. Ключевое слово `paused` приостанавливает анимацию, а ключевое слово `running` — продолжает воспроизведение. По умолчанию используется ключевое слово `running`.

Пример: `animation-play-state: paused;`

transform

Приводит к изменению элемента одним или несколькими способами, включая масштабирование, вращение, скашивание или перемещение. При использовании этого свойства в браузере Safari необходимо использовать вендорный префикс. Кроме того, оно не поддерживается в браузере Internet Explorer 9 и более ранних версиях.

Значения: ключевые слова `rotate()`, `translate()`, `skew()` или `scale()`. Каждое ключевое слово использует значение определенного типа. Например, `rotate()` требует указания значения угла вращения — `180deg`, `translate()` — значения в процентах, единицах `em` или пикселах, `skew()` — два значения угла, а `scale()` — положительное или отрицательное число. К одному и тому же элементу можно применить более одного вида преобразования.

Примеры:

```
transform: rotate(45deg);
transform: scale(1.5);
transform: skew(45deg 0) rotate(200deg) translate(100px, 0) scale(.5);
```

transform-origin

Определяет позицию, в которой происходит преобразование. Например, обычно при вращении элемента за ось вращения берется центр элемента. Но его можно также вращать вокруг одного из его четырех углов.

Значения: два значения, одно для координаты исходной точки по горизонтали, а другое — по вертикали. Можно использовать те же ключевые слова и значения, что и для свойства `background-position`.

Примеры:

```
transform-origin: left top;
transform-origin: 0% 100%;
transform-origin: 10px -100px;
```

transition

Лаконичный способ определения свойств `transition-property`, `transition-duration`, `transition-timing-function` и `transition-delay` (рассматриваемых ниже).

Свойство `transition` анимирует изменения свойств каскадных таблиц стилей элемента. Например, можно анимировать изменение фонового цвета кнопки навигации с красного на зеленый при установке поверх нее указателя мыши.

Значения: список свойств с пробелом в качестве разделителя, включающий свойства `transition-property` (необязательное, по умолчанию присвоено значение `all`), `transition-duration` (обязательное), `transition-timing-function` (необязательное, по умолчанию присвоено значение `ease`), `transition-delay` (необязательное, по умолчанию присвоено значение `0`).

Пример: `transition: background-color 1.5s ease-in-out 500ms;`

transition-property

Определяет свойства каскадных таблиц стилей, подвергаемые анимации при изменении форматирования элемента.

Значения: анимируемое свойство или ключевое слово `all`.

Примеры:

```
transition-property: width, left, background-color;  
transition-property: all;
```

transition-duration

Определяет длительность анимации перехода.

Значения: значение в секундах — `1s` или в миллисекундах — `1000ms`.

Пример: `animation-duration: 2s;`

transition-timing-function

Задаёт скорость анимации перехода в течение определенного периода. Например, установив длительность перехода 5 секунд, можно также управлять тем, как проигрывается переход в течение этих 5 секунд, например медленный старт и быстрый финиш.

Значения: одно из пяти ключевых слов: `linear`, `ease`, `ease-in`, `ease-out` или `ease-in-out`. Для настройки тайминга перехода вручную можно также применять значение кубической кривой Безье.

Примеры:

```
transition-timing-function: ease-out;  
transition-timing-function: cubic-bezier(.20, .96, .74, .07);
```

transition-delay

Определяет время отсрочки начала перехода в секундах или миллисекундах.

Значения: значение в секундах — `1s` или в миллисекундах — `1000ms`.

Пример: `transition-delay: 1.5s;`

Свойства таблицы

Существует несколько свойств каскадных таблиц стилей, которые относятся исключительно к HTML-таблицам. В главе 10 можно найти подробные инструкции по использованию каскадных таблиц стилей с таблицами.

border-collapse

Определяет, схлопываются ли границы ячеек таблицы. Если они не схлопываются, браузеры добавляют пространство размером несколько пикселей между каждой

ячейкой. Даже если вы удалите это пространство, присвоив значение 0 атрибуту `cellspacing` HTML-элемента `table`, браузеры будут отображать сдвоенные границы. Таким образом, граница каждой ячейки отобразится рядом с границей соседней ячейки, что вызовет визуальное удвоение линий границ. Присвоение значения `collapse` свойству `border-collapse` устраняет и промежуток между ячейками, и удвоение границ (см. раздел «Форматирование таблиц» главы 11). Свойство применимо только к элементу `table`.

Значения: `collapse`, `separate`.

Пример: `border-collapse: collapse;`

border-spacing

Устанавливает расстояние между ячейками в таблице. Используется взамен HTML-атрибута `cellspacing` элемента `table`. Однако браузер Internet Explorer версии 7 и ниже не поддерживает свойство `border-spacing`, поэтому лучше продолжать использовать атрибут `cellspacing` в элементах `table`, чтобы гарантировать отображение пространства между ячейками во всех браузерах.

ПРИМЕЧАНИЕ

Если вы хотите удалить пространство, которое браузеры по умолчанию добавляют между ячейками, присвойте значение `collapse` свойству `border-collapse`.

Значения: одно значение устанавливает одновременно расстояние по вертикали и горизонтали между границами ячеек. Если значений два, то первое определяет горизонтальное расстояние, а второе — вертикальное.

Пример: `border-spacing: 0 10px;`

caption-side

Если свойство относится к заголовку таблицы, оно определяет, появится заголовок сверху или снизу таблицы. Поскольку согласно правилам языка HTML элемент `caption` должен указываться сразу после открывающего тега `table`, заголовок обычно появляется в верхней части таблицы.

Значения: `top`, `bottom`.

Пример: `caption-side: bottom;`

empty-cells

Определяет, как браузер должен отображать пустые ячейки таблицы. В HTML-коде это выглядело бы следующим образом: `<td></td>`. Значение `hide` скрывает любую часть ячейки, оставляя пустое пространство, при этом границы, фоновые цвета и изображения не отображаются в пустой ячейке. Применяйте это свойство в стиле, формирующем элемент `table`.

Значения: `show`, `hide`.

Пример: `empty-cells: show;`

ПРИМЕЧАНИЕ

Свойство `empty-cells` не поддерживается в браузере Internet Explorer 7 и более ранних версиях.

table-layout

Управляет тем, как браузер создает таблицу, и может немного влиять на скорость отображения страницы браузером. Присвоение значения `fixed` вынуждает браузер привести все столбцы к той же ширине, что задана для столбцов из первой строки, из-за чего таблица визуализируется быстрее. Значение `auto` используется по умолчанию, при нем браузер автоматически визуализирует таблицу, поэтому, если вас устраивает скорость визуализации таблиц на странице, не беспокойтесь об этом свойстве. Если же вы используете его, то применяйте к стилю, форматирующему элемент `table`.

Значения: `auto`, `fixed`.

Пример: `table-layout: fixed;`

Прочие свойства

Каскадные таблицы стилей предлагают и другие дополнительные и зачастую интересные свойства. Они позволяют улучшать веб-страницы, задавая специальный контент и курсоры, управляют версиями страницы для вывода на печать и т. д.

content

Определяет текст, который появляется либо до, либо после элемента. Используйте это свойство с псевдоэлементами `:after` и `:before`. Вы можете добавить открывающую кавычку перед цитируемым материалом и закрывающую кавычку после него.

Значения: текст в кавычках ("как этот"), ключевые слова `normal`, `open-quote`, `close-quote`, `no-open-quote`, `no-close-quote`. А также значение HTML-атрибута.

Примеры:

```
p.advert:before { content: "И теперь слово спонсору..."; }
a:after { content: " (" attr(href) ") "; }
```

ПРИМЕЧАНИЕ

Добавление текста таким способом (как пример с открывающими и закрывающими кавычками) называют генерируемым контентом. Подробные сведения вы найдете на сайтах tinymce.com/4qunt и tinymce.com/b37oc.

cursor

Позволяет изменять вид курсора, когда указатель мыши устанавливается поверх определенного элемента. Например, вы можете задать ему вид вопросительного знака, если указатель мыши находится поверх ссылки, предоставляющей дополнительную информацию по какой-либо теме.

Значения: auto, default, crosshair, pointer, move, e-resize, ne-resize, nwresize, n-resize, se-resize, sw-resize, s-resize, w-resize, text, wait, help, progress. Вы также можете использовать URL-адрес, чтобы использовать в качестве указателя другое изображение (см. примечание ниже). Указатель мыши, который находится поверх ссылки, выглядит как стрелка, поэтому, если вы хотите, чтобы какие-либо элементы на странице демонстрировали пользователю, что он может щелкнуть на них, добавьте в стиль объявление `cursor: pointer`.

Примеры:

```
cursor: help;  
cursor: url(images/cursor.cur);
```

ПРИМЕЧАНИЕ

Не все браузеры поддерживают URL-адреса изображений для указателя. Для получения более полной информации перейдите на сайт tinyurl.com/q7eqosv.

opacity

Позволяет управлять прозрачностью любого элемента и всех его потомков. При этом сквозь элемент могут просвечивать находящиеся на заднем плане цвета, изображения и контент. Учтите, что при применении свойства `opacity` к `div`-контейнеру тот же уровень прозрачности получают все содержащиеся в нем заголовки, изображения, абзацы и другие `div`-контейнеры. То есть, если элементу `div` присвоить свойство `opacity: .5`; (50%-ная прозрачность), изображение внутри этого контейнера также станет наполовину прозрачным, даже если непосредственно изображению будет присвоено свойство `opacity: 1`;

Значения: десятичное значение в диапазоне от 0 до 1. Значение 0 означает невидимость, а значение 1 — полную непрозрачность.

Пример: `opacity: .5`;

orphans

Определяет минимальное количество строк текста, которые можно оставить в нижней части выводимой на печать страницы. Предположим, вы печатаете страницу на лазерном принтере и абзац в пять строк разъехался на две страницы, причем всего одна строка находится в нижней части первой страницы, а четыре оставшиеся перенесены на вторую. Поскольку одна строка выглядит «висячей», вы можете дать инструкцию браузеру разбивать абзац, только если, скажем, по крайней мере три строки остаются в нижней части выводимой на печать страницы (на момент написания книги только браузер Орега поддерживал это свойство).

Значения: числа, например 1, 2, 3.

Пример: `orphans: 3`;

page-break-inside

Препятствует тому, чтобы элемент был разбит на две выводимые на печать страницы. Если вы хотите расположить фотографию и подпись к ней вместе на одной

странице, оберните их в один контейнер `div`, а затем примените к нему стиль со свойством `page-break-inside`.

Значения: `avoid`.

Пример: `page-break-inside: avoid;`

widows

Противоположность свойства `orphans`, описанного выше. Данное свойство определяет минимальное количество строк, которое должно появиться в верхней части выводимой на печать страницы. Скажем, принтер может поместить четыре из пяти строк абзаца в нижней части страницы и должен будет переместить последнюю строку в верхнюю часть следующей страницы. Чтобы предотвратить появление таких «висячих» строк, используйте свойство `widows`, инструктирующее браузер помещать не менее двух или трех строк в верхнюю часть выводимой на печать страницы.

Значения: числа, например 1, 2, 3.

Пример: `widows: 3;`

Приложение 2. Информационные ресурсы, посвященные CSS

К сожалению, одна книга не может ответить на все ваши вопросы о каскадных таблицах стилей. Однако, к счастью, существует много ресурсов по языку CSS, предназначенных как для начинающих, так и для опытных веб-разработчиков. В этом приложении вы найдете ссылки на те ресурсы, которые помогут вам разобраться с общими понятиями каскадных таблиц стилей и научат решать конкретные задачи, такие как создание навигационной панели или компоновка макета веб-страницы.

Справочники

Справочники по свойствам CSS бывают как официальные, так и нет. Конечно, существуют сайты и учебные онлайн-пособия, но вам не обязательно бороздить Всемирную паутину, чтобы узнать о CSS. Некоторые из этих справочников можно найти в «старомодном» бумажном варианте.

Консорциум W3C

Текущая работа CSS (tinyurl.com/n8xhw). Здесь находятся все спецификации CSS, включая самые новые дополнения. Можно щелкнуть на любой спецификации, чтобы получить ее подробное описание, при этом следует учитывать, что она может быть не полностью реализована во всех браузерах. Этот сайт — официальный источник информации о каскадных таблицах стилей.

Другие онлайн-справочники

- **Справочник по CSS в сети разработчиков Mozilla** (tinyurl.com/og7x6jt). Сеть разработчиков Mozilla (MDN) предоставляет один из самых полных справоч-

ников по каскадным таблицам стилей (а также по HTML5, JavaScript и другим веб-технологиям).

- **Еще один справочник по CSS** (tinyurl.com/km7hdmh) содержит обширные и подробные описания для большинства свойств каскадных таблиц стилей. Включает в себя множество примеров использования свойств, а также подробные описания с примечаниями.
- **Can I use...** (caniuse.com). Этот часто обновляемый сайт предоставляет подробную информацию по совместимости браузеров со свойствами CSS. Здесь можно определить, будет ли то или иное свойство CSS работать, к примеру, в Internet Explorer 9.
- **Ресурсы CSS3files** (css3files.com) предоставляет подробные инструкции и прекрасные демонстрации работы наиболее популярных свойств CSS3, таких как анимация, тени, градиенты и многое другое. Кроме того, публикует статьи по теме последних достижений в области каскадных таблиц стилей.

Справочная информация по CSS

Даже при наличии замечательных справочников (таких как эта книга) иногда приходится обращаться за помощью к знатокам. Вы можете зарегистрироваться на одном из ресурсов, где специалисты по каскадным таблицам стилей отвечают на вопросы по электронной почте, либо внимательно просмотреть огромное количество информации на тематическом форуме.

Форумы

- **Stack Overflow** (stackoverflow.com). Один из лучших ресурсов, посвященный Всемирной паутине и программированию. Тысячи специалистов отвечают на вопросы посетителей из области вычислительной техники. Чтобы задать вопрос, посвященный CSS, а также найти вопросы по теме других пользователей, перейдите по адресу tinyurl.com/lhc6q7.
- **CSSCreator** (tinyurl.com/ndexl). Оживленный форум, предлагающий помощь и советы по всем темам, начиная с базовой CSS-верстки и заканчивая усовершенствованными методами.
- **SitePoint** (tinyurl.com/o7287qr). Другая полезная группа CSS-гуру.
- **CSS-Tricks** (tinyurl.com/o3o5z9z). На этом небольшом форуме можно найти нужную информацию по CSS. Если вы программируете на PHP или JavaScript, то также найдете здесь много полезного для себя.

Подсказки, приемы и советы по CSS

Всемирная паутина превращает каждого во владельца сайта и, соответственно, автора контента. И при таком обилии ресурсов тяжело перебирать весь контент

и находить в нем понятную, краткую и точную информацию. Во Всемирной паутине есть множество сайтов, посвященных каскадным таблицам стилей. Ниже приведены лучшие из них.

- **CSS-Tricks** (css-tricks.com). Этот блог, который ведет всего один человек, полон прекрасных советов по использованию каскадных таблиц стилей. Вы найдете здесь часто обновляющиеся советы и рекомендации, а также исчерпывающие видеоуроки.
- **Sitepoint** (tinyurl.com/nfzjw4m). Содержит множество статей и руководств по технологии CSS-верстки. Здесь также часто появляются самые последние новости из мира каскадных таблиц стилей.
- **Smashing Magazine** (tinyurl.com/yf5th8n). Здесь собраны одни из лучших тематических ресурсов во Всемирной паутине, а в категории CSS вы найдете практически бесконечное количество ссылок, освещающих самые креативные подходы к применению каскадных таблиц стилей для создания веб-дизайна.

CSS-навигация

В главе 9 было показано, как с нуля создавать навигационные кнопки для сайта. И учебные онлайн-пособия — отличный способ закрепить знания. Кроме того, как только вы поймете весь процесс в подробностях, вам не нужно будет делать это самостоятельно каждый раз. Во Всемирной паутине вы можете найти примеры панелей навигации, которые могут вдохновить вас на новые свершения.

Учебные пособия

- **Listutorial** (tinyurl.com/hw3jx). Пошаговые пособия по созданию навигационных систем из неупорядоченных списков.
- **Адаптивные панели навигации с помощью CSS** (tinyurl.com/p9bepgy). За этим названием скрывается учебник, содержащий пошаговые инструкции для создания адаптивной панели навигации.
- **Как создать адаптивное навигационное меню с помощью CSS** (tinyurl.com/oguucsu). Из этой статьи вы можете узнать, как использовать CSS-код для создания адаптивного навигационного меню.

Онлайн-примеры

- **NavNav** (navnav.co) содержит примеры панелей навигации, презентации и учебники. На сайте приведены примеры использования CSS- и JavaScript-кода для создания захватывающих систем навигации.
- **5 шаблонов навигационных меню для мобильных устройств** (tinyurl.com/p27cr4d). Изучите различные шаблоны навигационных меню для мобильных устройств.

- **Скрытые навигационные меню** (<http://tinyurl.com/kvj5qw5>). Какое преимущество дает скрытая панель навигации? Она экономит пространство на экране. Этот учебник покажет вам, как создать меню, которое открывается нажатием кнопки. И все это достигается с помощью CSS-кода!
- **Шаблоны навигационных меню и приоритеты** (tinyurl.com/n9fuysk). Настройка навигационного меню в зависимости от размера экрана.

CSS-верстка

CSS-верстка настолько гибка и удобна для применения, что можно потратить всю жизнь, исследуя ее возможности. И некоторые люди, кажется, делают только это. Вы можете извлечь пользу из их трудов, читая статьи, изучая онлайн-примеры и экспериментируя с инструментами, которые могут сделать некоторую работу с каскадными таблицами стилей за вас.

- **Изучение CSS разметки** (tinyurl.com/o33nblg). Содержит русскоязычные интерактивные уроки, обучающие различным способам CSS-верстки.
- **Бесплатные адаптивные CSS-макеты** (tinyurl.com/ofb3th2). Этот сайт содержит базовые HTML- и CSS-шаблоны, необходимые для верстки макета страницы. Дизайны адаптивные (глава 15), поэтому подстраиваются под размер экрана устройства посетителя.
- **Pure-макеты** (tinyurl.com/msapspz) содержит множество примеров базовых шаблонов, созданных на основе фреймворка Pure.
- **Twitter Bootstrap** (getbootstrap.com). Сайт с различными инструментами. Содержит HTML-, CSS- и JavaScript-компоненты, которые позволяют легко создать законченную адаптивную, основанную на модульной сетке страницу с JavaScript-сценариями.
- **Foundation** (foundation.zurb.com). Еще один ресурс, посвященный инструментам веб-дизайна. Начальная страница очень похожа на страницу Twitter. Сайт содержит HTML-, CSS- и JavaScript-инструменты, а также много документации, благодаря которой относительно легко учиться верстать.

Демонстрационные сайты

Доскональное знание стандарта CSS не поможет, если ваше воображение не работает. Отличным источником вдохновения может стать творческая работа других людей. С помощью поисковых систем вы найдете множество CSS-сайтов, а ниже перечислены некоторые из ресурсов, на которых вы можете оценить красочные CSS-дизайны.

- **CSS ZenGarden** (csszengarden.com). Самый главный сайт, посвященный CSS-верстке: много различных дизайнов для одного и того же HTML-кода.
- **CSS Line** (cssline.com). Замечательная галерея вдохновляющих CSS-дизайнов с инновационной системой фильтров. Интересуетесь сайтами, которые используют

определенный цвет? Вы можете просматривать только их. Или, если вы хотите увидеть сайты, обладающие конкретными функциями, например уникальной системой навигации и типографикой, можно отфильтровать только их.

- **The Awwwards** (awwwards.com). Сайт, обладающий прекрасным дизайном, несмотря на свое название.
- **CSS Design Awards** (cssdesignawards.com). Этот сайт ежедневно отмечает «победителей». Неизвестно, кто представляет жюри, но сайт отмечает самые прекрасно оформленные сайты.

Указатель

- CSS, 16
 - история, 45
 - помощь, 707
 - проверка правильности, 52
 - ресурсы, 706
- doctype, объявление типа документа, 43
- Dreamweaver, 21
- em, единица измерения, 671
 - для кнопок, 347
 - размер шрифта, 178
- Firefox
 - overflow, свойство, 236
 - отступы, 196
 - проверка HTML-код, 40
- Internet Explorer
 - измерение в пикселях, 671
 - настройка списков, 197
- JavaScript
 - ID-селекторы, 71
 - динамические меню, 328
- Opera
 - padding, свойство, 214
- Safari
 - отступы, 196
- URL, 673
- XHTML, 19
- Атрибуты, 19
 - cellspacing
 - в таблице, 392
 - class, 69
 - id, 72
 - src, 196
 - title, 84
- Баннер, 263, 483
- Блочная модель, 211
 - блочные (прямоугольные) элементы, 219
 - встроенные (inline) элементы, 219
- Боковое меню
 - закругление углов, 306
 - на всю высоту, 442
 - создание, 251
- Всплывающие меню, 328
- Выравнивание таблиц, 391
 - текста, 189
- Вычисление ширины/высоты, 232
- Генерируемое содержимое, 82
- Генерируемый контент, 197
- Границы, 221
 - вокруг плавающего элемента, 242
 - замена изображениями, 302
 - отступы, 225
 - плавающие элементы, 241
 - удаление, 611
 - форматирование, 223
- Графика, 257
 - для ссылок, 318
 - предварительная загрузка, 330
- Значения, 48, 668
 - HTML, 19
 - URL, 673
 - десятичные (RGB), 669
 - длины и размеры, 670
 - ключевые слова, 672
 - пиксели, 671
 - проценты, 669
 - цвета, 668

- Изображения, 257
 - бесплатные примеры, 277
 - заключение в рамку, 291
 - как ссылки, 318
 - надписи, 486
 - обучающий урок, 290
 - повтор, 263
 - предварительная загрузка, 330
 - удаление границ, 611
- Каскадность, 120
 - и наследование, 121
 - несколько стилей, 123
 - обучающий урок, 135
 - особенности, 125
 - состояния ссылки, 311
- Каскадные таблицы стилей (Cascading Style Sheets), 16
- Кнопки, 315
 - навигационные, форматирование, 617
 - центрирование текста, 328
- Комментарии, 600
- Контрастность, 180
- Конфликт полей, 217, 684
- Кэш, 49, 50
- Маркеры
 - графические, 196
 - диск, 193
 - квадрат, 193
 - окружность, 193
 - удаление, 321
- Навигация, 309
 - обучающие примеры, 708
 - панели навигации, 320, 618
 - вертикальные, 322, 340
 - всплывающие меню, 328
 - горизонтальные, 324, 346
 - границы, 324
- Наследование, 110, 121
 - в таблицах стилей, 111
 - значимость, 126
 - ключевых слов, 672
 - обучающий урок, 114
 - ограничения, 112
 - преимущества непосредственно примененного стиля, 123
 - размер шрифта, 177, 179
- Обтекание содержимого
 - порядок написания HTML-кода, 241
 - фоны и границы, 241
- Объявления, 48
 - блок объявления, 46, 48
- Основной (базовый) размер шрифта текста, 176
- Ошибки
 - Firefox
 - позиционирование фоновых рисунков, 268
- Перезагрузка страницы,
 - принудительная, 50
- Плавающие элементы, 239, 414
 - clear, свойство, 242
 - внутри плавающих элементов, 433
 - горизонтальная панель навигации, 327
 - границы, 242
 - перепады, 446
 - ширина, 429
- Подписи, 293, 486
- Позиционирование, 462
 - position, свойство, ключевые слова, 465
 - абсолютное, 463, 477
 - float, свойство, 463
 - visibility, свойство, 474
 - наложение элементов (z-index), 473
 - родственные отношения, 468
 - внутри элемента, 477
 - горизонтальное/вертикальное смещение, 319
 - наложение элементов, 471
 - обучающий урок, 483
 - относительное, 468
 - свойства, 462
 - сокрытие частей страницы, 474
 - стратегии, 476
 - фиксированное, 463
 - фреймы, 479
 - фреймы, 479
- Потомки, отношения тегов, 76

- Проверка кода
CSS, 52
HTML, 40
- Псевдоклассы
active, 80, 310
focus, 81
hover, 80, 310, 314
 изображения, 320
 кнопки, 317
link, 80, 311
visited, 80, 310
 изображения, 320
значимость, 126
- Псевдоэлементы
after, 82
before, 82
first-child, 86
first-letter, 81, 193
first-line, 81, 193
форматирование абзацев, 193
- Путь
абсолютный, 265
корневой относительный, 265
относительный, 60
относительный от документа, 266
- Разметка
на основе плавающих элементов, 427, 667
 непостоянная ширина, 415, 431
 фиксированная ширина, 415, 431
предустановленная, 432
с фиксированной шириной, 414
 создание из свободной, 431
- Редакторы
EditPlus, 21
skEdit, 21
- Родственные отношения (наследование),
76, 110, 119, 121, 122, 123, 126
- Свойства, 48
background, 276, 686
 границы, 226
 кнопки, 315
 плавающие элементы, 242
background-attachment, 271, 686
 fixed, значение, 271
 scroll, значение, 271
 сокращенный вариант, 276
background-color, 687
 кнопки, 315
 сокращенный вариант, 276
background-image, 257, 687
background-position, 267, 318, 688
 ключевые слова, 264
 повторное отображение рисунков, 263
 процентные значения, 269
 сокращенный вариант, 276
 точные значения, 268
background-repeat, 263, 688
border, 223, 680
 изображения, 257
 кнопки, 315
 подчеркивание ссылок, 313
 таблицы, 392
 фон, 225
border-collapse, 392, 701
 значения, 393
border-color, 681
border-spacing, 702
border-style, 682
border-width, 682
bottom, 689
caption-side, 702
clear, 242, 690
 значения, 244
 колоннитулы, 435
 плавающие элементы, 435
clip, 690
color, 171, 668, 674
 значения, 172, 674
content, 703
cursor, 703
display, 221, 691
 none, значение, 221
 в сравнении со свойством visibility, 474
 горизонтальные панели навигации, 324
empty-cells, 702
float, 239, 427, 691
 изображения, 258
 ключевые слова, 239
 отмена свойством clear, 242

- порядок исходного кода, 241, 428
 - предотвращение перепадов, 446
 - фоны и границы, 241
 - font, 190, 674
 - font-family, 674
 - font-size, 175, 675
 - наследование, 178, 179
 - font-style, 181, 675
 - font-variant, 182, 675
 - font-weight, 181, 675
 - height, 232, 691
 - вычисление фактической высоты, 232
 - и свойство overflow, 236
 - left, 692
 - letter-spacing, 184, 676
 - line-height, 187, 676
 - встроенные элементы, 219
 - наследование, 188
 - list-style, 197, 679
 - list-style-image, 196, 679
 - list-style-position, 195, 679
 - list-style-type, 193, 679
 - margin, 189, 196, 611, 684
 - в списках, 322
 - конфликты полей, 216, 684
 - обучающий урок, 244
 - отрицательные значения, 218
 - размеры полей, 214
 - сокращенный набор, 215
 - max-height и max-width, 692
 - orphans, 704
 - outline, 683
 - overflow, 236, 693
 - auto, ключевое слово, 236
 - hidden, ключевое слово, 236, 241
 - scroll, ключевое слово, 236
 - visible, ключевое слово, 236
 - padding, 196, 685
 - в списках, 322
 - встроенные элементы, 219, 320
 - в таблицах, 390
 - для границ, 225
 - для изображений, 257, 320
 - для кнопок, 315
 - размеры отступов, 214
 - сокращенный набор, 215
 - page-break-inside, 704
 - position, 694
 - right, 694
 - table-layout, 703
 - text-align, 189, 676
 - в таблицах, 390
 - text-decoration, 182, 676
 - подчеркивание ссылок, 314
 - text-indent, 189, 677
 - text-transform, 182, 677
 - top, 695
 - vertical-align, 391, 677
 - visibility, 474, 695
 - widows, 705
 - width, 233, 323
 - в таблицах, 395
 - вычисление фактической ширины, 232
 - ширина столбцов, 441
 - word-spacing, 184, 678
 - z-index, 473, 696
 - наследование, 110
 - позиционирование, 318
- Селекторы, 46
- ID, 71
 - # (символ решетки), 102
 - в сравнении с селекторами классов, 418
 - обучающий урок, 100
 - с использованием JavaScript, 71
 - атрибутов, 84
 - [] (квадратные скобки), 84
 - групповые, 73, 618
 - обучающий урок, 99
 - дочерних элементов
 - угловая скобка (>), 86
 - форматирование списков, 92
 - классов, 67
 - . (точка), 68
 - в сравнении с ID-селекторами, 418
 - обучающий урок, 104
 - правила именования, 68
 - потомков, 77, 614
 - группирование ссылок, 312

- изображения, 258
- обучающий урок, 106
- универсальный (*), 74
- Списки
 - маркированные, 193, 268, 278, 304, 321
 - нумерованные, 195
 - свойства, 678
 - создание, 91
 - форматирование, 92, 179, 194, 203
- Ссылки, 309
 - внешние
 - выделение (обучающий урок), 338
 - стилизация, 313
 - группирование с помощью селекторов потомков, 312
 - использование изображений, 318
 - навигационные панели, 320
 - обучающий урок, 334
 - печать, 532
 - подчеркивание, 313
 - состояния, 309
 - стилизация, 80, 313
 - центрирование текста на кнопках, 328
- Стили, 46
 - встроенные, 53
 - группирование, 607
 - каскадность, 120
 - комментарии, добавление, 600
 - конфликт, 114
 - наследование, 111
 - обучающий урок, 53
 - определение, какой принимать, 128
 - организация, 602
 - размещение, 53
 - ссылки, 80, 309
 - форматирование, 48
- Столбцы
 - добавление в формы, 398
 - плавающие элементы, 428
 - разметка с множеством столбцов (обучающий урок), 449
 - расчет ширины, 447
 - ресурсы, 709
 - стилизация, 395
- Таблицы, 387
 - выравнивание, 390
 - границы, 392
 - отступы, 390
 - свойства, 701
 - стилизация, 389
 - обучающий урок, 401, 544
 - чередование строк, 394
- Таблицы стилей, 49
 - внешние, 49, 52, 58
 - @import, правило, 608
 - размещение, 128
 - связывание с веб-страницей, 59
 - внутренние, 49, 50
 - перевод во внешние, 51, 56
 - организация, 602
- Теги
 - <div>, 33
 - как элемент-контейнер, 215
 - организация таблиц стилей, 615
 - , 257
 - для печати, 271
 - поля/отступы, 220
 - , 34
 - позиционирование элемента, 477
- Текст
 - абзацы, 186, 201
 - отступ первой строки, 189
 - с выступающей строкой (выступ), 190
 - форматирование, 201
 - буквица, создание, 81
 - выравнивание, 189
 - заголовки, форматирование, 201
 - контрастность, 180
 - курсив, 181
 - малые прописные буквы, 182
 - наследование, 191
 - первая буква, первая строка абзаца, 191
 - полужирный, 181
 - прописные буквы, 182
 - свойства, 673
 - списки, 193, 203
 - текстовые редакторы, 20

- украшение, 182
 - мерцание, ключевое слово `blink`, 182
- форматирование
 - обучающий урок, 198
- цветовое оформление, 674
- шрифты, 142, 611
- Теневые эффекты, добавление, 264, 298, 299, 300
- Установка цвета фона, 225
- Фоновые изображения
 - `background-image`, свойства ссылки, 318
 - `background-image`, свойство URL, 259
 - добавление в ссылки, 313
 - обучающий урок, 300
 - печать, 271
 - повторение, 263
 - позиционирование, 264
 - свойство `background-image`, 258
 - обучающий урок, 300
 - стилизация ссылок (обучающий урок), 334
 - фиксация на месте, 271
- Формы
 - обучающий урок, 406
 - форматирование, 398
- Фреймы, 479
 - наборы фреймов, 480
- Цвета, 668
- Элементы
 - HTML-формы
 - кнопки, 397
 - переключатели, 397
 - поля ввода, 397
 - раскрывающиеся списки, 397
 - тег `fieldset`, 397
 - тег `legend`, 397
 - флажки, 397
 - блочные, 71, 219, 241
 - вложенные, 77
 - встроенные, 219, 241
 - наложение, 471
 - сестринские
 - соединение, + (плюс), 93
- Элементы-контейнеры, 214, 239
 - для плавающих элементов, 438

Дэвид Макфарланд

Новая большая книга CSS

Перевел с английского *С. Черников*

Заведующий редакцией
Ведущий редактор
Художник
Корректор
Верстка

*О. Сивченко
Н. Гринчик
С. Заматевская
Е. Павлович
А. Барцевич*

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —

Книги печатные профессиональные, технические и научные.

Подписано в печать 04.03.16. Формат 70×100/16. Бумага писчая. Усл. п. л. 58,050. Тираж 1000. Заказ 0000.

Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».

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

Сайт: www.chpk.ru. E-mail: marketing@chpk.ru

Факс: 8(496) 726-54-10, телефон: (495) 988-63-87



**ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР» предлагает:
профессиональную, популярную и детскую нон-фикшн литературу**

Заказать книги оптом можно в наших представительствах:

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: voronej@piter.com

Екатеринбург: ул. Толедова, д. 43а; тел./факс: (343) 378-98-41, 378-98-42;
e-mail: office@ekat.piter.com; skype: ekat.manager2

Нижний Новгород: тел.: 8 930 712-75-13; e-mail: yashny@yandex.ru; skype: yashny1

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 229-68-09; e-mail: pitvolga@mail.ru


УКРАИНА

Киев: Московский пр., д. 6, корп. 1, офис 33
тел./факс: (044) 490-35-69, 490-35-68; e-mail: office@kiev.piter.com

Харьков: тел./факс: +38 067 545-55-64; e-mail: sasha@kharkov.piter.com

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163; тел./факс: +37 517 208-80-01;
e-mail: gv@minsk.piter.com

 **Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок**
тел./факс: (812) 703-73-73; e-mail: sales@piter.com

 **Издательский дом «Питер» приглашает к сотрудничеству авторов**
тел./факс: (812) 703-73-72, (495) 234-38-15

 **Заказ книг для вузов и библиотек**
тел./факс: (812) 703-73-73, гоб. 6243; e-mail: uchebnik@piter.com

 **Заказ книг по почте:** на сайте www.piter.com; тел.: (812) 703-73-74, гоб. 6216;
e-mail: books@piter.com

 **Вопросы по продаже электронных книг:** тел.: (812) 703-73-74, гоб. 6217;
e-mail: kuznetsov@piter.com





КНИГА-ПОЧТОЙ



ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **books@piter.com**
- по телефону: **(812) 703-73-74**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

-  Наложным платежом с оплатой при получении в ближайшем почтовом отделении.
-  С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
-  Электронными деньгами. Мы принимаем к оплате: Яндекс.Деньги, Webmoney и Kiwi-кошелек.
-  В любом банке, распечатав квитанцию, которая формируется автоматически после совершения вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ:

- Посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку ваших покупок максимально быстро. Дату отправления вашей покупки и дату доставки вам сообщат по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте www.piter.com)
- Можно оформить доставку заказа через почтоматы, (адреса почтоматов можно узнать на нашем сайте www.piter.com)

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, факс, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

ВАША УНИКАЛЬНАЯ КНИГА

Хотите издать свою книгу? Она станет идеальным подарком для партнеров и друзей, отличным инструментом для продвижения вашего бренда, презентом для памятных событий! Мы сможем осуществить ваши любые, даже самые смелые и сложные, идеи и проекты.

МЫ ПРЕДЛАГАЕМ:

- издать вашу книгу
- издание книги для использования в маркетинговых активностях
- книги как корпоративные подарки
- рекламу в книгах
- издание корпоративной библиотеки

Почему надо выбрать именно нас:

Издательству «Питер» более 20 лет. Наш опыт – гарантия высокого качества.

Мы предлагаем:

- услуги по обработке и доработке вашего текста
- современный дизайн от профессионалов
- высокий уровень полиграфического исполнения
- продажу вашей книги во всех книжных магазинах страны

Обеспечим продвижение вашей книги:

- рекламой в профильных СМИ и местах продаж
- рецензиями в ведущих книжных изданиях
- интернет-поддержкой рекламной кампании

Мы имеем собственную сеть дистрибуции по всей России, а также на Украине и в Беларуси. Сотрудничаем с крупнейшими книжными магазинами.

Издательство «Питер» является постоянным участником многих конференций и семинаров, которые предоставляют широкую возможность реализации книг.

Мы обязательно проследим, чтобы ваша книга постоянно имелась в наличии в магазинах и была выложена на самых видных местах.

Обеспечим индивидуальный подход к каждому клиенту, эксклюзивный дизайн, любой тираж.

Кроме того, предлагаем вам выпустить электронную книгу. Мы разместим ее в крупнейших интернет-магазинах. Книга будет сверстана в формате ePub или PDF – самых популярных и надежных форматах на сегодняшний день.

Свяжитесь с нами прямо сейчас:

Санкт-Петербург – Анна Титова, (812) 703-73-73, titova@piter.com

Москва – Сергей Клебанов, (495) 234-38-15, klebanov@piter.com