

# 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<sup>1</sup> модульной сетки*, которые легко группируются для создания *колонок контента* различной ширины. Распространенное число столбцов модульной сетки — 12, поскольку 12 легко де-

---

<sup>1</sup> В этой книге вертикальная единица модульной сетки называется столбцом вместо колонки, чтобы избежать путаницы с привычными колонками контента.

ится 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 пикселях будет изменяться, но его относительная ширина всегда будет равняться трем столбцам.

И в этом заключается прелест 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 ([getskeleton.com](http://getskeleton.com)). Но она не одна в своем роде.

- Simple Grid ([tinyurl.com/d3vwkuu](http://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](http://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](http://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](http://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 пикселов она сворачивает в одну колонку многоколоночной страницы. 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](http://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 <link href="https://fonts.googleapis.com/css?family=Roboto:400,300,700&subset=latin,Cyrillic"
rel="stylesheet" type="text/css">
```

## 2. Добавьте контейнеры `div`.

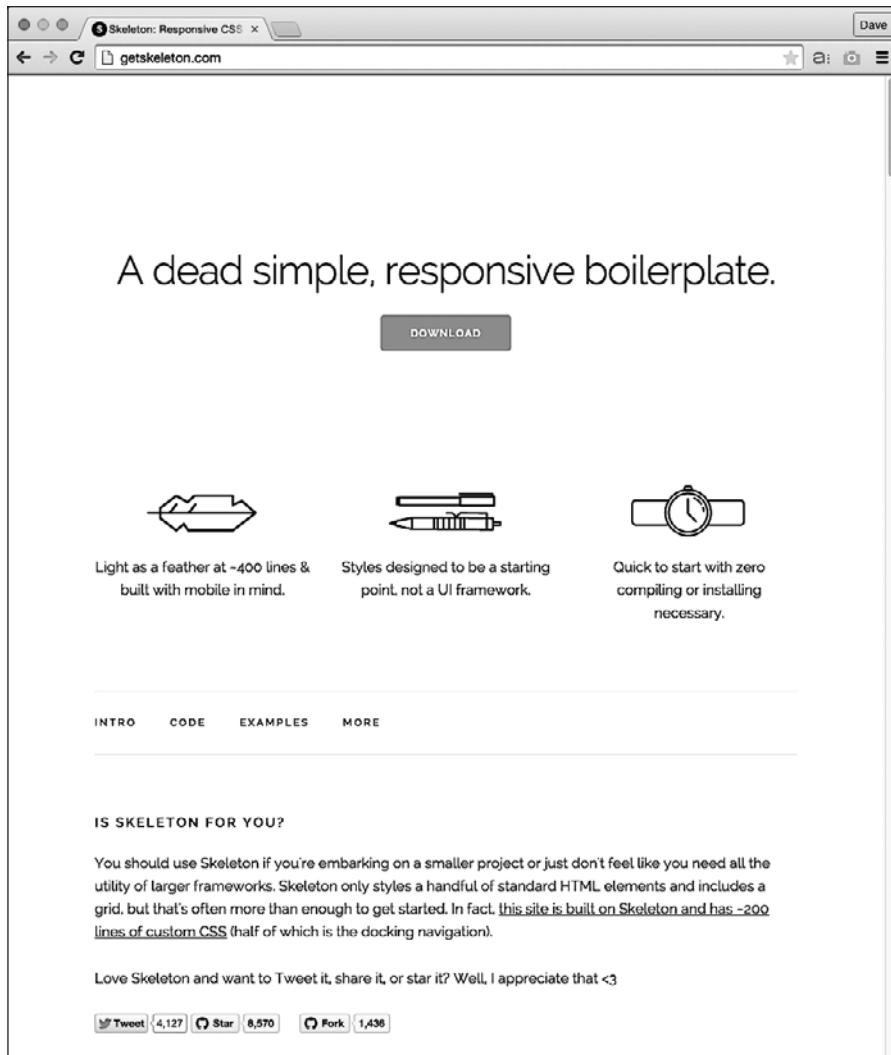
Как упоминалось ранее в этой главе, в модульной сетке контейнеры `div` используются для построения строк и колонок. Страница может иметь только один контейнер, но бывают случаи, когда для создания определенного эффекта необходимо несколько контейнеров (подробнее об этом читайте в следующем разделе). 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:

```
<div class="container">
  <div class="row">
    </div>
  <div class="row">
    </div>
  <div class="row">
    </div>
</div>
```

#### 4. Добавьте элементы div для колонок.

Система Skeleton основана на модульной сетке, состоящей из 12 столбцов, поэтому каждый из добавленных вами элементов div должен быть не менее одного (что очень мало) или не более 12 (полная ширина контейнера) столбцов в ширину. Кроме того, если вы используете более одной колонки, их общая ширина должна составлять 12 столбцов. Например, если у вас есть три колонки, каждая из них может быть шириной четыре столбца или, к примеру, одна колонка может быть шириной два столбца, в то время как две другие — по пять столбцов.

Вы определяете элементы div как колонки, добавив имя columns в качестве атрибута класса. Затем укажите ширину этой колонки, используя число в диапазоне от 1 до 12. Помните, что имена классов в CSS не могут начинаться с чисел, например 1, 2 или 12, поэтому вам необходимо писать числа словами. Например, чтобы создать три равные по ширине колонки, вы можете добавить три следующих элемента div:

```
<div class="container">
  <div class="row">
    <div class="four columns">
      </div>
    <div class="four columns">
      </div>
    <div class="four columns">
      </div>
    </div>
  </div>
</div>
```

#### COBET

Если вы хотите создать одну колонку в строке, не добавляйте элементы div внутри строки. Просто добавьте необходимый контент следующим образом:

```
<div class="row">
  <p>Этот текст заполняет всю ширину контейнера. Получается одна колонка.</p>
</div>
```

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">
```

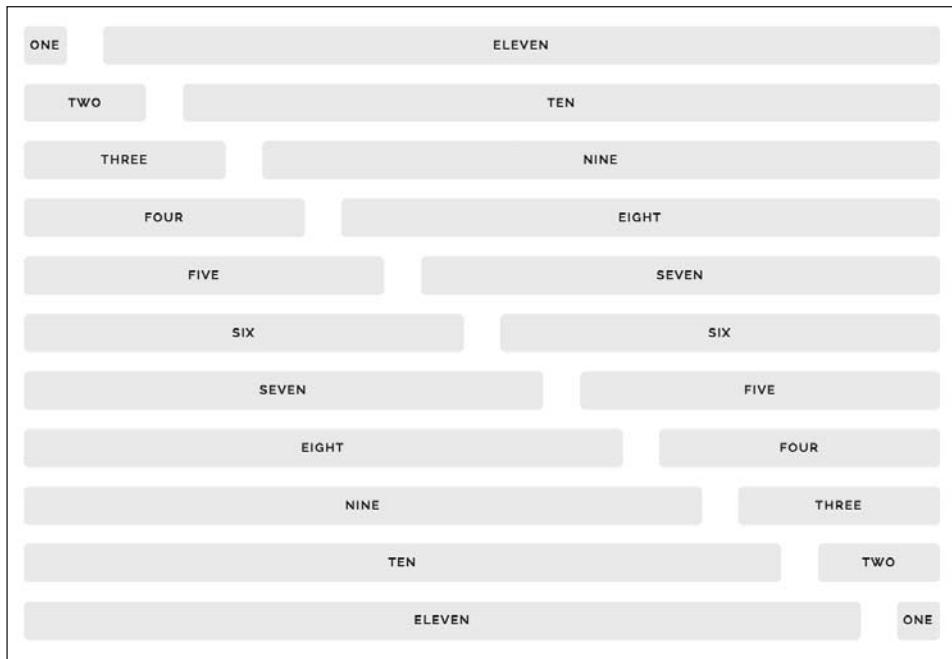
Система Skeleton в плане дизайна в первую очередь отдает предпочтение мобильным устройствам (см. раздел «Медиазапросы» главы 15), поэтому для подгонки дизайна под окна браузеров различной ширины вы будете использовать медиазапросы, начиная с нацеленных на узкий экран смартфона (в подразделе «Приоритет 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:

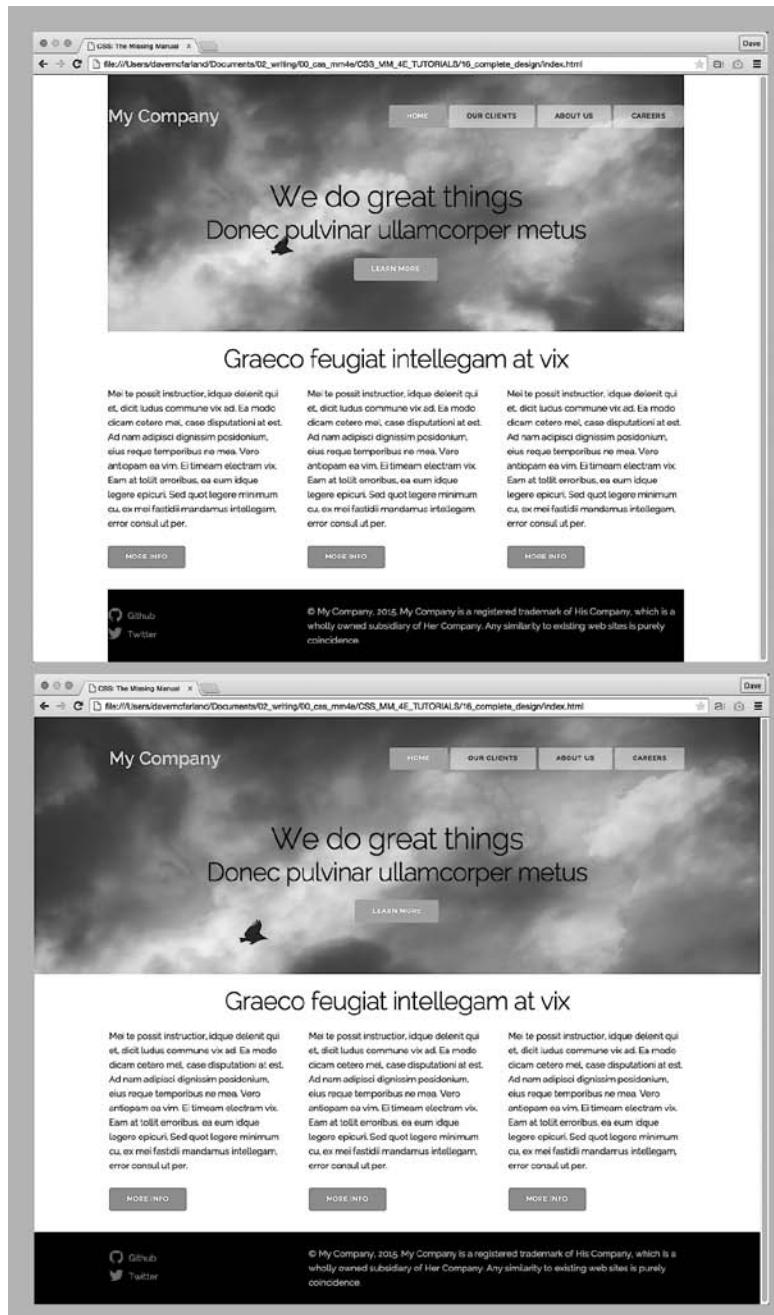
```
<div class="row">
  <div class="one-third column">
    ...
  </div>
  <div class="two-thirds column">
    ...
  </div>
</div>
```

---

## 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:

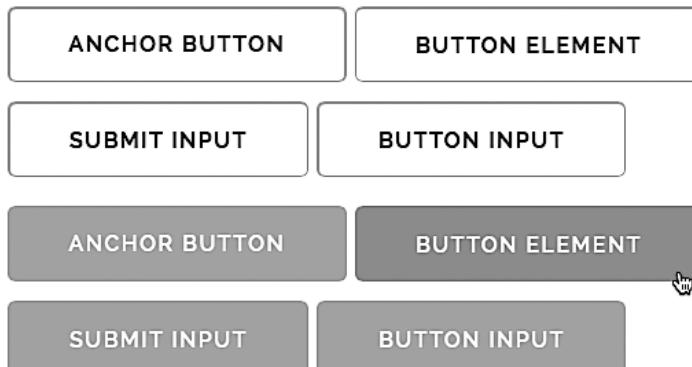
```
<a href="index.html" class="button">Главная</a>
```

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 крана 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 правилах `@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) пикселов в ширину. Если ширина окна или разрешение экрана, к примеру, 320 пикселов, стили внутри этого медиазапроса не будет 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 пикселов и выше */  
@media (min-width: 400px) {  
  
}  
/* стили для ширины крупнее фаблетов */  
/* стили для ширины 550 пикселов и выше */
```

```
@media (min-width: 550px) {  
}  
/* стили для ширины крупнее планшетов */  
/* стили для ширины 750 пикселов и выше */  
@media (min-width: 750px) {  
}  
/* стили для ширины крупнее компьютерных мониторов */  
/* стили для ширины 1000 пикселов и выше */  
@media (min-width: 1000px) {  
}  
/* стили для ширины крупнее компьютерных HD-мониторов */  
/* стили для ширины 1200 пикселов 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 пикселов).

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 пикселов 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 пикселов, 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](https://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>
</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="#">О 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="#">О 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="#">О 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="#">О 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`.

- 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.

- 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 раз рекламированная 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.

- 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.

- 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.

- 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`.

- 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 instructior...</p>
    <p><a href="#" class="button">Дополнительно</a></p>
  </div>
  <div class="four columns">
    <p>Mei te possit instructior...</p>
    <p><a href="#" class="button">Дополнительно</a></p>
  </div>
  <div class="four columns">
    <p>Mei te possit instructior...</p>
    <p><a href="#" class="button">Дополнительно</a></p>
  </div>
</div>
```

(Приведенный HTML-код сокращен и не включает «бред» на латинице в каждой из колонок.) Наконец, вы добавите нижний колонтитул на страницу.

16. 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, вставляется контейнер с одной строкой и двумя колонками. Ширина одной из них равна восьми столбцам, а второй — четырем столбцам.

17. 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.

1. 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.

- 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.

- 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.

4. 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`.

5. 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. Теперь вы можете добавить фотографию, которая заполнит окно браузера по всей ширине, но выйдет за пределы области заголовка.

6. 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:

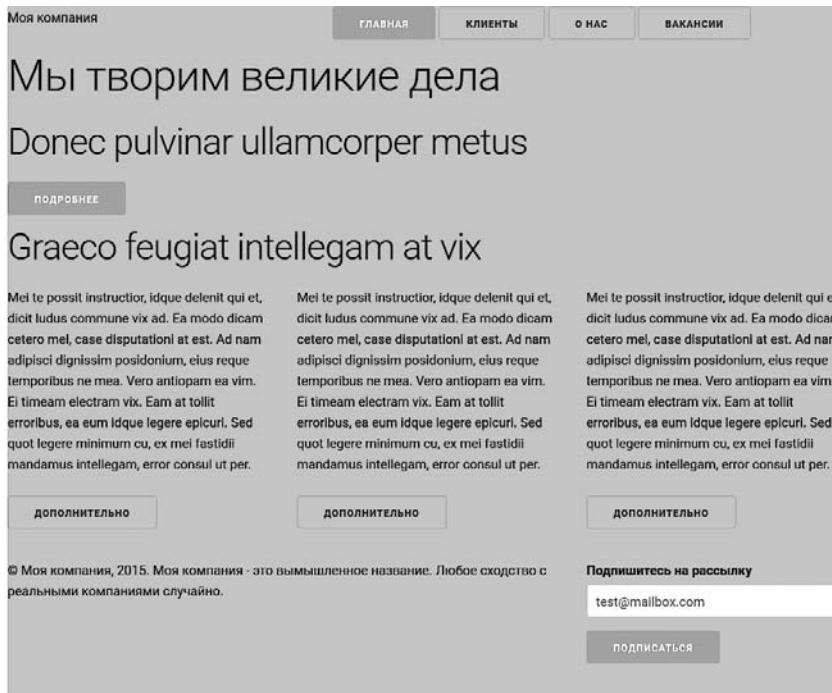
```
/* запросы mobile first */
/* стили для любой ширины */
.header {
    padding: 50px 0 70px 0;
    background-image: url(..../imgs/header.jpg);
    background-size: cover;
}
```

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.

7. 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.

#### 8. 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.

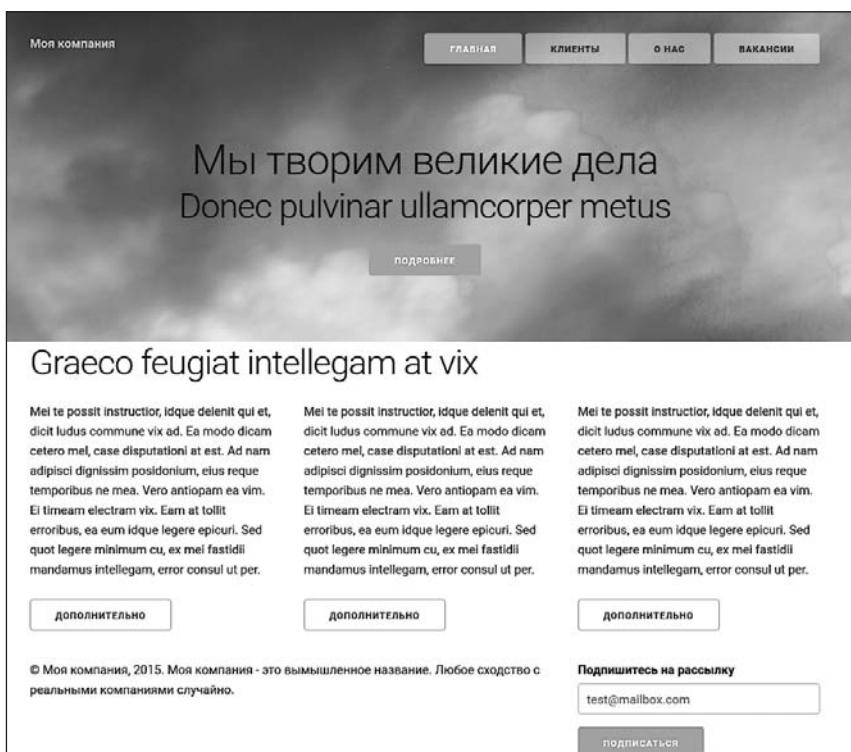
- Добавьте следующий код под только что добавленными стилями кнопок:

```
.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.

- 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 будут применяться, независимо от ширины окна браузера, как на малых экранах смартфонов, так и на очень широких компьютерных мониторах.

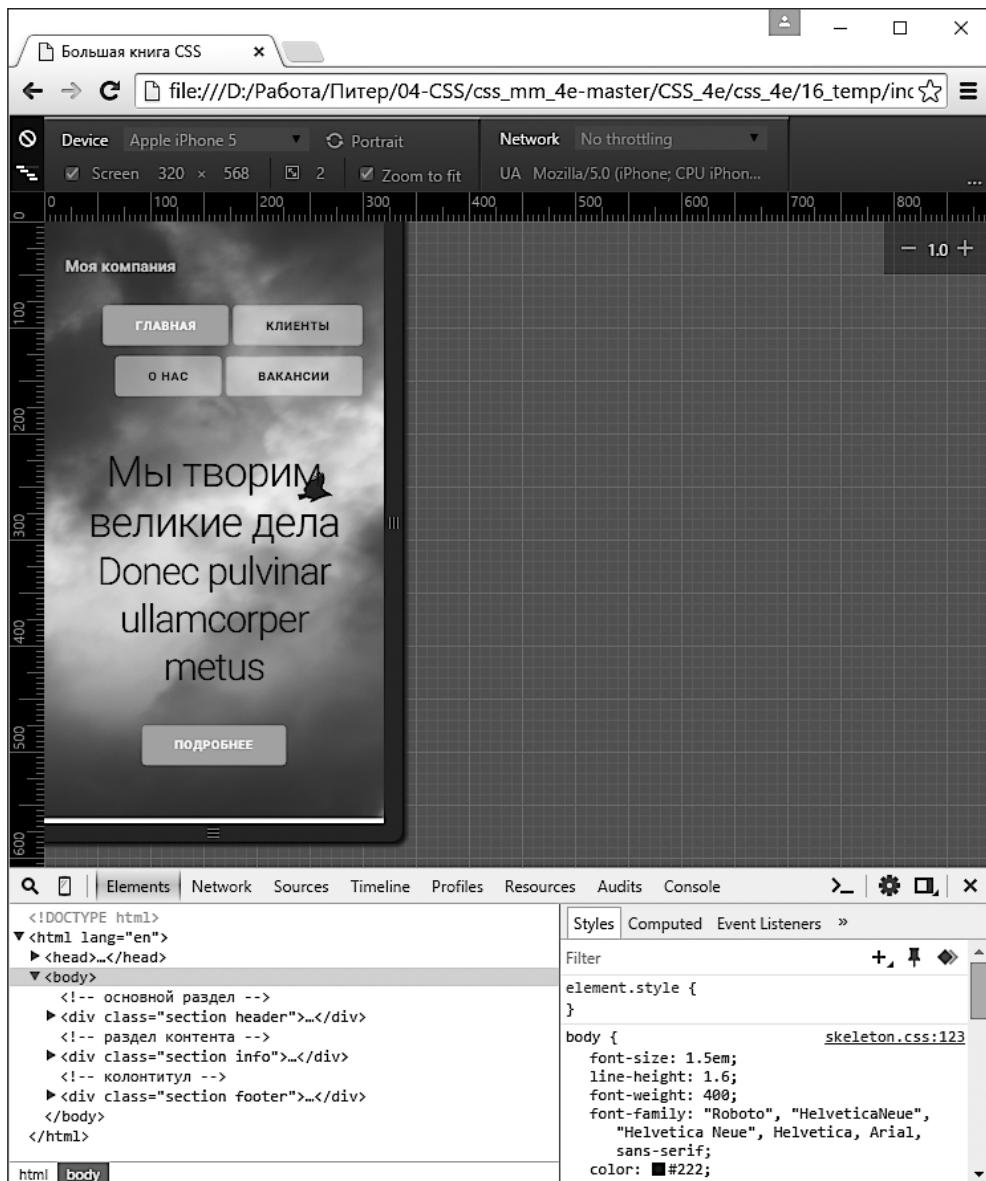
Во-первых, вы измените стиль `.logo` так, чтобы дизайн лучше смотрелся на небольшом экране смартфона.

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 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 пикселов. Другими словами, вам не нужно добавлять какие-либо стили к первому медиазапросу, значение свойства min-width которого равно 400 пикселам. Тем не менее при экранном разрешении по горизонтали 550 пикселов 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 пикселов 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 пикселов 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` пикселов. Затем медленно увеличьте ее.

Дизайн страницы, когда окно браузера уже `550` пикселов, показан вверху слева на рис. 16.13. По мере увеличения ширины окна браузера страница будет изменяться два раза: один раз при ширине `550` пикселов (*вверху справа*) и еще раз при ширине `750` пикселов (*внизу*).



**Рис. 16.12.** Модульная сетка системы Skeleton меняется при отображении на устройствах с экранным разрешением по горизонтали выше 550 пикселов.

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](http://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.

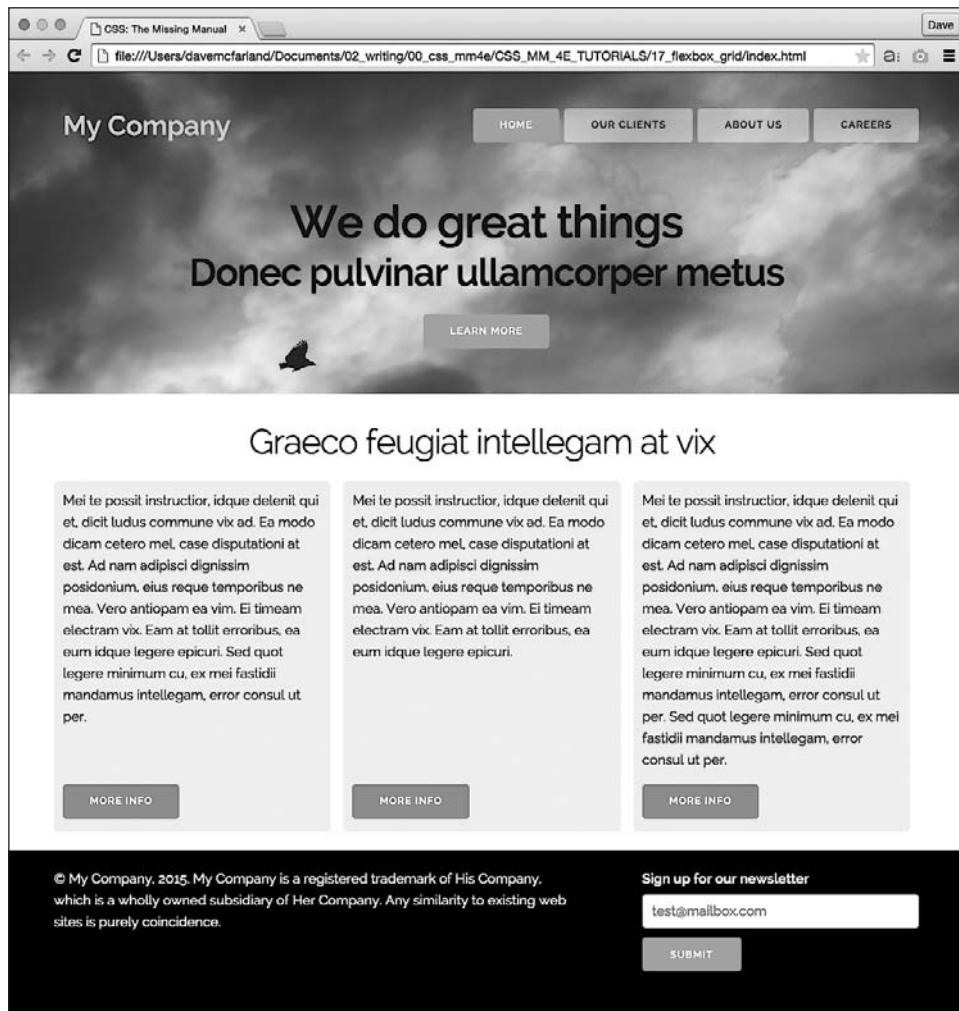
Тем не менее Всемирная паутина развивается, и рабочая группа CSS Консорциума W3C работает над тем, чтобы предоставить дизайнерам гибкие и мощные свойства каскадных таблиц стилей для верстки макетов страниц. Одним из новых методов, который уже довольно широко поддерживается браузерами, является спецификация *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 выше<sup>1</sup>, 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

<sup>1</sup> Частичная поддержка. — Примеч. пер.

## Основы flexbox-верстки

Суть flexbox-верстки довольно проста. Для работы с ней необходимы лишь два компонента.

- 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.
- 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, наверно, 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](http://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 пикселов от края контейнера). Результат действия кода показан на нижнем изображении на рис. 17.2.

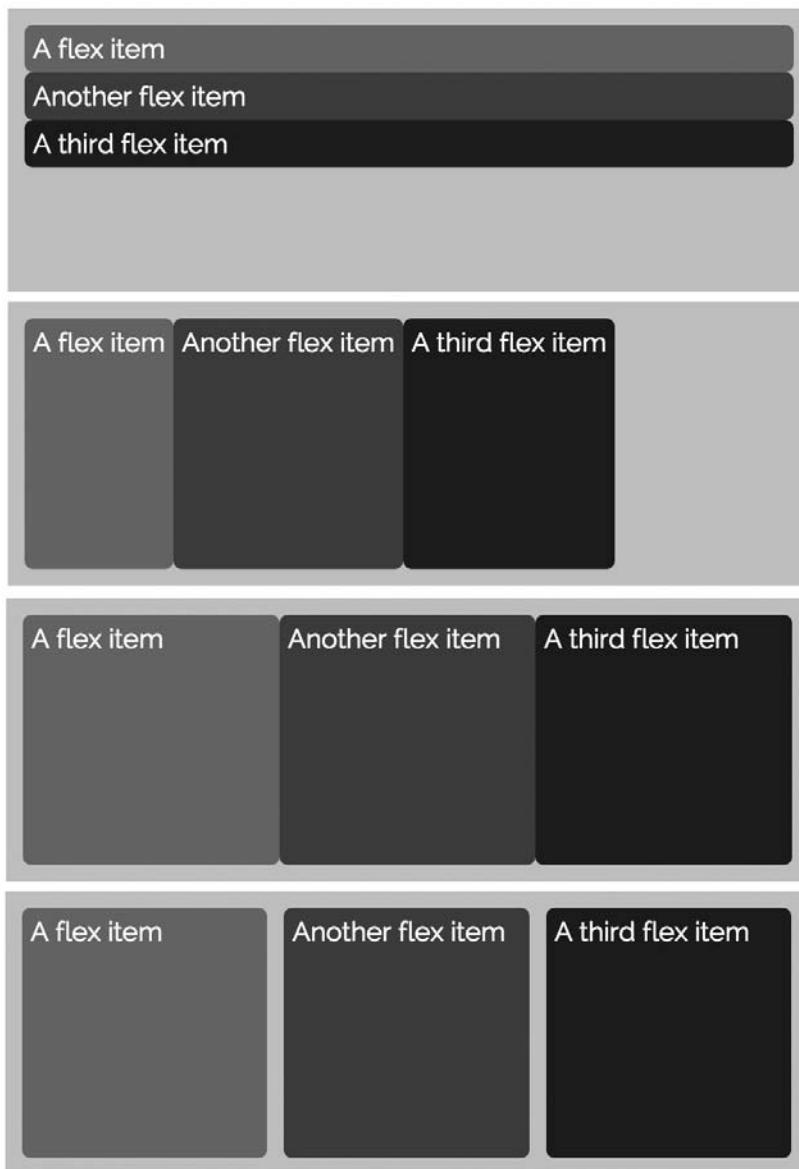
В теории метод `flexbox`-верстки довольно прост. Как вы можете видеть, для достижения эффекта не требуется длинного CSS-кода. Самое главное заключается в том, что вам не нужно беспокоиться об объектах, выпадающих из своих контейнеров, как это было с обтекаемыми элементами (см. раздел «Решение проблем 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 определяет, 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.

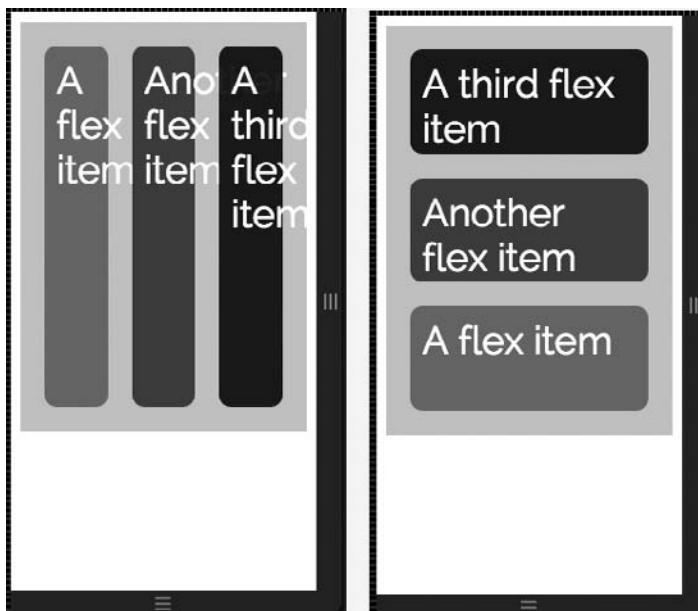


**Рис. 17.2.** Метод flexbox-верстки позволяет легко создавать колонки одинаковой ширины и высоты, расположенные рядом друг с другом без использования обтекаемых элементов. Дополнительное пространство, которое вы видите вокруг элементов (ярко-серый фон), — отступ, добавленный в flex-контейнер

Помните, что вы применяете это свойство к контейнеру — элементу, который обворачивается 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.

---

**Flex item #1**

```
.flex1 {
  flex: 1 1 300px;
}
```

Li European lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc, litot Europa usa li sam vocabular. Li lingues differe solmen in li grammatica, li pronunciation e li plu commun vocabules.

**Flex item #2**

```
.flex2 {
  flex: 1 1 300px;
}
```

Por scientie, musica, sport etc, litot Europa usa li sam vocabular.

**Flex item #3**

```
.flex3 {
  flex: 2 2 600px;
}
```

At solmen va esser necessari far uniform grammatica, pronunciation e plu sommun paroles. Ma quando lingues coalesce, li grammatica del resultant lingue es plu simplic e regulari quam ti del coalescent lingues. Li nov lingua franca va esser plu simplic e regulari quam li existent European lingues. It va esser tam simplic quam Occidental in fact, it va esser Occidental. A un Angleso it va semblar un simplificat Angles, quam un skeptic Cambridge amico dit me que Occidental es. Li European lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc, litot Europa usa li sam vocabular.

**Flex item #3**

```
.flex3 {
  flex: 2 2 600px;
}
```

At solmen va esser necessari far uniform grammatica, pronunciation e plu sommun paroles. Ma quando lingues coalesce, li grammatica del resultant lingue es plu simplic e regulari quam ti del coalescent lingues. Li nov lingua franca va esser plu simplic e regulari quam li existent European lingues. It va esser tam simplic quam Occidental in fact, it va esser Occidental. A un Angleso it va semblar un simplificat Angles, quam un skeptic Cambridge amico dit me que Occidental es. Li European lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc, litot Europa usa li sam vocabular.

**Flex item #1**

```
.flex1 {
  flex: 1 1 300px;
}
```

Li European lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc, litot Europa usa li sam vocabular. Li lingues differe solmen in li grammatica, li pronunciation e li plu commun vocabules.

**Flex item #2**

```
.flex2 {
  flex: 1 1 300px;
}
```

Por scientie, musica, sport etc, litot Europa usa li sam vocabular.

**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 выключения (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). Разумеется, 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). Разумеется, 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).

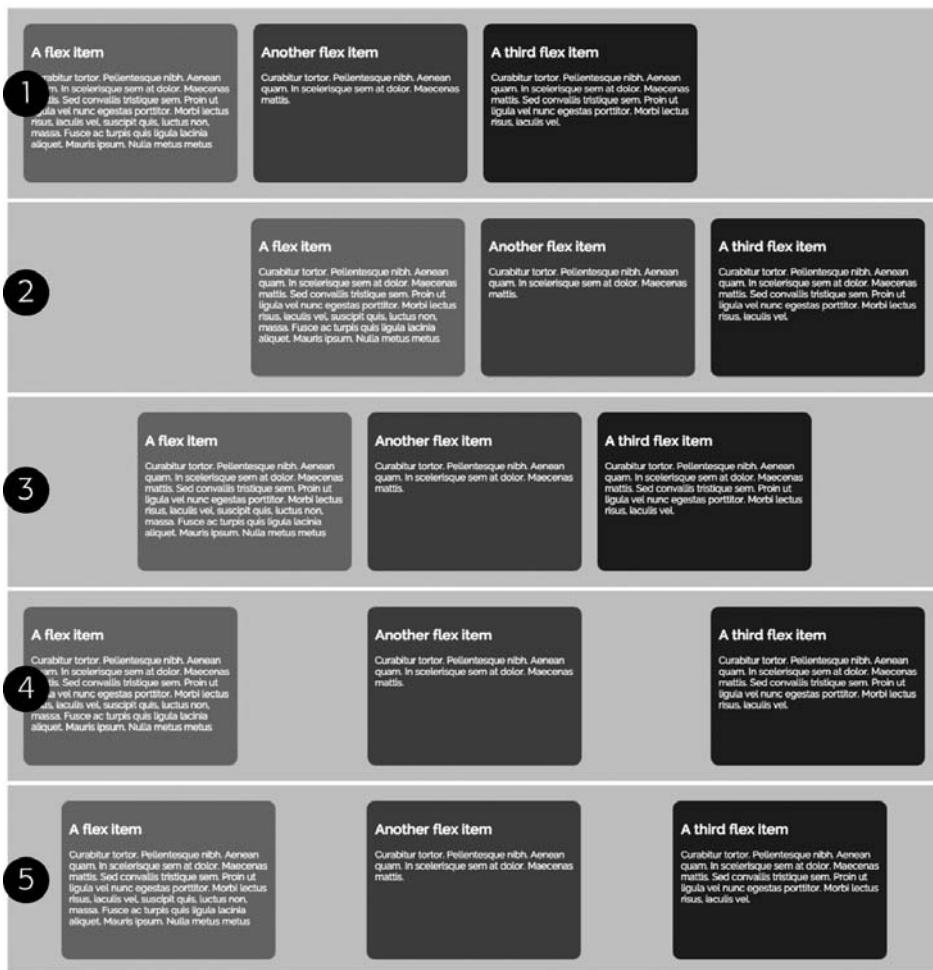
Чтобы использовать это свойство, необходимо добавить его в стиль, форматирующий flex-контейнер. Кроме того, необходимо убедиться, что для всех элементов указана ширина. Например, чтобы выровнять элементы, как показано на изображении 5 на рис. 17.5, можно использовать следующий CSS-код:

```
.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 вертикально-му 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. Растигивает 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 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, форматирующий 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, форматирующий 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.

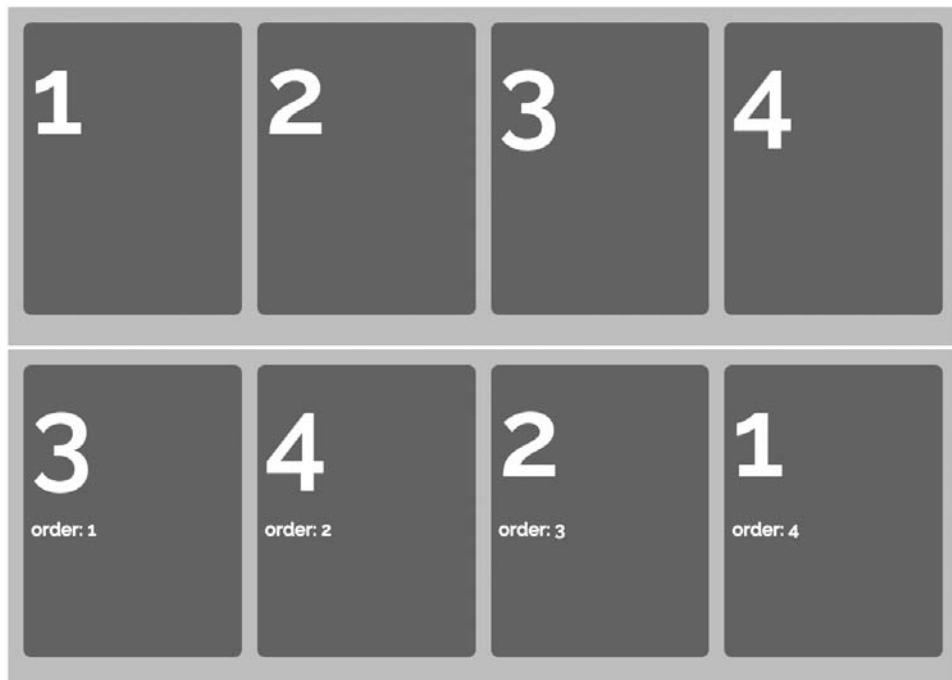
Несмотря на то что дизайнеры разработали способы, позволяющие разбивать на колонки поток контента, которым по умолчанию была представлена любая веб-страница, они всегда зависели от HTML-кода. Например, если требовалось создать страницу, на которой в одной колонке отображается основной контент, а две 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 внушильное 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 --&gt;
  &lt;/div&gt;
  &lt;div class="sidebar1"&gt;
    <!-- навигация и дополнения --&gt;
  &lt;/div&gt;
  &lt;div class="sidebar2"&gt;
    <!-- еще дополнения --&gt;
  &lt;/div&gt;
&lt;/div&gt;</pre>
```

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

---

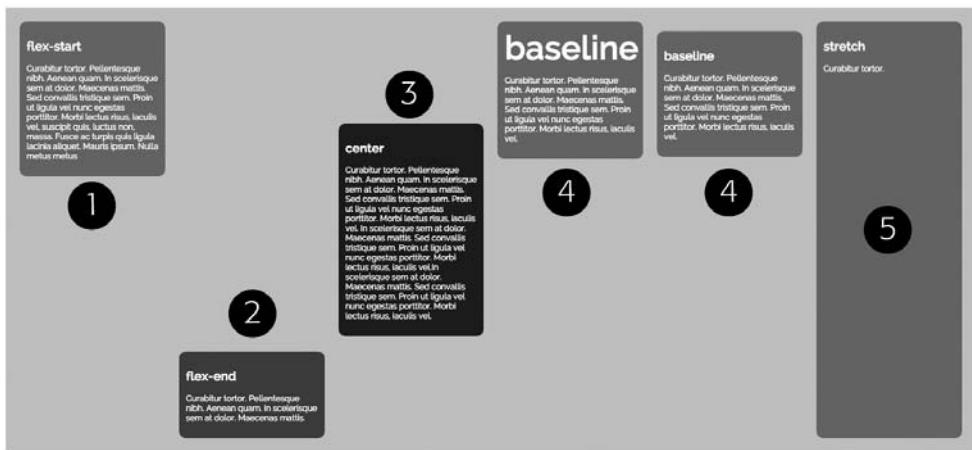
Если в качестве направления потока контента flex-контейнера использовать значение `column`, то свойство `order` будет определять порядок следования элементов сверху вниз: flex-элементы с меньшим номером отображаются выше flex-элементов с большим номером. Кроме того, значения `column-reverse` и `row-reverse` 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 склоняются. 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](https://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. Во-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.

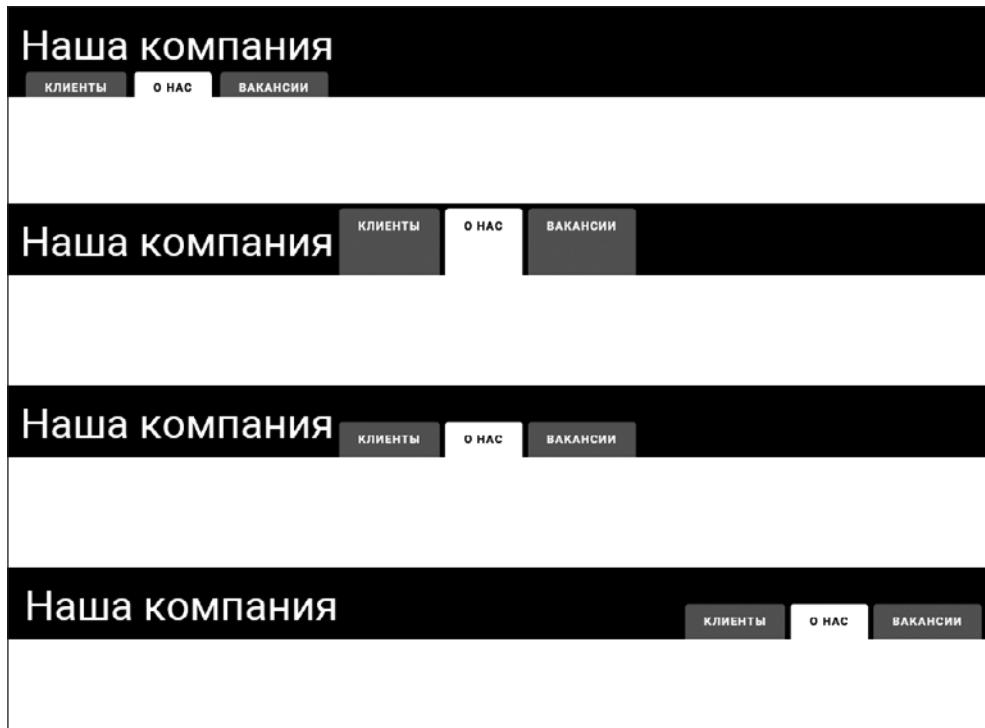
3. Измените стиль `.banner`, добавив свойство `align-items`:

```
.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 действиях.

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:

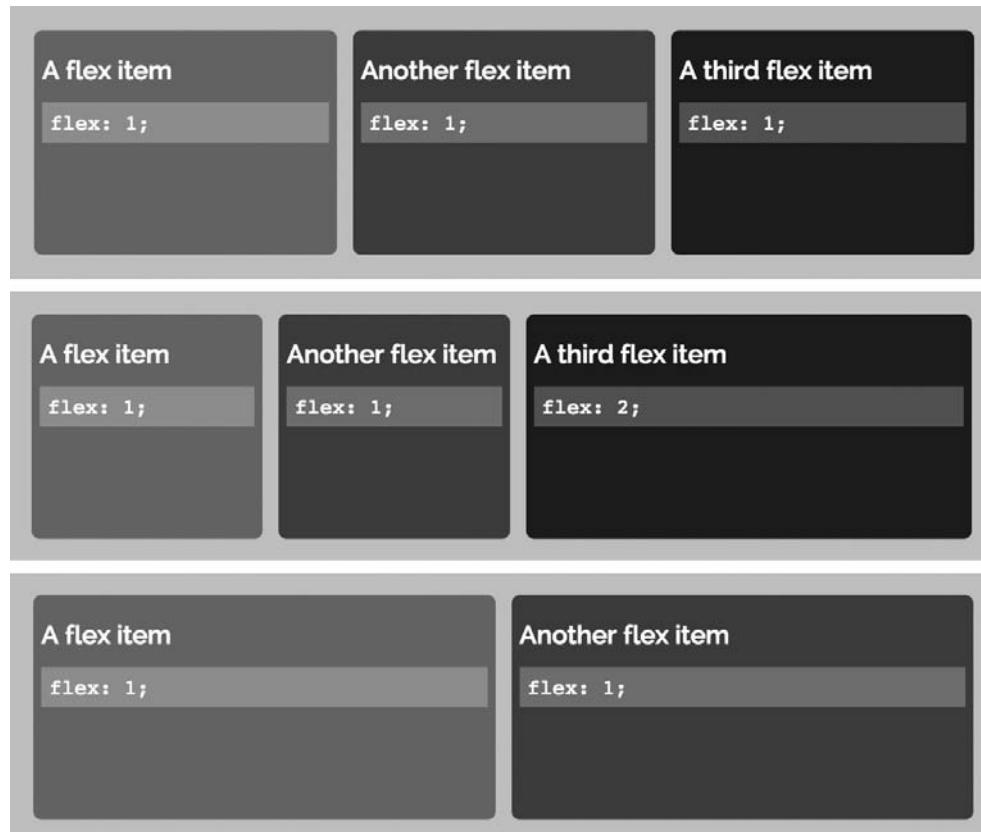
```
.container {  
    display: flex;  
}  
.container div {  
    flex: 1;  
}
```

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

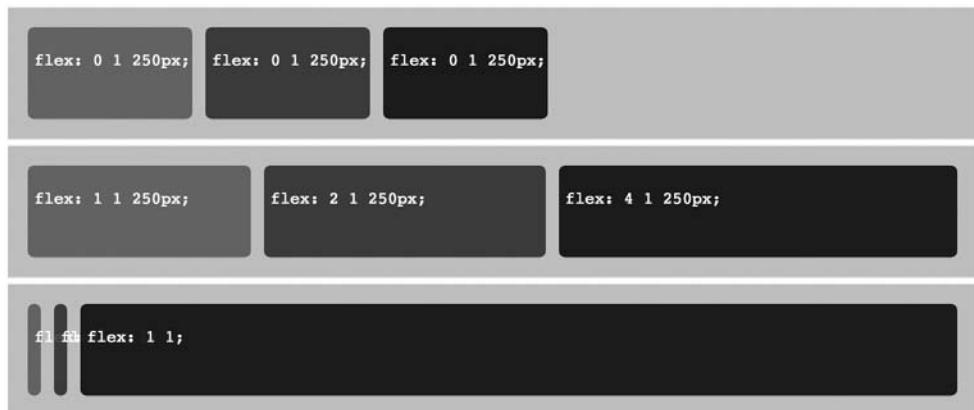
На сайте [tinyurl.com/qznwflh](http://tinyurl.com/qznwflh) находится статья, подробно описывающая использование свойства flexbox.

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 пикселов. 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 пикселов.

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 пикселов, 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 пикселов). Значения свойств `flex-basis` каждого из элементов различны и составляют 300, 200 и 100 пикселов 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 пикселов. Таким образом, путем вычитания полученного нами результата из ширины контейнера можно определить дополнительное пространство:  $1000 - 600 = 400$ .

Другими словами, доступно 400 пикселов 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:  $300 + 50 = 350$  пикселов.

Ширина второго элемента состоит из суммы значения свойства `flex-basis` и доли дополнительного пространства, приходящегося на него:  $200 + (50 \times 3) = 350$  пикселов.

А к ширине последнего элемента необходимо прибавить  $1/2$  от 400 пикселов:  $100 + 200 = 300$  пикселов.

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`.

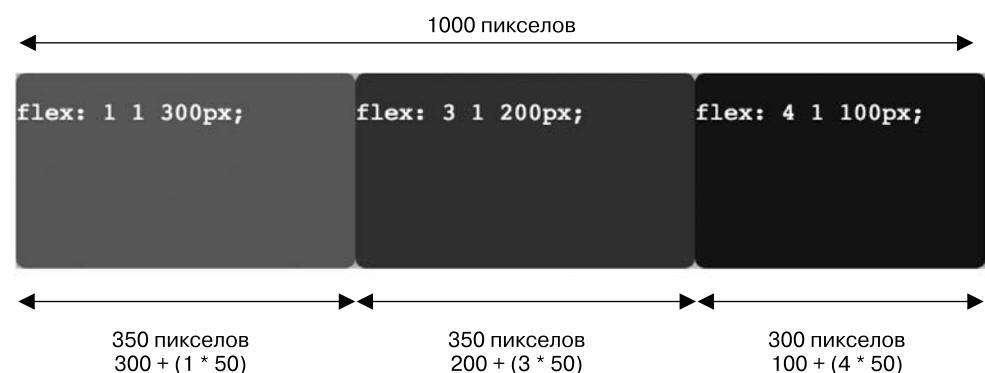


Рис. 17.13. Фактическая ширина flex-элемента зависит от ширины flex-контейнера и значений свойств `flex-grow` и `flex-basis`

## 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 `nowrap`, согласно которому все элементы должны находиться рядом друг с другом в одной строке.

Например, flex-элементы, изображенные на рис. 17.14, занимают 1000 пикселов в ширину. Поскольку значение свойства `flex-basis` каждого из них составляет 400 пикселов (общая ширина — 1200 пикселов), 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](http://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 пикселов, и три `flex`-элемента, свойству `flex-basis` каждого из которых присвоено значение `400px`. Поскольку  $400 + 400 + 400$  больше, чем 1000 пикселов, все три `flex`-элемента не помещаются в контейнере. Однако, поскольку  $400 + 400$  меньше, чем 1000 пикселов, два из них без проблем умещаются 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 пикселов, 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 присвоенное ему свойства `flex-basis` такое же, как у двух других элементов, и составляет 100 пикселов. 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 пикселов, можно присвоить свойству `flex-basis` каждого из них значение `200px`. Если окно браузера станет уже 600 пикселов, то первый и второй элементы будут располагаться рядом друг с другом, а третий будет перенесен на новую строку. Если окно браузера станет уже 400 пикселов, 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 неимоверных 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](https://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`.



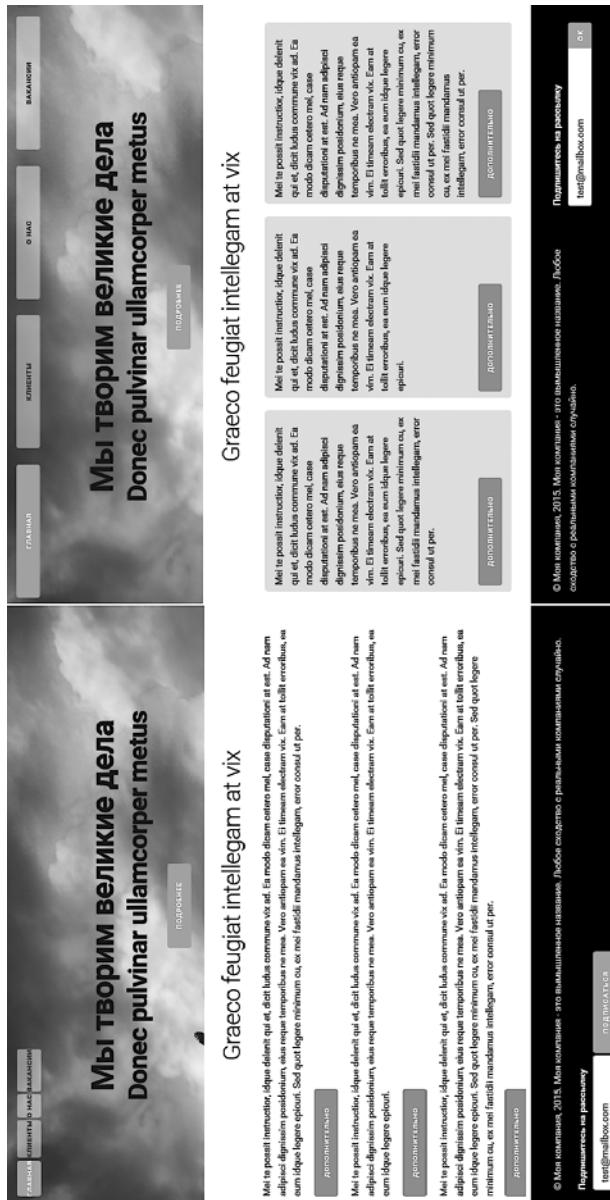
**Рис. 17.16.** Чтобы размеры flex-элементов сохранили свои пропорции, убедитесь, что значения свойства `flex-basis`, определяющего базовую ширину ваших flex-элементов, пропорционально соответствуют значениям свойства `flex-grow` этих элементов (*вверху*). Если это не так, вы не сможете точно установить 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 выдавливанию 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.

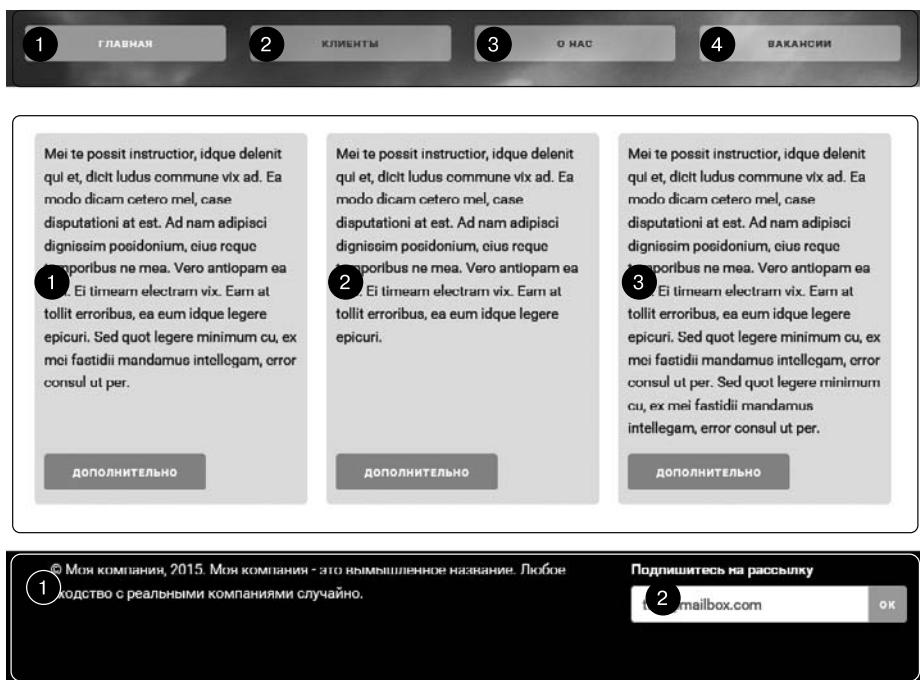
Далее вам придется выполнить некоторые довольно сложные настройки. Вы разместите кнопки навигации равномерно по контейнеру таким образом, чтобы крайняя левая кнопка находилась у левого края контейнера, правая кнопка — возле правого края контейнера, а две другие кнопки были равномерно распределены между ними.

- 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 немногого 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.

- 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`.

- 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 смешены 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.

Mei te possit instructior, idque delenit qui et, dicit ludus commune vix ad. Ea modo dicam cetero mel, case disputationi at est. Ad nam adipisci dignissim posidonium, eius reque temporibus ne mea. Vero antipopam ea vim. Ei timeam electram vix. Eam at tollit erroribus, ea eum idque legere epicuri. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per.

**ДОПОЛНИТЕЛЬНО**

Mei te possit instructior, idque delenit qui et, dicit ludus commune vix ad. Ea modo dicam cetero mel, case disputationi at est. Ad nam adipisci dignissim posidonium, eius reque temporibus ne mea. Vero antipopam ea vim. Ei timeam electram vix. Eam at tollit erroribus, ea eum idque legere epicuri.

**ДОПОЛНИТЕЛЬНО**

Mei te possit instructior, idque delenit qui et, dicit ludus commune vix ad. Ea modo dicam cetero mel, case disputationi at est. Ad nam adipisci dignissim posidonium, eius reque temporibus ne mea. Vero antipopam ea vim. Ei timeam electram vix. Eam at tollit erroribus, ea eum idque legere epicuri. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per.

**ДОПОЛНИТЕЛЬНО**

Mei te possit instructior, idque delenit qui et, dicit ludus commune vix ad. Ea modo dicam cetero mel, case disputationi at est. Ad nam adipisci dignissim posidonium, eius reque temporibus ne mea. Vero antipopam ea vim. Ei timeam electram vix. Eam at tollit erroribus, ea eum idque legere epicuri. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per.

**ДОПОЛНИТЕЛЬНО**

Mei te possit instructior, idque delenit qui et, dicit ludus commune vix ad. Ea modo dicam cetero mel, case disputationi at est. Ad nam adipisci dignissim posidonium, eius reque temporibus ne mea. Vero antipopam ea vim. Ei timeam electram vix. Eam at tollit erroribus, ea eum idque legere epicuri. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per.

**ДОПОЛНИТЕЛЬНО**

Mei te possit instructior, idque delenit qui et, dicit ludus commune vix ad. Ea modo dicam cetero mel, case disputationi at est. Ad nam adipisci dignissim posidonium, eius reque temporibus ne mea. Vero antipopam ea vim. Ei timeam electram vix. Eam at tollit erroribus, ea eum idque legere epicuri. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per. Sed quot legere minimum cu, ex mei fastidii mandamus intellegam, error consul ut per.

**ДОПОЛНИТЕЛЬНО**

**Рис. 17.19.** Flex-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)

- 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;
}
```

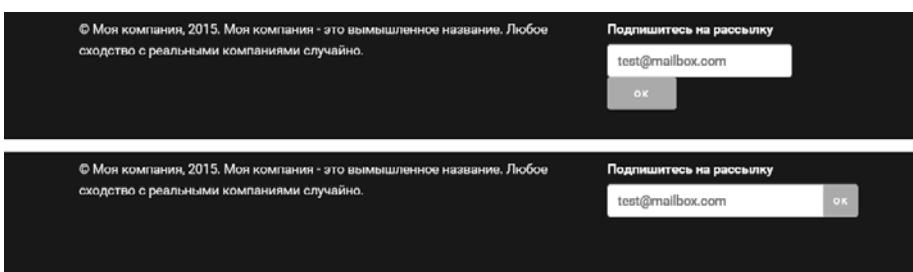
Эти стили увеличивают область с информацией об авторских правах в два раза по сравнению с веб-формой подписки на рассылку. Небольшое поле справа отодвигает веб-форму подписки на рассылку от уведомления об авторских правах (рис. 17.20, *вверху*).

Последнее, что осталось сделать, — это организовать элементы веб-формы. Было бы прекрасно, если бы кнопка отправки данных (OK) 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.

- 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 колонитула в flex-контейнер, а текстовую метку, поле ввода и кнопку отправки данных внутри веб-формы — в flex-элементы. Далее вы растяните метку на всю ширину веб-формы.



**Рис. 17.20.** С помощью flexbox легко выровнять элементы веб-формы, такие как поле ввода и кнопка отправки данных (*внизу*)

- 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 `OK`).

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 пикселов. Другими словами, стили, размещенные в нем, будут применяться только к устройствам (и окнам браузеров), экранное разрешение по горизонтали которых — менее 501 пикселя.

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):

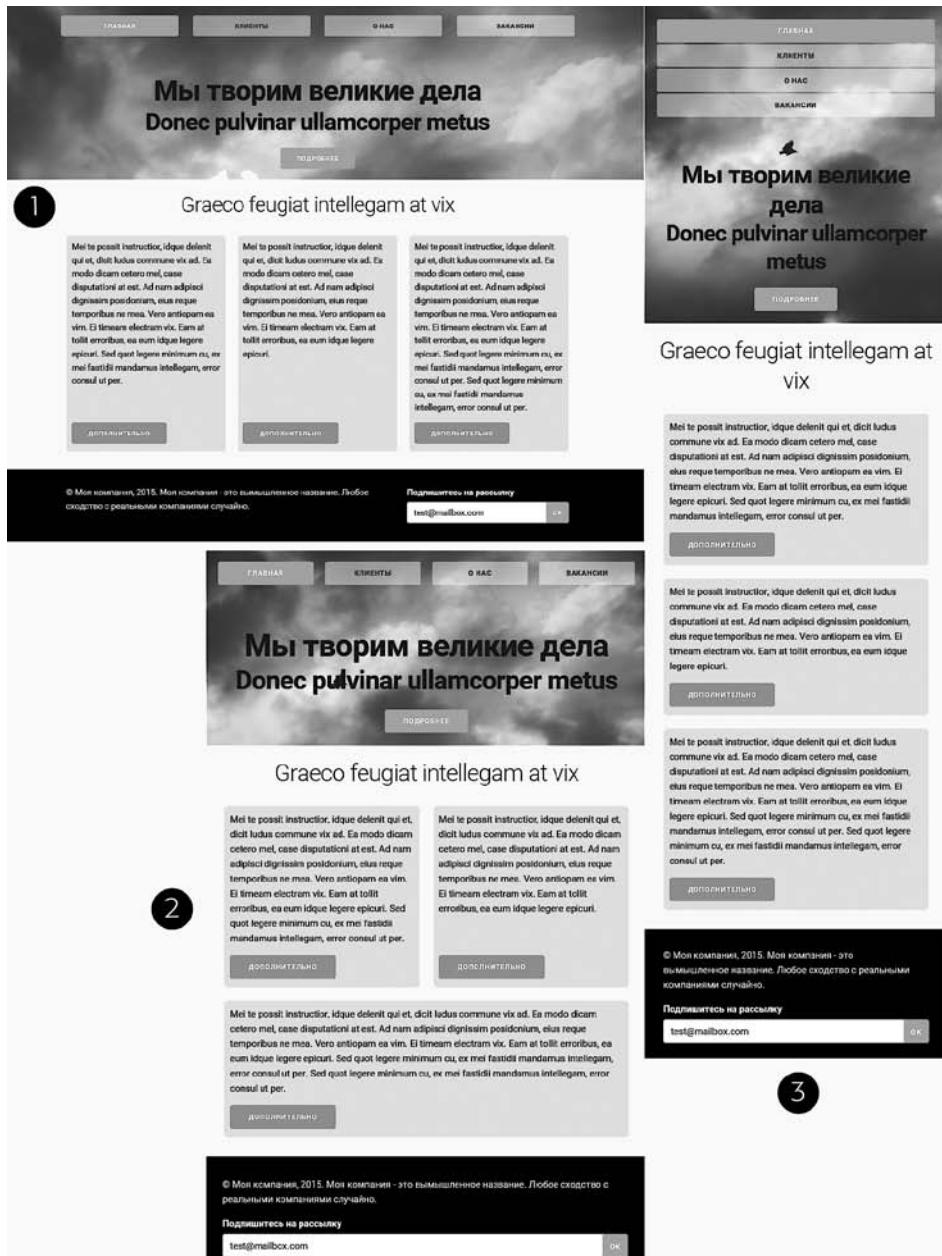
```
@media (max-width: 500px) {  
    .nav {  
        flex-flow: column;  
    }  
    .nav a {  
        width: 100%;  
        margin-bottom: 2px;  
    }  
}
```

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 немногого уже, три колонки превращаются в две, а элементы нижнего колонтитула перетекают и располагаются друг под другом (2). Наконец, при ширине окна браузера менее 500 пикселов 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 нечитываются 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 выясняете, 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](http://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 пикселов, а завтра, может

быть, вы присвойте ему размер 24 пикселя 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 формирование 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 пикселя 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 (смещать 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

```

```

The screenshot shows a website header for 'COSMOFARMER 2.0' with a sub-header 'Your online guide to apartment farming'. Below the header is a navigation bar with links: Home, Features, Experts, Quiz, Projects, and Horoscopes. The main content area has a title 'Bathtub Hydroponics'. On the left, there is a small image of a bunch of carrots and some descriptive text. On the right, there is a large image of a tomato. Both images have a thin black border around them. The text next to the images is in a placeholder font.

**Image Class Example:**

```

    <img alt="A bunch of carrots" class="imgFrame" />
    <img alt="A tomato" class="imgFrame" />
  
```

The text in the main content area is placeholder text (Lorem ipsum).

**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 — группировка 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.

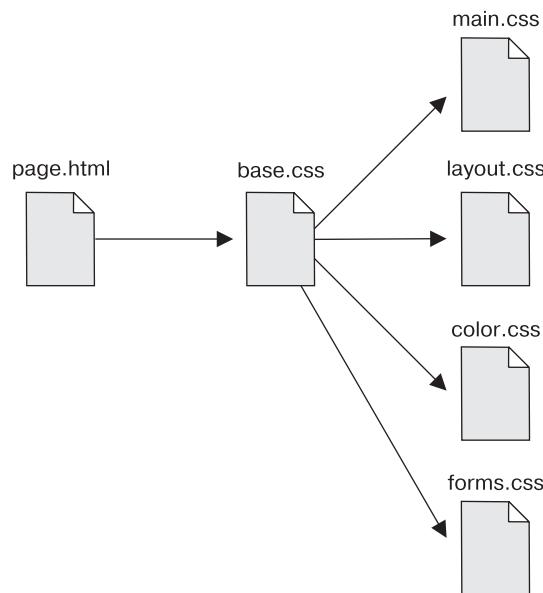
Например, создайте файл `color.css` со стилями для изменения цвета сайта, файл `forms.css`, который управляет внешним видом веб-форм, файл `layout.css` — для компоновки макета и файл `main.css`, который охватывает все остальное форматирование (см. рис. 18.3, справа).

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

Пропроцессоры 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.

---

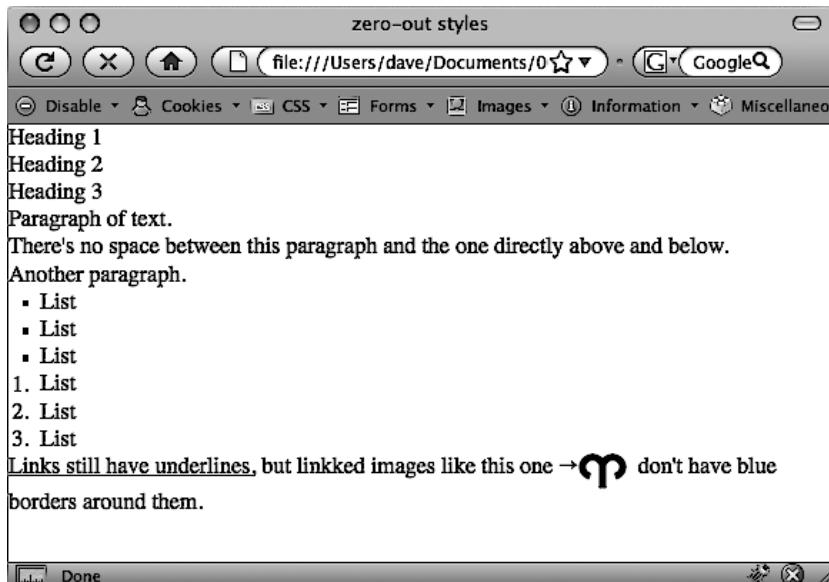
## Устранение конфликтов стилей в браузере

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 1 `em`, браузеры используют различные размеры для других элементов. Вы можете сделать так, чтобы для всех элементов был установлен размер шрифта 1 `em`, 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 однообразную высоту строк.** Браузеры по умолчанию могут с незначительным различием отображать высоту строк текста. Применив 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 ([tinyurl.com/cop5s89](http://tinyurl.com/cop5s89)) 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 — *мягкая классификация*. 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="text/css" href="main.css?v=2">
```

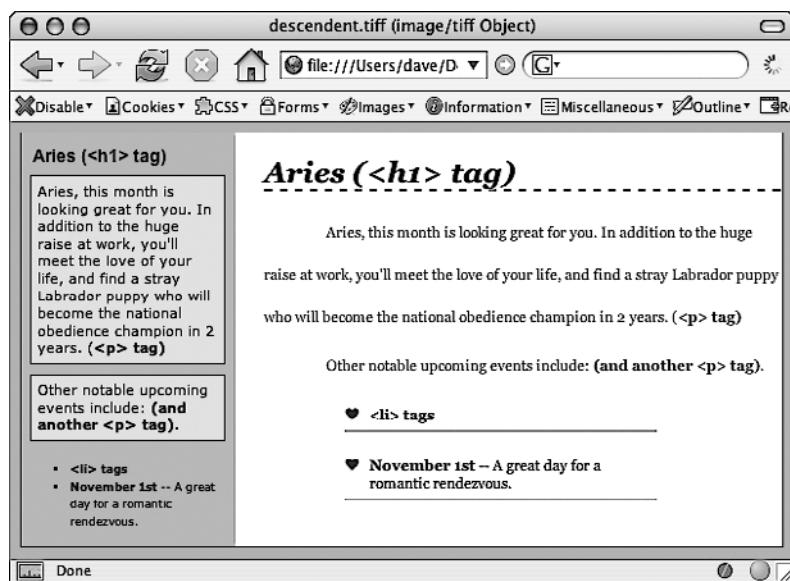
Браузер определит, что таблица стилей отлична от сохраненной в кэше версии файла `main.css` и загрузит файл с веб-сервера.

На самом деле код `?v=1` ничего не делает и не влияет на то, как работает веб-сервер. Это лишь способ сообщить о том, что страница должна быть обработана в определенном режиме.

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 предлагаю 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) форматируются 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">`.

2. Добавьте навигационную панель на страницу.

Пошаговые инструкции по созданию навигационной панели вы найдете в разделе «Использование ролловеров» главы 9.

3. Обозначьте каждую ссылку навигационной панели.

Для ссылки на главную страницу у вас может использоваться следующий код: `<a href="/index.html" class="homeLink">Главная</a>`. Атрибут `class` позволяет обозначить ссылку как указывающую на главную страницу. Повторите это для других ссылок: `<a href="/news/" class="newsLink">Новости</a>` 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.

4. 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: #FBF99;  
}
```

#### 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 «теги 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/qzo9jk](http://tinyurl.com/qzo9jk). Проект 00CSS можно найти по адресу [tinyurl.com/kfz84cn](http://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](http://smacss.com). По адресу [tinyurl.com/pl38ake](http://tinyurl.com/pl38ake) можно найти множество видеоуроков, 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. *При-меси* 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.

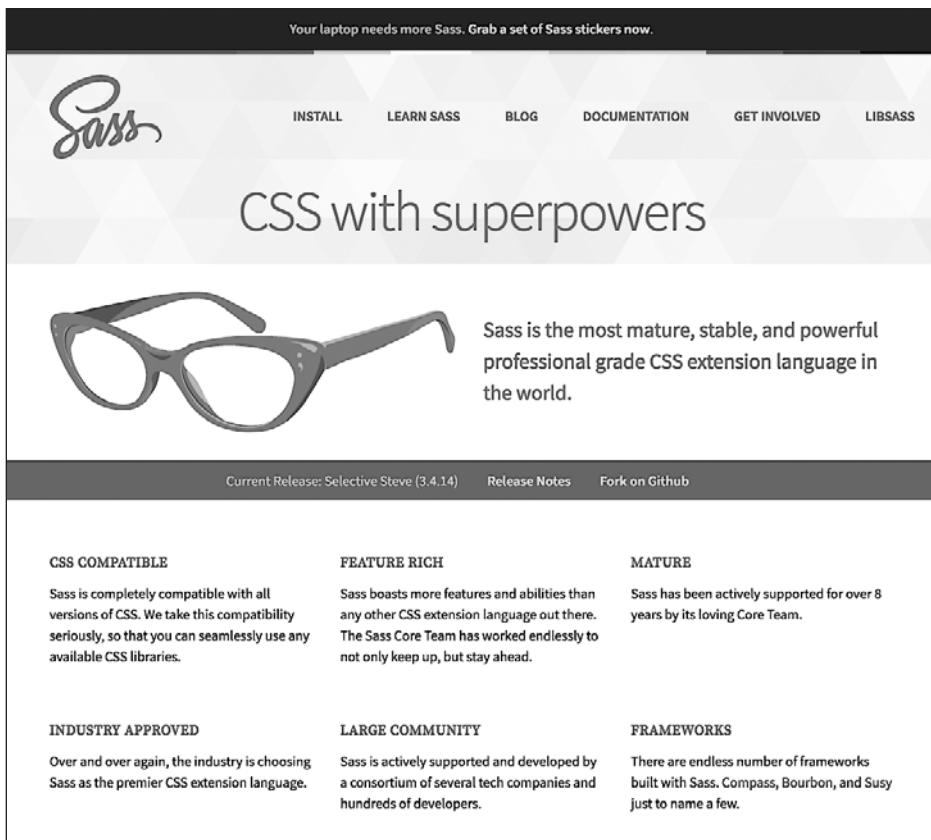
#### ПРИМЕЧАНИЕ

---

Подробную информацию об инструменте Sass можно найти на официальном сайте [sass-lang.com](http://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 инсталлировать 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](http://gruntjs.com)) и Gulp ([gulpjs.com](http://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/](http://rubyinstaller.org/downloads/).

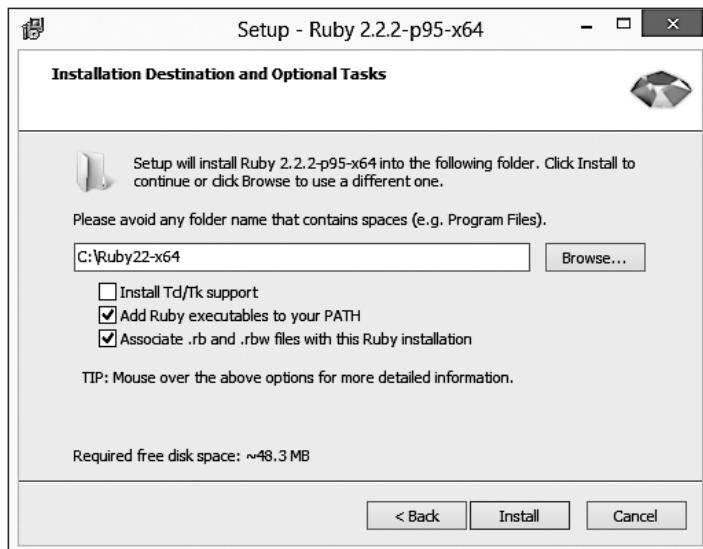
Скачайте последнюю версию среды разработки Ruby, перейдя по одной из ссылок загрузки. Если на вашем компьютере установлена 64-разрядная версия операционной системы Windows, выберите 64-разрядную версию. Установщик представляет собой обычный исполняемый файл Windows с расширением .exe.

### 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 флагшки **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 флагок. 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 флагок, 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](http://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.

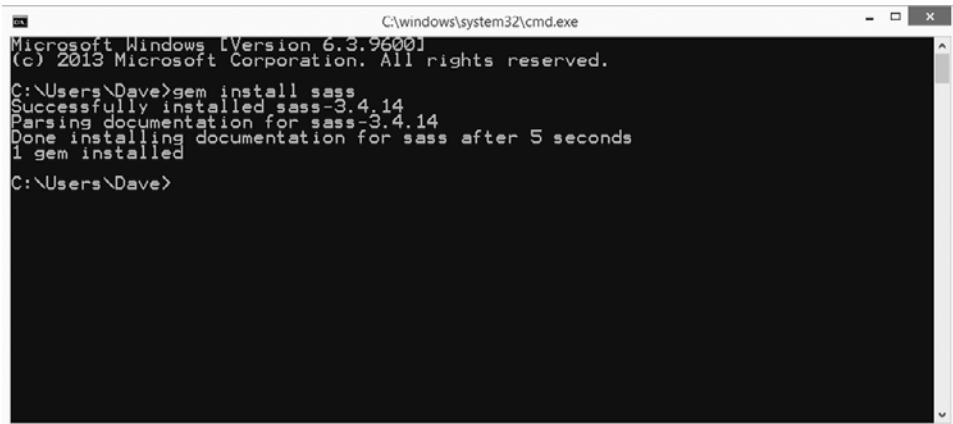
Кроме того, если вы не установили флагок **Add Ruby executables to your PATH** (Добавить в системные пути исполняемые файлы Ruby) при установке среды разработки Ruby (см. рис. 19.2), то получите сообщение об ошибке вида «**gem**' is not recognized as an internal or external command, operable program 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.



The screenshot shows a Windows command prompt window titled 'C:\windows\system32\cmd.exe'. The window displays the following text:  
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`:

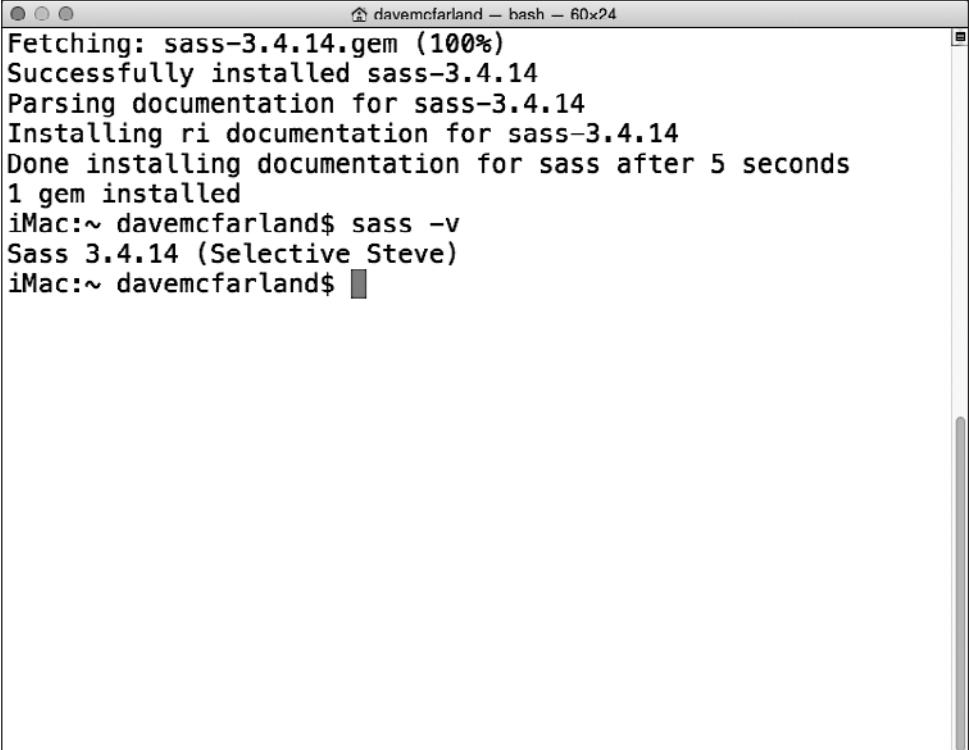
```
sudo gem install sass
```

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](http://tinyurl.com/ksvxl79)) — популярное бесплатное приложение, предназначенное как для операционной системы Windows, так и для OS X.
- Приложение Koala ([koala-app.com](http://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](http://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/3flf6bo](http://tinyurl.com/3flf6bo)) — 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 генерирует 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/bgssifk](http://tinyurl.com/bgssifk).

- 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](http://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

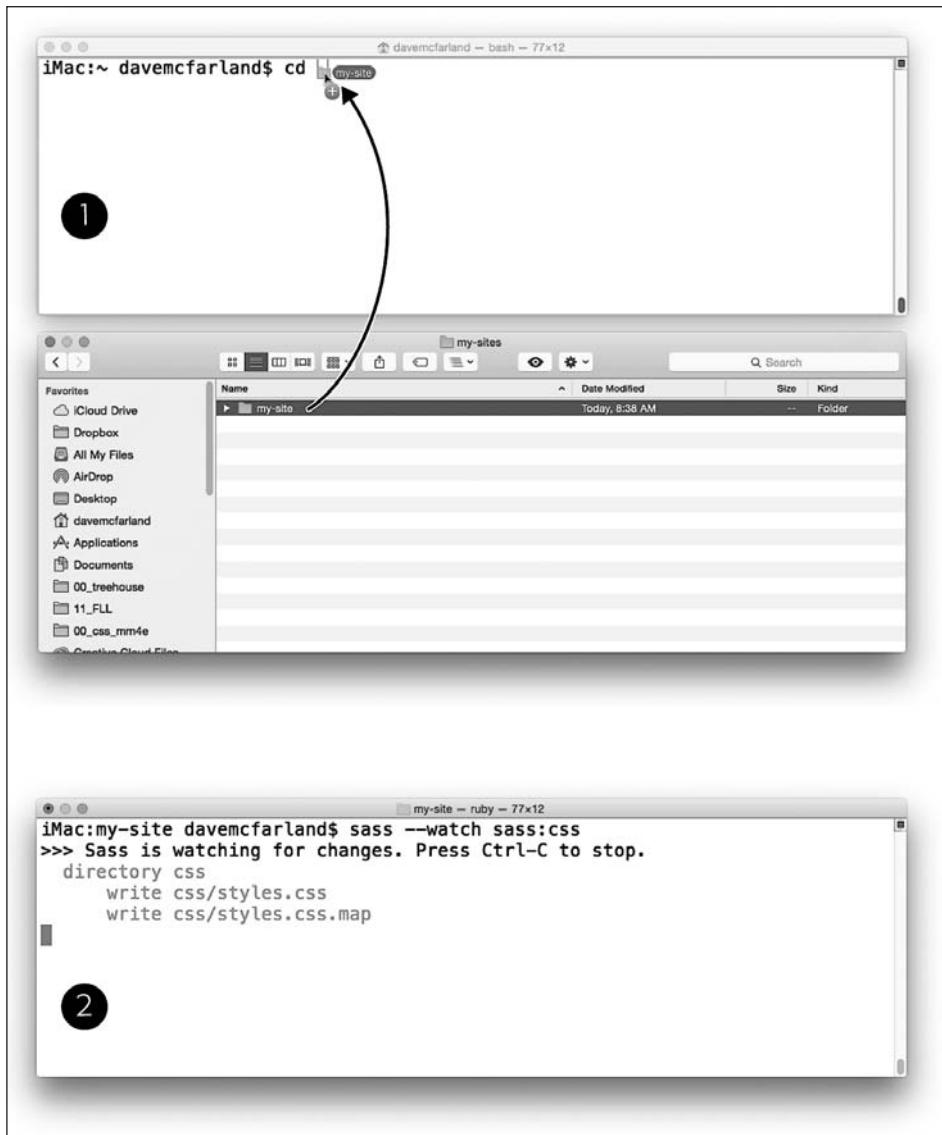
Препроцессор Sass также может оптимизировать финальный CSS-файл, удалив комментарии и лишние пробелы, а также внести другие изменения, которые позволяют уменьшить размер CSS-файла. В результате файл таблицы стилей у посетителей вашего сайта будет загружаться максимально быстро. Чтобы сжать CSS-файлы 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 правилаброса, 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 сокращение: 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 [tinyurl.com/ctemo55](http://tinyurl.com/ctemo55) и [tinyurl.com/zc5c3or](http://tinyurl.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: #FFEDD5;
$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 пикселов 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/qdqyo65](http://tinyurl.com/qdqyo65), [tinyurl.com/npxfy8t](http://tinyurl.com/npxfy8t) и русскоязычный [ресурс tinyurl.com/perlp4h](http://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 а 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: #FFEE00;
  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.

Метазык 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: #FFEDD5;  
    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: #FFEDD5;  
  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: #FFEDD5;  
        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 пиксель, 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](http://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** иметь 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 многоократно. 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-свойства 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 [tinyurl.com/perlp4h](http://tinyurl.com/perlp4h).

## 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 *передадите* 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](http://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](http://neat.bourbon.io)) — библиотека примесей, которая упрощает создание CSS-сеток, разработанная создателями библиотеки Bourbon.
- **Susy** ([susy.oddbird.net](http://susy.oddbird.net)) — еще один набор примесей для создания CSS-сеток. Эта библиотека очень популярна среди пользователей Sass.
- **Breakpoint** ([breakpoint-sass.com](http://breakpoint-sass.com)) — библиотека примесей, созданная с одной только целью — упростить и ускорить написание медиазапросов.
- **Compass** ([compass-style.org](http://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 примести 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 2 em, но увеличивался до 3 em, когда экранное разрешение по горизонтали превышает 1200 пикселов. 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](http://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 пикселов — 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 стилями, 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 пикселов. 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 следующий 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 SASS»). В этом случае он вычитает 1 пиксель 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 пиксель 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 пикселов. 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

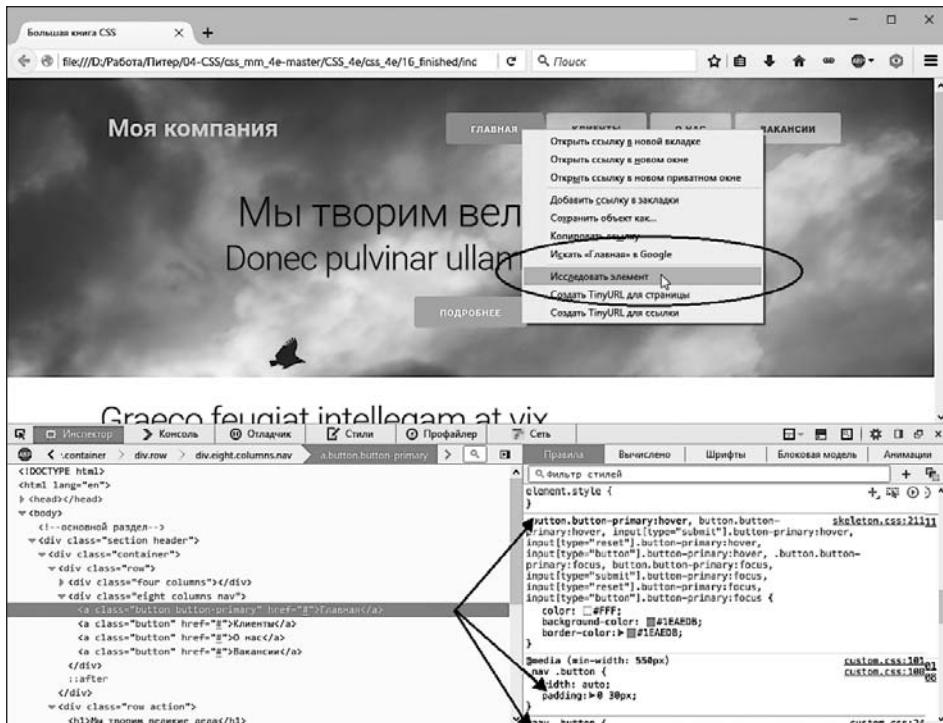
---

Существует много других способов, с помощью которых использование медиазапросов в Sass можно упростить. О них вы можете прочитать в следующих статьях: [tinyurl.com/ztcn289](http://tinyurl.com/ztcn289), [tinyurl.com/pqwj7tr](http://tinyurl.com/pqwj7tr), [tinyurl.com/7ntbgl3](http://tinyurl.com/7ntbgl3) и [tinyurl.com/18r](http://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 обращаться 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**

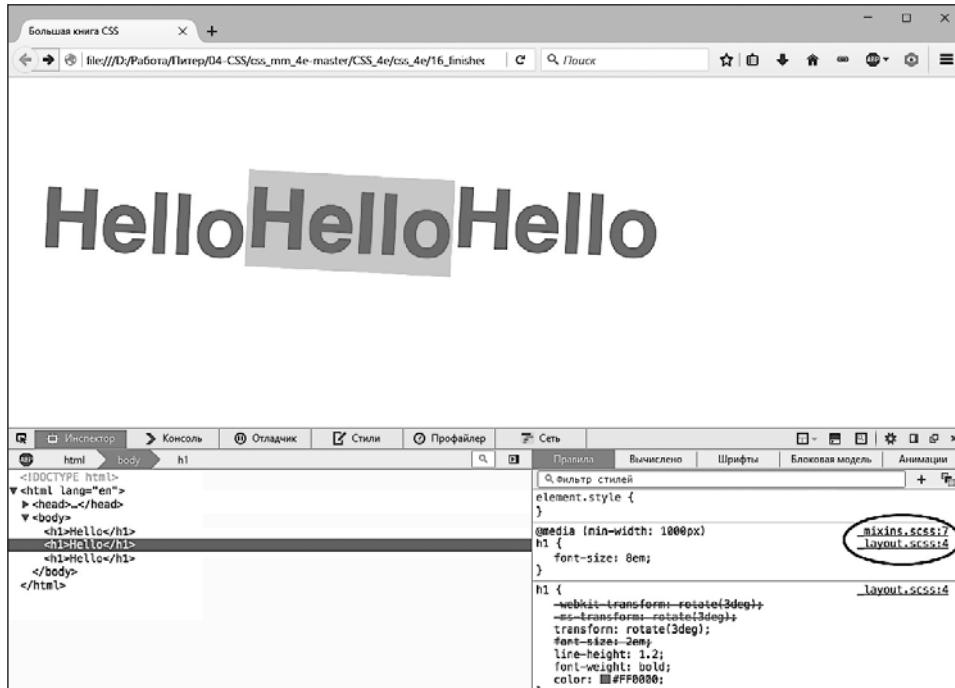
В старых версиях Sass приходилось включать функцию формирования карт кода вручную с помощью следующего кода:

```
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 веб-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 форматируют 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](http://tinyurl.com/n8xhw). Полный список свойств каскадных таблиц стилей опубликован на сайте [tinyurl.com/ke6h5dt](http://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 поддерживают

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](http://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 (полнная 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](http://tinyurl.com/zn2d) или более совершенное средство управления палитрами цветов на сайте [kuler.adobe.com](http://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, цицero, 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: пикселях, em и процентах.

## Пиксели

Пиксель — это одна точка на экране компьютера. Пиксели предоставляют последовательный метод обозначения размеров от компьютера к компьютеру: 72 пикселя на одном мониторе составляют 72 пикселя 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 —  $800 \times 600$ ,  $1024 \times 768$ ,  $1600 \times 1200$  или еще какое-нибудь, 72 пикселя могут выглядеть как 5 сантиметров на одном мониторе и всего 3 сантиметра — на другом. Однако пиксели 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 пикселях. 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 «M» 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.

В добавок 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 пикселя. 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  $1 \text{ vmin}$  равно  $1 \text{ vw}$ . Однако если ширина окна браузера больше высоты, то  $1 \text{ vmin}$  равно  $1 \text{ vh}$ .

## Единица `vmax`

Равна максимальному из значений величин `vh` или `vw`. Другими словами, если высота окна браузера меньше ширины, то значение  $1 \text{ vmax}$  равно  $1 \text{ vw}$ . Однако если ширина окна браузера меньше высоты, то  $1 \text{ vmax}$  равно  $1 \text{ vh}$ .

## 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`. Прочтайте 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 задаете 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 кавычках. 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 пикселам (с расчетом, что вы не изменяли базовый размер шрифта браузера); x-small — 10 пикселам, small — 13, large — 18, x-large — 24 и xx-large — 32 пикселам. 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 ем и пиксели 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 ем, пиксели 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 `ne` 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, *aБВГде* — буквы, которые именно так набраны в HTML-коде. Значение none отменит другие унаследованные свойства форматирования от элемента-предка и отобразит на экране *aБВГде*.

**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-space (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 пикселях) 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 пикселях, 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 третью — 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 эллипсоидных 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, форматирующее 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 пикселам и у элемента также есть отступы и границы, браузер отображает элемент на экране в пространстве, превышающем по ширине 300 пикселов.

**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 пикселов, отступы в 20 пикселов 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 пикселов, а если присвоено значение border-box, то браузеру отобразит `div`-элемент шириной 300 пикселов. 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 фоном элемента, 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 отображалось 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 пикселях и единицах em рассчитываются от верхнего левого угла элемента, поэтому, чтобы поместить изображение на расстоянии 5 пикселов от левого края и 10 пикселов 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 пикселях 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 фоне 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 пикселях, 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 форматируемого элемента относительно нижнего края ближайшего предка. Если форматируемый 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 сноска 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 форматируемый 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 пикселов. Последнее число — смещение слева — левый край области отсечения. В этом примере смещение равно `10px`, так что будет скрыта область размером 9 пикселов слева. Второе число определяет ширину области отсечения плюс значение смещения слева (последнее значение); если левый край области отсечения составляет 10 пикселов и вы хотите, чтобы видимая область была шириной 100 пикселов, 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 пикселам ( $30 + 5 = 35$  пикселов).

**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 форматируемого 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 форматируемого элемента относительно левого края ближайшего предка. Если форматируемый 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 пикселов 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 «рассыпляется» 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 установлено 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 умещающиеся 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 форматируемого элемента относительно правого края ближайшего предка. Если форматируемый 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 пикселов 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.

Скрытие 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 наслоения 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 налагаются) (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

Этот 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 — `1000ms`.

**Пример:** `animation-duration: 2s;`

## animation-timing-function

Задает 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 — `1000ms`.

**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. Этот 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 пикселях, 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

Задает 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 пикселов 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` устраниет 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, форматирующем 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 спонсор..."; }  
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 [tinyurl.com/4qunt](http://tinyurl.com/4qunt) и [tinyurl.com/b37oc](http://tinyurl.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, nw-resize, 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](http://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 получат 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 Орга 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](http://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](http://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](http://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](http://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.
- **Ресурс CSS3files** ([css3files.com](http://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](http://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](http://tinyurl.com/lhc6q7).
- **CSSCreator** ([tinyurl.com/ndexl](http://tinyurl.com/ndexl)). Оживленный форум, предлагающий помошь и советы по всем темам, начиная с базовой CSS-верстки и заканчивая усовершенствованными методами.
- **SitePoint** ([tinyurl.com/o7287qr](http://tinyurl.com/o7287qr)). Другая полезная группа CSS-гуру.
- **CSS-Tricks** ([tinyurl.com/o3o5z9z](http://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 паутине 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](http://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](http://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](http://tinyurl.com/yf5th8n)). Здесь собраны одни из лучших тематических ресурсов во Всемирной паутине, а в категории CSS вы найдете практически бесконечное количество ссылок, освещивающих 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](http://tinyurl.com/hw3jx)). Пошаговые пособия по созданию навигационных систем из неупорядоченных списков.
- **Адаптивные панели навигации с помощью CSS** ([tinyurl.com/p9bergv](http://tinyurl.com/p9bergv)). За этим названием скрывается учебник, содержащий пошаговые инструкции для создания адаптивной панели навигации.
- **Как создать адаптивное навигационное меню с помощью CSS** ([tinyurl.com/oduyusci](http://tinyurl.com/oduyusci)). Из этой статьи вы можете узнать, как использовать CSS-код для создания адаптивного навигационного меню.

## Онлайн-примеры

- **NavNav** ([navnav.co](http://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](http://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/h9fuyck](http://tinyurl.com/h9fuyck)). 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](http://tinyurl.com/o33nblg)). Содержит русскоязычные интерактивные уроки, обучающие различным способам CSS-верстки.
- **Бесплатные адаптивные CSS-макеты** ([tinyurl.com/ofb3th2](http://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](http://tinyurl.com/msapspz)) содержит множество примеров базовых шаблонов, созданных на основе фреймворка Pure.
- **Twitter Bootstrap** ([getbootstrap.com](http://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](http://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](http://csszengarden.com)). Самый главный сайт, посвященный CSS-верстке: много различных дизайнов для одного и того же HTML-кода.
- **CSS Line** ([cssline.com](http://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](http://awwwards.com)). Сайт, обладающий прекрасным дизайном, несмотря на свое название.
- **CSS Design Awards** ([cssdesignawards.com](http://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
  - ем, 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  
скрытие 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  
колонитулы, 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  
чертежование 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  
<img>, 257  
для печати, 271  
поля/отступы, 220  
<span>, 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
    - тер `fieldset`, 397
    - тер `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	<i>О. Сивченко</i>
Ведущий редактор	<i>Н. Гринчик</i>
Художник	<i>С. Заматевская</i>
Корректор	<i>Е. Павлович</i>
Верстка	<i>А. Барцевич</i>

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](http://www.chpk.ru). E-mail: [marketing@chpk.ru](mailto: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. «Электроразводная», 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, доб. 6243; e-mail: uchebnik@piter.com

---

↗ **Заказ книг по почте:** на сайте [www.piter.com](http://www.piter.com); тел.: (812) 703-73-74, доб. 6216;  
e-mail: books@piter.com

---

↗ **Вопросы по продаже электронных книг:** тел.: (812) 703-73-74, доб. 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](http://www.piter.com)
- по электронной почте: [books@piter.com](mailto:books@piter.com)
- по телефону: **(812) 703-73-74**

**ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:**

- Ⓐ Наложенным 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 сообщают по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте [www.piter.com](http://www.piter.com))
- Можно оформить доставку заказа через почтоматы, (адреса почтоматов можно узнать на нашем сайте [www.piter.com](http://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](mailto:titova@piter.com)  
**Москва** – Сергей Клебанов, (495) 234-38-15, [klebanov@piter.com](mailto:klebanov@piter.com)