4 Механизм наследования стилей

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

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

Что такое наследование?

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

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

ПРИМЕЧАНИЕ

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

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

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

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

Упрощение таблиц стилей через наследование

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

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

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

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

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

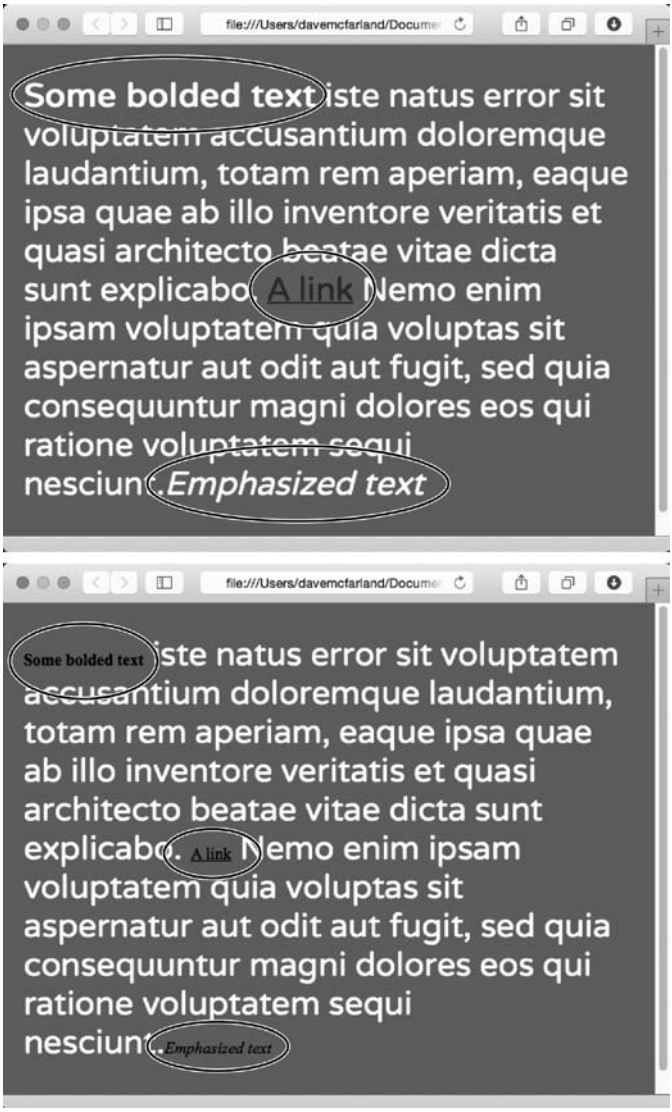

Рис. 4.1. Наследование позволяет копировать свойства из окружающих элементов

Ограничения наследования

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

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

ПРИМЕЧАНИЕ

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

Ниже приведены конкретные случаи, когда наследование не применяется.

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

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

ПРИМЕЧАНИЕ

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

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

ПРИМЕЧАНИЕ

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

Практикум: наследование

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

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

Одноуровневое наследование

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

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

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

ПРИМЕЧАНИЕ

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

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

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

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

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

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

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

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

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

Наследование при форматировании всей веб-страницы

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

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

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

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

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

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

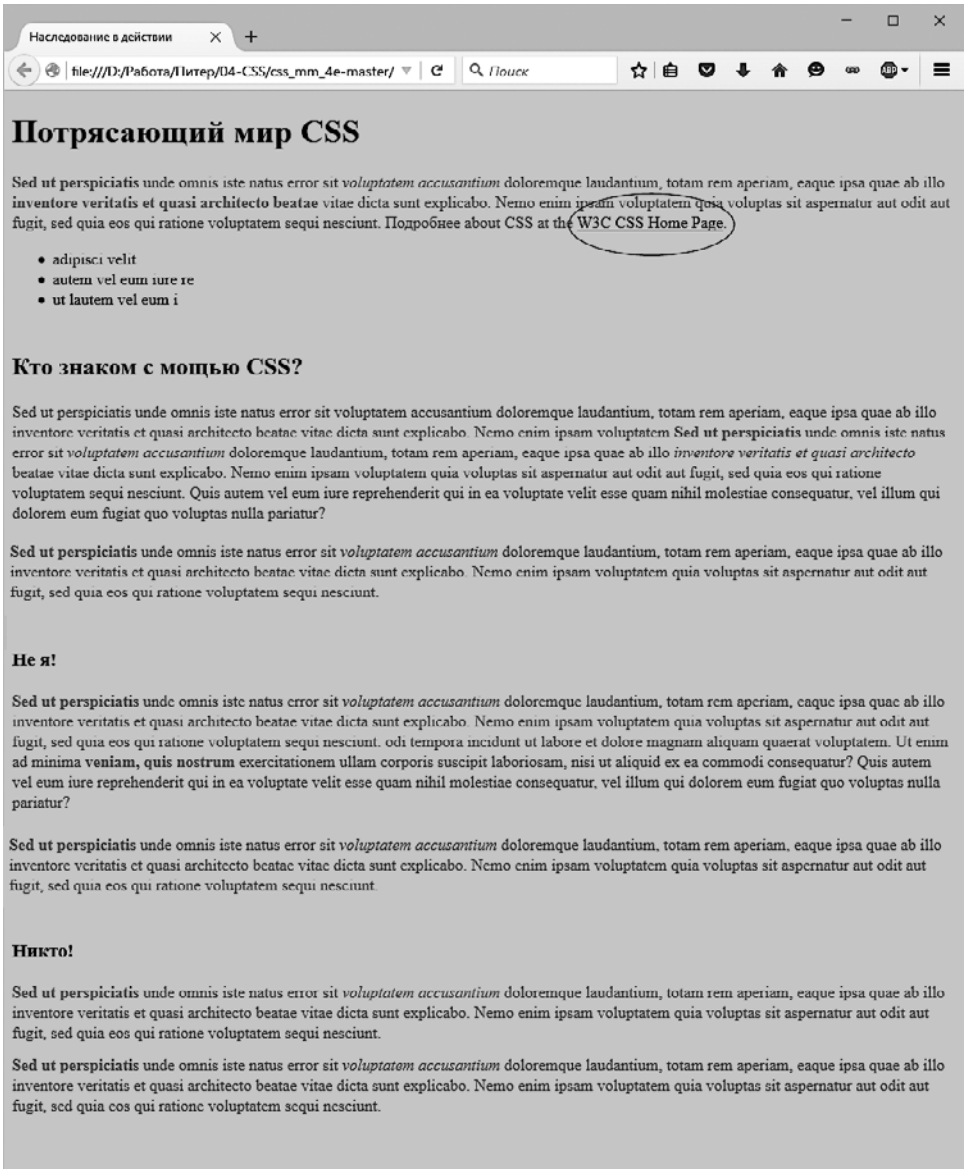

Рис. 4.3. Принцип наследования в действии! Текст, выделенный полужирным шрифтом, приобрел тот же цвет, что и абзац p, окружающий его. Но что это? Ссылка в конце первого абзаца осталась синей (выделена на рисунке). Почему так произошло, вы узнаете в следующей главе

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

Законченный стиль должен иметь следующий вид:

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

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

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

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

5. Сохраните и просмотрите веб-страницу в браузере.

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

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

ПРИМЕЧАНИЕ

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

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

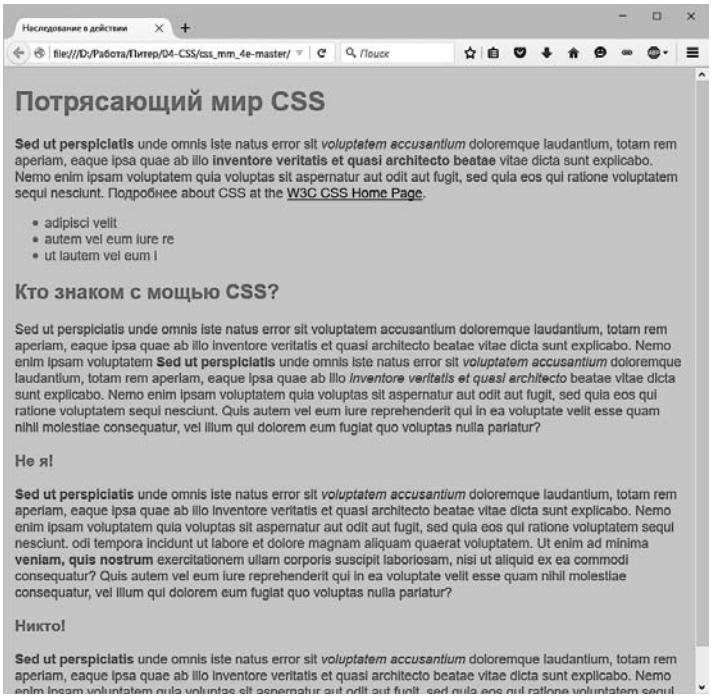

Рис. 4.4. Стиль, примененный к элементу body, переносит его свойства на все элементы, отображаемые в окне браузера. Благодаря этому очень легко форматировать эффекты на странице

Исключения механизма наследования

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

1. Вернитесь в редактор HTML-кода с открытым файлом `inheritance.html`.

   Дополним только что созданный стиль `p`.

2. Найдите определение стиля `p`, щелкните кнопкой мыши сразу за свойством цвета (`color: rgb(50,122,167);`) и нажмите клавишу Enter для перехода на новую строку.

   Сейчас мы создадим отступ слева для всех абзацев веб-страницы.

3. Добавьте два свойства к стилю следующим образом:

```
p {
  color: rgb(50,122,167);
  padding-left: 20px;
  border-left: solid 25px rgba(255,255,255,.5);
}
```

Этот код изменит отступ слева каждого абзаца и сместит текст таким образом, чтобы он не касался границы. Свойство `padding` создает отступ от границы размером 20 пикселов.

4. Сохраните файл и просмотрите его в браузере.

   Обратите внимание, что все абзацы `p` приобрели слева толстую светлую границу. Однако у элементов, *вложенных* в абзац `p` (например, `em`), нет такого отступа или границы (рис. 4.5). Это поведение браузера оправданно: веб-страница выглядела бы странно, если бы каждый элемент `em` и `strong` в абзаце имел дополнительную границу и отступ слева размером 20 пикселов!

   Если вы хотите увидеть, что бы случилось, если бы эти свойства наследовались, отредактируйте селектор `p` таким образом: `p, p *`, что сделает его групповым. Первая часть — это тот селектор `p`, который вы только что создали. А вторая часть — `p *` — буквально означает следующее: «выберите все элементы внутри абзаца `p` и примените к ним стиль» (`*` — универсальный селектор, был описан в предыдущей главе).

Рис 4.5. Большинство CSS-свойств наследуются вложенными элементами (например, цвет шрифта). Однако поля, отступы и границы являются исключением. Справочная информация относительно наследования CSS-свойств приведена в приложении 1

ПРИМЕЧАНИЕ

Финальный вариант веб-страницы, которую вы создали в этом практикуме, можно найти в папке 04_finished.

5 Управление сложной структурой стилей: каскадность

По мере того как вы будете создавать все более сложные таблицы стилей, вы все чаще будете задаваться вопросом: почему конкретный элемент веб-страницы выглядит именно так? Из-за особенностей наследования CSS, описанных в предыдущей главе, на любой элемент влияют окружающие его элементы. Стиль становится сложнокомбинированным. Например, элемент `body` может передать форматирование абзацу текста, а тот — гиперссылке, находящейся внутри абзаца. Другими словами, может произойти наследование CSS-свойств и `body`, и `p` *одновременно*.

Кроме того, может возникать конфликт: одно и то же CSS-свойство многократно описывается в различных стилях, которые затем относятся к одному элементу веб-страницы (например, стиль `p` во внешней таблице стилей и другой стиль `p` во внутренней). Вы можете наблюдать такую ситуацию, когда текст отображается ярко-синим цветом, несмотря на то, что установлен красный. Существует особая система, которая управляет взаимодействием стилей и определяет их приоритет при конфликте. Этот механизм называется *правилами каскадности* или просто *каскадностью*.

ПРИМЕЧАНИЕ

В данной главе описываются проблемы, возникающие при построении сложных таблиц стилей, работа которых основана на принципах наследования и использовании более сложных типов селекторов, таких как селекторы потомков (см. раздел «Идентификаторы: отдельные элементы веб-страницы» главы 3). Все правила достаточно логичны и понятны для опытного веб-дизайнера, но у начинающего все-таки может возникнуть множество сложностей. Можно сравнить обучение этому языку с овладением тонкостями налогового кодекса. Если у вас нет желания углубленно изучать все особенности языка CSS, просто пропускайте теоретический материал и переходите к выполнению заданий практикума. Вы всегда сможете вернуться к этой главе после того, как овладеете основами каскадных таблиц стилей.

Каскадность стилей

Каскадность — ряд правил, определяющих, какие именно свойства стилей применяются к элементам веб-страницы, то есть задающих последовательность приме-

нения многократно определенных стилей. Другими словами, каскадность указывает, каким образом браузер должен обработать сложную структуру, относящуюся к одному и тому же элементу, и что делать, если возникает конфликт свойств. Это происходит в двух случаях: из-за механизма наследования (одинаковое свойство наследуется от нескольких родительских элементов-предков) и когда один или более стилей применяются к одному элементу веб-страницы (например, вы применили к абзацу стиль с помощью класса, а также создали стиль для элемента `p` и оба стиля применяются к этому абзацу).

Объединение унаследованных стилей

Как вы узнали из предыдущей главы, наследование каскадных таблиц стилей гарантирует, что однородные, взаимосвязанные элементы веб-страницы (например, все слова в абзаце, даже если они являются вложенными гиперссылками или расположены в другом элементе) получают форматирование родительских элементов. Это избавляет от необходимости создания отдельных стилей для каждого элемента веб-страницы. Поскольку элемент может унаследовать свойства *любого* предка (например, ссылка, наследующая шрифт родительского элемента `p`), определить, почему конкретный элемент отформатирован именно так, может быть сложной задачей. Предположим такую ситуацию: к элементу `body` применен конкретный шрифт, к `p` — размер шрифта, а к `a` — цвет. Таким образом, любой элемент `a` абзаца унаследует стили `body` и `p`. Другими словами, унаследованные стили будут объединены, сформировав один сложный.

Страница, представленная на рис. 5.1, имеет три стиля: один для `body`, второй для `p` и третий для `strong`. CSS-код выглядит следующим образом:

```
body { font-family: Verdana, Arial, Helvetica, sans-serif; }
p { color: #F30; }
strong { font-size: 24px; }
```

Как показано на рисунке, элемент `strong` отформатирован с изменением шрифта, цвета и кегля. Конечный внешний вид элемента определяется сочетанием всех трех стилей (собственным, устанавливающим кегль (размер шрифта) и двумя наследованными от селекторов тегов `body` и `p`).

Элемент `strong` вложен в абзац, который, в свою очередь, вложен в `body`. Стили `strong` наследует у всех элементов-предков, получая форматирование шрифта (`font-family`) от элемента `body` и цвет (`color`) от абзаца `p`. Кроме того, `strong` имеет собственное правило, устанавливающее размер шрифта 24 пиксела. Конечный внешний вид элемента определяется сочетанием всех трех стилей. Другими словами, `strong` выглядит так, будто для него создали следующий стиль:

```
strong {
  font-family: Verdana, Arial, Helvetica, sans-serif;
  color: #F30;
  font-size: 24px;
}
```

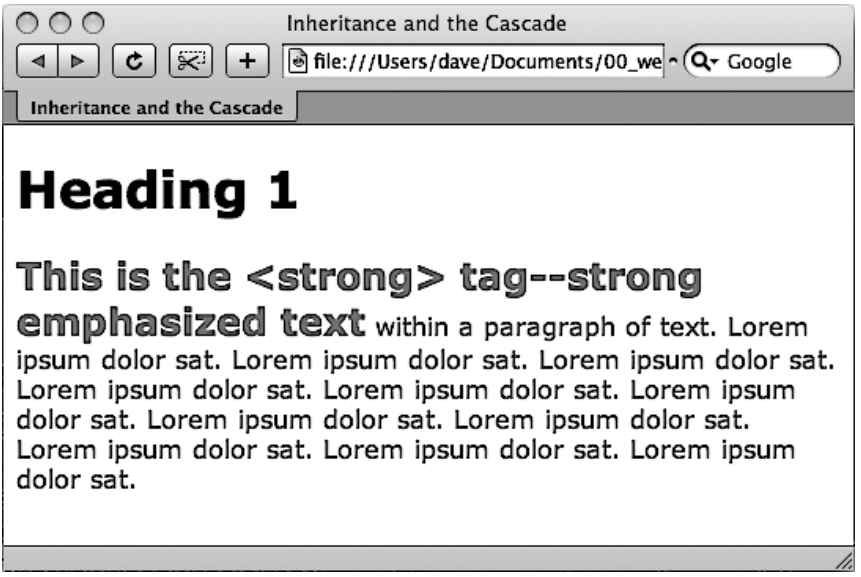

Рис. 5.1. Благодаря наследованию можно отформатировать один элемент несколькими стилями

Превосходство ближайших предков

Как видно из предыдущего примера, комбинирование стилей различных элементов с применением наследования создает полный набор форматирования. Но что произойдет в случае конфликта унаследованных свойств каскадных таблиц стилей? Представьте себе страницу, где вы установили красный цвет шрифта для элемента `body` и зеленый цвет шрифта для элемента `p`. Теперь предположим, что внутри абзаца имеется элемент `strong`. Он наследует стиль как у `body`, так и у элемента `p`. Так какого цвета текст внутри элемента `strong`: красный или зеленый? Правильный ответ: зеленый цвет, унаследованный от стиля абзаца. Браузер применит стиль, который является *самым близким* по отношению к форматируемому элементу.

В данном примере любые свойства, унаследованные от `body`, являются скорее общими. Они относятся ко всем элементам веб-страницы. Однако стиль `p` более близок к `strong` и, можно сказать, находится по соседству с ним. Форматирование `p` применяется непосредственно к абзацу и его вложенным элементам.

По сути, если элемент не имеет собственного, явно определенного стиля, то при возникновении конфликтов с унаследованными свойствами одержат победу ближайшие предки (рис. 5.2, *1*).

Это еще один пример, позволяющий разобраться в концепции каскадности. Если есть CSS-стиль, определяющий цвет текста для таблицы `table`, и еще один, определяющий *другой* цвет для ячейки `td`, то элементы, заключенные в ячейки данной таблицы (`td`), — элементы абзаца, заголовка, маркированного списка — унаследуют цвет стиля `td`, поскольку он является ближайшим предком.

Преимущества непосредственно примененного стиля

Единственный стиль, обладающий наивысшим приоритетом, является ближайшим предком в «генеалогическом» дереве каскадных таблиц стилей. Это тот стиль, который напрямую применен к элементу. Предположим, что цвет шрифта устанавливается для элементов `body`, `p` и `strong`. Стиль абзаца `p` более специфичен, чем `body`, но `strong` — еще специфичнее, чем все остальные. Он форматирует только элементы `strong`, игнорируя любые конфликтующие свойства, унаследованные от других элементов (см. рис. 5.2, *2*). Другими словами, свойства стиля, явно определенного для элемента, отменяют любые унаследованные.

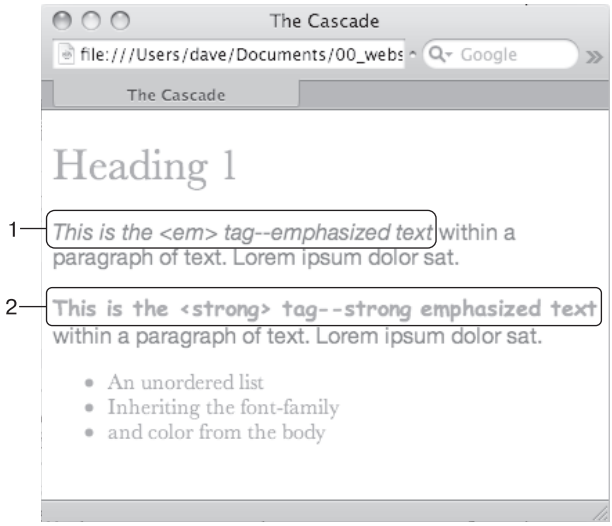

Рис. 5.2. Браузеры определяют, какие свойства применять при возникновении конфликтов с наследуемыми стилями

Это правило объясняет, почему некоторые свойства не применяются. Гиперссылка, находящаяся внутри абзаца, текст которого отформатирован красным шрифтом, по-прежнему будет отображена в окне браузера синей с подчеркиванием. Это происходит потому, что есть собственный предопределенный стиль для элемента привязки `a`. Таким образом, унаследованный цвет текста не будет применен.

ПРИМЕЧАНИЕ

О том, как игнорировать встроенные стили браузера для элемента a и установить желаемый стиль для гиперссылок, будет рассказано в главе 9.

Множество стилей для одного элемента

Наследование — один из способов форматирования элемента несколькими стилями. Однако можно определить много стилей, которые будут применены *непосредственно*

к элементу. Рассмотрим такой случай: имеется прикрепленная к веб-странице внешняя таблица стилей, среди прочих содержащая стиль для элемента абзаца p. В коде самой страницы есть внутренняя таблица, которая *также* содержит стиль для элемента абзаца p. К одному из этих элементов на странице применен еще и класс. Так, для одного элемента абзаца p непосредственно определено три различных стиля. Какой (или *какие*) из них должен использовать браузер?

Ответ: все зависит от множества факторов — типов стилей, порядка, в котором они были созданы. Браузер сам выбирает, применить один или несколько стилей одновременно. Рассмотрим ситуации, когда множество стилей может применяться к одному и тому же элементу.

- **К элементу одновременно применен селектор тега и класс.** Например, стиль тега для элемента h2 и класс .leadHeadline могут быть представлены следующим HTML-кодом:

```
<h2 class="leadHeadline">Для вас будущее наступило!</h2>
```

  К элементу h2 будут применены оба стиля.

ПРИМЕЧАНИЕ

Описание того, что произойдет в случае конфликта стилей, следует далее.

- **Одинаковые имена стилей встречаются несколько раз в таблице.** К примеру, это может быть два стиля — групповой селектор (см. раздел «Форматирование групп элементов» главы 3), например .leadHeadline, .secondaryHeadline, .newsHeadline и класс .leadHeadline в той же таблице. Форматирование элемента с классом leadHeadline будет определяться обоими стилями.
- **К элементу одновременно применены стили класса и идентификатора.** Это может быть идентификатор #banner и класс .news. HTML-код имеет вид: . К элементу div будут применены оба стиля.
- **С веб-страницей связано несколько таблиц, и в каждой из них содержится стиль с одинаковым именем.** Стили с одинаковыми именами могут использоваться во внешней и во внутренней таблице стилей. Или же один и тот же стиль может появиться в нескольких внешних таблицах стилей, каждая из которых связана с одной и той же страницей.
- **Единственный элемент веб-страницы может быть объектом воздействия сложных селекторов.** Это обычная ситуация, когда вы используете селекторы потомков (см. раздел «Классы: точное управление» главы 3). Допустим, на веб-странице имеется элемент div (например,) и внутри него заключен абзац с классом: <p class="byline">. В этом случае к этому абзацу будут применимы следующие селекторы:

```
#mainContent p
#mainContent .byline
p.byline
.byline
```

Если к конкретному элементу веб-страницы применено несколько стилей, то браузер объединяет их свойства при условии, что они не конфликтуют между со-

бой. Приведенный ниже пример разъясняет этот принцип. Предположим, есть абзац, в котором указаны имя дизайнера веб-страницы и ссылка на адрес его электронной почты. HTML-код может выглядеть следующим образом:

```
<p class="byline">Создано <a href="mailto:jean@cosmofarmer.com">Жорой Ивановым</
a></p>
```

Между тем в таблице стилей веб-страницы присутствуют три стиля для форматирования гиперссылки:

```
a { color: #6378df; }
p a { font-weight: bold; }
.byline a { text-decoration: none; }
```

Первый стиль окрашивает все элементы `a` в зеленовато-голубой цвет; второй стиль форматирует все элементы `a`, находящиеся в абзаце `p`, полужирным шрифтом; а третий стиль убирает подчеркивание ссылок, вложенных в элементы с классом `byline`.

Воздействие всех трех стилей распространяется на такой часто используемый элемент, как `a`. Ни одно из правил этих стилей не конфликтует с остальными. Ситуация похожа на случай, рассмотренный выше в разделе «Объединение унаследованных стилей»: стили объединяются между собой и образуют один комбинированный «суперстиль», содержащий все три свойства. Таким образом, данная гиперссылка отображается зеленовато-голубым полужирным шрифтом и без подчеркивания.

ПРИМЕЧАНИЕ

Имейте в виду, что на стиль форматирования этой ссылки также могут влиять унаследованные свойства. Например, может быть унаследовано начертание шрифта абзаца. Лучше понять работу механизма каскадности вам помогут несколько инструментов, описанных во врезке «ЧаВо» ниже.

Особенности каскадности: какие стили имеют преимущество

Предыдущий пример слишком прост. Но что получится, если каждый из трех стилей ссылок, приведенных выше, имеет *свое* определение начертания в свойстве `font-family`? Какой из них выберет браузер?

Если вы внимательно читали книгу, то помните, что механизм каскадности устанавливает несколько правил. *Побеждают (имеют преимущество) свойства самого близкого по отношению к форматируемому элементу, самого специфичного стиля.* Однако, как и в примере со стилями, не совсем понятно, какой из них является наиболее специфичным. К счастью, CSS предлагает метод *определения приоритетов*. Он основан на присвоении значений в условных единицах каждому типу селекторов: селекторам тегов, классам, идентификаторам и т. д. Система работает так.

- Селектор тегов имеет специфичность, равную **1 условной единице**.
- Класс — **10 условных единиц**.
- Идентификатор — **100 условных единиц**.
- Строчный стиль — **1000 условных единиц**.

ПРИМЕЧАНИЕ

Математические расчеты, используемые для определения приоритетов, на самом деле немного сложнее. Но эта формула работает во всех случаях, кроме самых странных и запутанных. Чтобы узнать о том, как браузеры рассчитывают приоритеты, посетите страницу w3.org/TR/css3-selectors/#specificity.

Чем больше числовое значение, тем выше специфичность (значимость) данного типа селектора. Предположим, вы создали три стиля:

- стиль тега для элемента `img` (специфичность = 1);
- класс с именем `.highlight` (специфичность = 10);
- идентификатор с именем `#logo` (специфичность = 100).

Веб-страница содержит следующий HTML-код: `<img id="logo" class="highlight" src="logo.gif" />`. Если определить одинаковый атрибут во всех трех стилях (например, свойство `border`), то будет применен стиль идентификатора (`#logo`), как наиболее специфичного.

ПРИМЕЧАНИЕ

Псевдоэлемент (например, ::first-child) обрабатывается браузером как селектор тегов и имеет специфичность 1 пункт. Псевдокласс (например, :link) рассматривается как класс и имеет специфичность 10 пунктов (см. раздел «Псевдоклассы и псевдоэлементы» главы 3).

Поскольку селекторы потомков состоят из нескольких простых, например `content p` или `h2 strong`, определить их специфичность сложнее: необходимо вычислить суммарное значение их приоритетов (табл. 5.1).

Таблица 5.1. Когда к единственному элементу применяется несколько стилей, браузер должен определить, какой из них будет применен при возникновении конфликта

Селектор	Идентификатор	Класс	Тег	Итого
p	0	0	1	1
.byline	0	10	0	10
p.byline	0	10	1	11
#banner	100	0	0	100
#banner p	100	0	1	101
#banner .byline	100	10	0	110
a:link	0	10	1	11
p:first-line	0	0	2	2
h2 strong	0	0	2	2
#wrapper #content .byline a:hover	200	20	1	221

ПРИМЕЧАНИЕ

Наследуемые свойства вообще лишены специфичности. Так, даже если элемент унаследует, например, селектор #banner, то его свойства в любом случае будут заменены теми, что непосредственно относятся к этому элементу.

Разрешение конфликтов: побеждает последний стиль.

Два стиля могут иметь одинаковый приоритет. Конфликт свойств с одинаковой специфичностью может произойти при двойном определении одинаковых селекторов. У вас может быть селектор элемента p во внутренней таблице и такой же во внешней. Или два различных стиля могут иметь одинаковый приоритет. В таком случае более значимым будет последний определенный стиль.

Ниже представлен пример HTML-кода.

```
<p class="byline">Создано <a class="email" href="mailto:jean@cosmofarmer.com">Жорой Ивановым</a></p>
```

В таблице для веб-страницы, содержащей вышеприведенные абзац и гиперссылку, присутствует два стиля:

```
p .email { color: blue; }
.byline a { color: red; }
```

Оба стиля имеют специфичность 11 (10 — для класса и 1 — для селектора тега) и относятся к элементу a. Конфликт этих стилей очевиден. Какой цвет выберет браузер для окрашивания гиперссылки в приведенном абзаце? Красный, так как он указан последним в таблице стилей.

ЧАВО

Инструменты в помощь

Существует ли какое-нибудь вспомогательное средство, чтобы представить в понятной форме воздействие, оказываемое механизмом каскадности на конечный дизайн веб-страницы?

Попытки определить все входы и выходы унаследованных свойств и конфликтующие стили сбивали с толку многих исследователей. Более того, применение математических приемов для определения специфики стиля как-то не привлекает обычного веб-дизайнера, особенно при использовании огромных таблиц стилей с большим количеством селекторов потомков.

Во всех современных браузерах имеется встроенная помощь в виде инспектора. Проще всего обследовать элемент на странице и все каскадные таблицы стилей, влияющие на его внешний вид, щелкнув правой кнопкой мыши (щелкнув кнопкой мыши с нажатой клавишей ^ в операционной системе OS X) на элементе (заголовке, ссылке, абзаце или изображении) и выбрав в контекстном меню пункт Исследовать элемент (Inspect Element). Откроется панель (обычно в нижней части веб-страницы), отображающая исходный код страницы с выбранным HTML-элементом. (В браузере Safari сначала нужно установить флажок Показывать меню "Разработка" в строке меню (Show Develop Menu) на вкладке Дополнения (Advanced) окна, открывающегося выбором команды меню Safari ▸ Настройки (Preferences).)

В правой части панели вы увидите стили, примененные к элементу. Обычно это «вычисленный» стиль, итоговая сумма всех CSS-свойств, примененных к элементу путем наследования и каскадирования, или некий синтезированный стиль элемента. Чуть ниже будут показаны правила стилей, примененные к элементу, перечисленные в порядке от более специфичных (в верхней части списка) к менее (в нижней части списка).

Вероятнее всего, вы увидите, что в перечне стилей некоторые свойства зачеркнуты. Это свидетельствует о том, что данное свойство либо не применяется к элементу, либо было замещено более специфичным стилем. Чтобы просмотреть два кратких учебных пособия по использованию инструментария разработчика по анализу CSS, посетите сайты tinyurl.com/oetspo9 и tinyurl.com/nb3ls5u.

Теперь представьте, что два стиля поменялись местами и таблица стилей имеет следующий вид:

```
.byline a { color: red; }
p .email { color: blue; }
```

В данном случае гиперссылка будет синей. Стиль с селектором `p .email` расположен в таблице после строки `.byline a`, поэтому его свойства имеют преимущество.

Теперь рассмотрим, что произойдет, если имеются конфликтующие стили (или их свойства) во внешней и внутренней таблицах стилей. В этом случае важна последовательность размещения на веб-странице (в HTML-коде файла). Если вы сначала добавляете внутреннюю таблицу, используя элемент `style` (см. главу 2), а *затем* присоединяете внешнюю таблицу с помощью элемента `link`, то будет применен стиль последней (в сущности, это принцип, который только что был описан: *значение имеет последний из конфликтующих стилей*). Вывод: будьте последовательны в размещении в коде веб-страницы ссылки на внешнюю таблицу стилей. Сначала ее нужно присоединить, а только затем добавлять внутренние таблицы, если абсолютно невозможно обойтись без одного или нескольких стилей, применяемых к одной странице.

УСТРАНЕНИЕ ОШИБОК

Аннулирование специфичности

Каскадные таблицы стилей предоставляют возможность полностью отменить специфичность стилей. Вы можете использовать этот прием, чтобы никакой другой более специфичный стиль не заместил конкретное свойство элемента веб-страницы. Для этого вставьте после нужного свойства значение `!important`.

Рассмотрим пример. Допустим, существует два стиля:

```
.nav a { color: red; }
a { color: teal !important; }
```

При обычном раскладе ссылка, вложенная в элемент с классом `.nav`, была бы окрашена в красный цвет, так как стиль, определенный селектором `.nav a`, специфичнее, чем селектор тега `a`. Однако добавление слова `!important` подразумевает, что данное свойство всегда будет иметь больший приоритет. Так, в вышеприведенном примере все ссылки веб-страницы, в том числе вложенные с классом `nav`, будут отображены зеленовато-голубым цветом.

Обратите внимание, что вы применяете метку `!important` к отдельному свойству, а не ко всему стилю, поэтому нужно добавить слово `!important` в конце каждого свойства, которое не должно быть замещено. В заключение нужно сказать: когда для двух одинаковых свойств различных стилей указано слово `!important`, опять вступает в силу правило специфичности и приоритет имеет более специфичный атрибут из отмеченных.

Будьте осторожны, применяя метку `!important`. Поскольку этот способ кардинален, при слишком частом его использовании ваши стили не будут соответствовать обычным правилам каскада, что приведет к «эскалации» использования метки `!important`. Другими словами, чтобы скомпенсировать специфичность свойства `!important` в одном стиле, вы будете применять его в другом. Затем, чтобы скомпенсировать действие метки `!important` во втором стиле, вам придется добавить его в третий стиль и т. д. Поэтому не рекомендуется использовать метку `!important` слишком часто. Прежде чем добавлять ее, испробуйте другой способ преодолеть конфликт, например переименуйте или поменяйте местами стили в таблице.

Управление каскадностью

Как вы могли заметить, чем больше CSS-стилей создано, тем больше вероятность запутаться в них. Например, можно создать класс, устанавливающий для шрифта определенное начертание и размер, но применение его к абзацу ни к чему не приводит. Эта проблема обычно связана с механизмом каскадности. Даже когда вы абсолютно уверены в конечном результате, все равно может существовать более специфичный стиль.

Есть несколько вариантов решения этой проблемы. Во-первых, можно использовать слово `!important` (как описано во врезке выше), чтобы *гарантировать* применение конкретного свойства. Этот способ не совсем удобен, так как трудно предугадать, что вы не захотите отменить такую специфичность свойства. Рассмотрим два других метода управления каскадностью.

Изменение приоритетов

На рис. 5.3 и 5.4 приведен пример, когда определенный стиль тега проигрывает в каскадной игре. К счастью, в большинстве случаев можно запросто изменить специфичность одного из конфликтующих стилей и приберечь метку `!important` для по-настоящему безысходных случаев. На рис. 5.3 два стиля форматируют первый абзац. Класс `.intro` не так специфичен, как `#sidebar p`. Таким образом, свойства класса `.intro` не будут применены к абзацу. Чтобы увеличить специфичность класса, добавьте к стилю идентификатор `#sidebar .intro`.

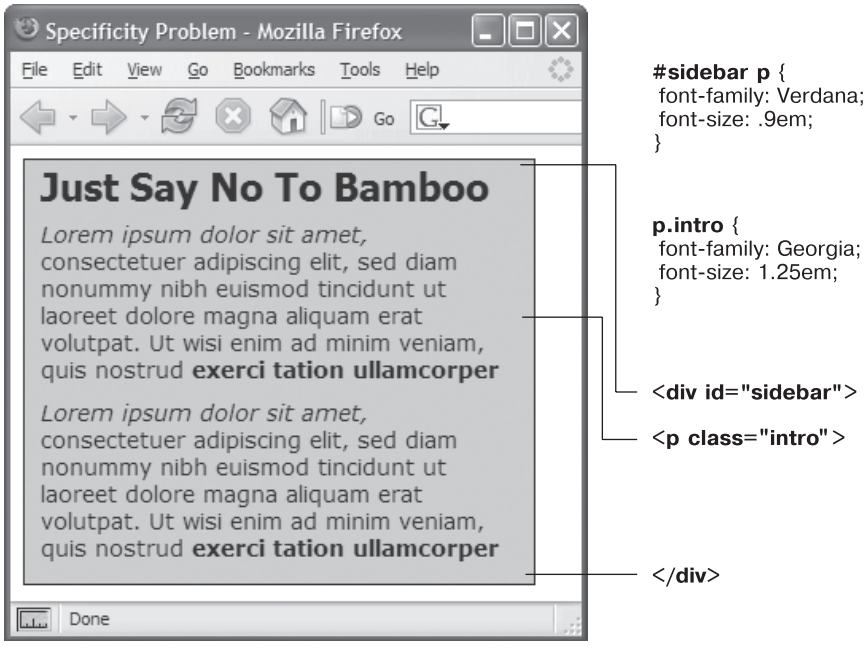

Рис. 5.3. Даже если класс применен к конкретному элементу (см. первый абзац), его свойства не всегда имеют эффект

В данном случае абзац размещен внутри элемента `div` с идентификатором `#sidebar`, поэтому селектор потомков `#sidebar p` специфичнее класса `.intro`. Решение: необходимо повысить значимость класса `.intro`, добавив перед ним идентификатор — `#sidebar p.intro` (рис. 5.4).

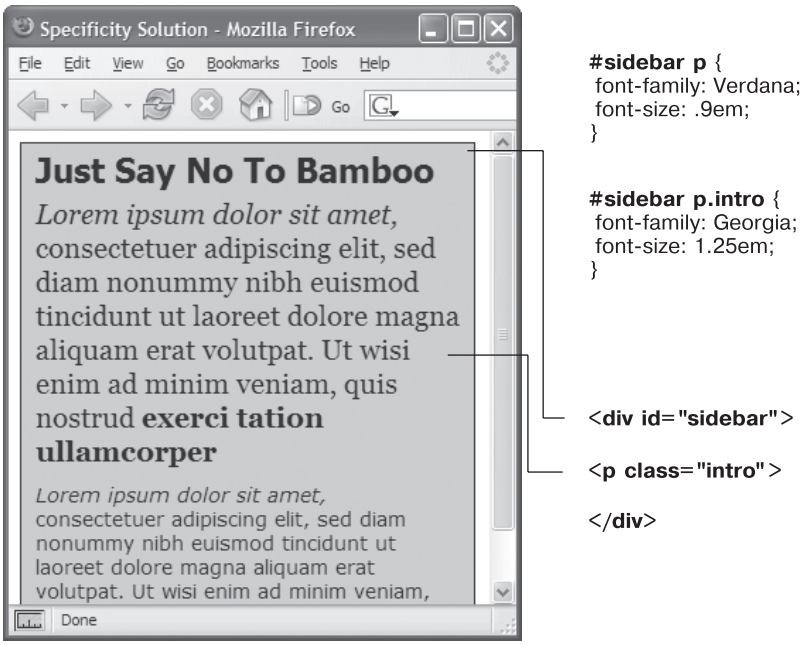

Рис. 5.4. Можно придать большую значимость стилевому классу, добавив перед ним идентификатор

Но простое добавление дополнительных селекторов с целью «победы» свойств стиля может привести к так называемым *войнам специфичности*, когда у вас в конечном итоге получаются таблицы стилей, содержащие очень длинные и запутанные имена стилей наподобие `#home #main #story h1`. Нужно стараться избегать стилей такого типа и использовать максимально короткие селекторы.

Выборочное игнорирование приоритетов

Можно тонко проработать дизайн веб-страниц, *выборочно* отменяя стили. Предположим, вы создали внешнюю таблицу и назвали ее `styles.css`, связав со всеми страницами сайта.

Этот CSS-файл содержит общие определения стилей дизайна страниц: шрифт и цвет для элементов заголовков `h1`, стиль элементов формы и т. д. Возможно, вы хотите, чтобы на главной (домашней) странице заголовок `h1` выглядел иначе, чем он отображен на остальных. Например, был выделен крупным полужирным шрифтом. Или абзац должен быть отформатирован шрифтом меньшего размера, чтобы вместить больший объем информации. Другими словами, вы все еще хотите ис-

пользовать большинство стилей файла styles.css, но необходимо отменить несколько атрибутов отдельных элементов (h1, p и т. д.).

Один из способов — создание внутренней таблицы, содержащей стили, которые вы хотите отменить и переопределить. Предположим, в файле styles.css имеется следующий стиль:

```
h1 {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 24px;
  color: #000;
}
```

Вы хотите, чтобы заголовок h1 главной веб-страницы был отображен крупным шрифтом красного цвета. Для этого добавьте во внутреннюю таблицу следующий стиль:

```
h1 {
  font-size: 36px;
  color: red;
}
```

В данном случае к элементу h1 на главной странице сайта будет применен шрифт Arial (из внешней таблицы стилей), но в то же время он будет окрашен в красный цвет размером 36 пикселов (стили, определенные во внутренней таблице стилей).

СОВЕТ

Убедитесь, что вы ссылаетесь на внешнюю таблицу стилей перед кодом внутренней в разделе заголовка HTML-страницы. Это гарантирует, что нужные стили будут иметь преимущество в тех случаях, когда специфичность идентична.

Другой метод заключается в создании еще одной внешней таблицы. Например, таблицы home.css, которую нужно будет присоединить к главной веб-странице в дополнение к styles.css. Файл home.css будет содержать те стили styles.css, которые вы хотите переопределить. Для правильной работы файл home.css должен быть присоединен после styles.css в HTML-коде веб-страницы:

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

СОВЕТ

Еще один способ выполнить точное постраничное форматирование веб-страниц основан на использовании различных имен класса для элемента body веб-страниц разного типа. Например, чтобы изменить дизайн отдельных веб-страниц, применяются идентификаторы .review, .story, .home, а затем создаются селекторы потомков. Эта методика рассматривалась в главе 3.

Как избежать конфликтов приоритетов

Многие веб-дизайнеры предпочитают вместо идентификаторов использовать классы. Одна из причин состоит в том, что идентификаторы обладают очень большой специфичностью, поэтому для их отмены требуется более высокая специфичность. Зачастую это приводит к войнам специфичности, при которых таблицы стилей

загружаются с излишне пространными и сложными селекторами. Суть этой проблемы проще объяснить на примере. Предположим, что в HTML-коде вашей страницы есть следующий фрагмент:

```
<div class="article">
<p>Абзац</p>
<p>Другой абзац</p>
<p class="special">Особенный абзац</p>
</div>
```

Вы решили, что нужно текст абзаца внутри `div`-контейнера `article` сделать красным, и создали следующий селектор потомков:

```
#article p { color: red; }
```

Но затем вам захотелось, чтобы текст одного абзаца с классом `special` был синим. Если просто создать класс, вы не получите желаемого результата.

```
.special { color: blue; }
```

Как уже говорилось, когда определяется, какое из свойств нужно применить к элементу, браузер использует для разрешения конфликтов стилей простую математическую формулу: браузеры присваивают идентификатору значение `100`, классу — значение `10`, а селектору тегов — значение `1`. Поскольку селектор `#article p` составлен из одного идентификатора и одного тега (суммарный показатель специфичности — `101`), он заменяет собой простой стиль класса, заставляя вас изменить селектор:

```
#article .special {color: blue; }
```

К сожалению, это изменение оказывается причиной возникновения еще двух проблем. Во-первых, селектор становится длиннее, и во-вторых, теперь этот синий цвет применяется только тогда, когда класс `special` появляется внутри какого-нибудь элемента с идентификатором `article`. Например, если вы скопируете HTML-код `<p class="special">Особенный абзац</p>` и вставите его в какой-нибудь другой позиции страницы, текст уже не будет синим. Иначе говоря, использование идентификаторов делает селекторы не только длиннее, но и бесполезнее.

А теперь посмотрим, что получится, если просто заменить все идентификаторы классами. Предыдущий код HTML приобретет следующий вид:

```
<div class="article">
<p>Абзац</p>
<p>Другой абзац</p>
<p class="special">Особенный абзац</p>
</div>
```

И код CSS можно заменить следующим:

```
.article p { color: red; }
p.special { color: blue; }
```

Первый стиль, `.article p`, является селектором потомков с уровнем специфичности 11 условных единиц. Второй стиль, `p.special`, также имеет уровень 11 условных единиц (один селектор тега и один класс) и означает «применить следующие свойства к любому абзацу `p` с классом `special`». Теперь, если вырезать этот HTML-

код и вставить его в какую-нибудь другую позицию страницы, вы получите синий текст, обусловленный стилем, к чему, собственно, вы и стремились.

Этот только один из примеров, но не составит труда найти таблицы стилей с абсурдно длинными селекторами наподобие `#home #article #sidebar #legal p` и `#home #article #sidebar #legal p.special`.

Идентификаторы могут быть и полезными. Например, многие системы управления контентом (CMS) используют их для определения уникальных элементов страницы. Их превосходящая значимость может легко превысить специфичность других стилей. Однако не стоит злоупотреблять идентификаторами. Они не дают ничего, что нельзя было бы получить с использованием простого класса или селектора тега, а их высокая специфичность может только привести к ненужному усложнению таблиц стилей.

ПРИМЕЧАНИЕ

Более подробные аргументы, почему следует избегать применения идентификаторов, изложены на странице tinyurl.com/6ge7z86.

С чистого листа

Браузеры применяют к элементам свои собственные стили: например, шрифт заголовков `h1` крупнее `h2`, они оба выделены полужирным начертанием, в то время как шрифт текста абзацев меньше и не выделен полужирным шрифтом; ссылки подчеркнутые и имеют синий цвет, а у маркированных списков есть отступ. В стандарте HTML нет ничего, что бы определяло все это форматирование: браузеры просто добавляют его для того, чтобы простой HTML-код при визуализации был более читабельным. Разные браузеры обрабатывают элементы очень похоже, но все же неодинаково.

Так, например, браузеры Chrome и Firefox используют свойство `padding` для создания отступа в маркированных списках, а Internet Explorer применяет свойство `margin`. Кроме того, вы сможете найти небольшие различия в размерах элементов в разных браузерах и обнаружить вовсе вводящее в заблуждение использование отступов самыми распространенными на сегодняшний день браузерами. Из-за этих несоответствий вы столкнетесь с проблемами, когда, например, Firefox добавит отступ от верхнего края, а Internet Explorer этого не сделает. Такого рода проблемы не ваша вина — они вытекают из различий стилей, встроенных в браузер.

Для предотвращения несоответствия стилей между браузерами лучше всего начинать таблицу стилей с чистого листа. Другими словами, удалить встроенное в браузер форматирование и добавить собственное. Концепция устранения стилей браузера называется *сбросом стилей* (CSS Reset).

В частности, есть базовый набор стилей, который вы должны включить в верхнюю часть своей таблицы стилей. Они устанавливают базовые значения для свойств, которые обычно по-разному обрабатываются во всех браузерах.

Рассмотрим шаблон сброса стандартных стилей:

```
html, body, div, span, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li,
fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed, figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary, time, mark, audio, video {
```

```
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  vertical-align: baseline;
}

article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav,
section {
  display: block;
}
body {
  line-height: 1.2;
}
ol {
  padding-left: 1.4em;
  list-style: decimal;
}
ul {
  padding-left: 1.4em
  list-style: square;
}
table {
  border-collapse: collapse;
  border-spacing: 0;
}
```

ПРИМЕЧАНИЕ

Показанный выше код сброса стилей взят из известного и авторитетного источника, составленного Эриком Мейером, который можно найти по адресу tinyurl.com/2amlyf.

Первый стиль — очень длинный групповой селектор, затрагивающий наиболее распространенные элементы и *«обнуляющий»* их. Он удаляет поля и отступы, устанавливая 100%-ный размер шрифта и убирая полужирное начертание. Благодаря этому шагу ваши элементы смотрятся практически одинаково (рис. 5.5). Но так и нужно — ведь вы хотите начать с чистого листа, а затем добавить собственное форматирование, чтобы все браузеры согласованно отображали ваш HTML-код.

Второй селектор (`article, aside, details...`) является еще одним групповым селектором, помогающим устаревшим браузерам правильно отображать новые HTML5-элементы. Третий селектор тега (`body`) устанавливает пространство между строками в абзаце (свойство `line-height`). Сведения о свойстве `line-height` будут приведены в следующей главе.

ПРИМЕЧАНИЕ

Вам не нужно вводить этот код самостоятельно. Вы найдете файл с именем reset.css в папке 05 архива с примерами с сайта github.com/mrightman/css_4e. Просто скопируйте стили из этого файла и вставьте их в свою таблицу стилей.

Еще один вариант сброса — использовать файл normalize.css — бесплатную таблицу стилей с открытым исходным кодом, которая позволяет различным браузерам отображать одни и те же элементы в согласованном виде. Он широко используется веб-дизайнерами. Найти файл normalize.css можно по адресу tinyurl.com/oy58gya.

Четвертый и пятый селекторы тегов (для элементов `ol` и `ul`) устанавливают согласованные отступы от левого края и определенное форматирование (в главе 6 вы изучите форматирование списков), а последний стиль упрощает добавление рамок к ячейкам таблицы (польза от установки этого стиля будет рассмотрена в главе 11).

Практикум: механизм каскадности

В этом практикуме вы увидите, как элементы взаимодействуют между собой и конфликтуют, что приводит к неожиданным результатам. Для начала взгляните на простую страницу, упомянутую выше, на которой были сброшены стандартные стили. Кроме того, есть несколько других стилей, обеспечивающих простую разметку. Затем мы создадим два стиля и понаблюдаем за действием механизма каскадности. Мы также рассмотрим, как наследование влияет на элементы веб-страницы и как браузер решает конфликты CSS-стилей. Наконец, вы узнаете, как решаются проблемы механизма каскадности.

Перед началом урока нужно загрузить файлы примеров, расположенные по адресу github.com/mrightman/css_4e. Перейдите по ссылке и загрузите ZIP-архив с файлами. Файлы текущего практикума находятся в папке 05.

Сброс стандартных стилей и создание стилей с чистого листа

Для начала взгляните на страницу, с которой будете работать.

1. В браузере откройте страницу `cascade.html` из папки 05 (см. рис. 5.5).

   Страница выглядит очень просто: две колонки (одна из них на синем фоне) и много однотипного текста. К этому файлу уже применялись некоторые стили, поэтому откройте CSS-код в текстовом редакторе и просмотрите его.

2. Откройте файл `styles.css` из папки 05 в текстовом либо в HTML-редакторе.

   Это внешняя таблица стилей, которую использует файл `cascade.html`. В ней уже есть несколько стилей: первая группа сбрасывает стандартные стили, что мы обсуждали на предыдущей странице. Они устраняют основные стили браузера, поэтому весь текст пока выглядит одинаково. Скоро вы будете создавать собственные стили, чтобы эта страница смотрелась более эффектно.

   Последние два стиля — классы `.main` и `.sidebar` — создают две колонки, показанные на рис. 5.5. HTML-код разделен на два элемента `div`, каждый из которых имеет собственный класс. Стили классов здесь, по существу, размещают два элемента `div` так, чтобы они отображались друг рядом с другом в виде колонок (как управлять макетом страницы и создавать колонки, вы узнаете в части III).

   Сначала вы добавите пару стилей, чтобы улучшить общий вид страницы и ее верхний заголовок.

3. В файле `styles.css` добавьте два этих стиля в нижней части таблицы стилей после строки с последней скобкой } класса `.sidebar`:

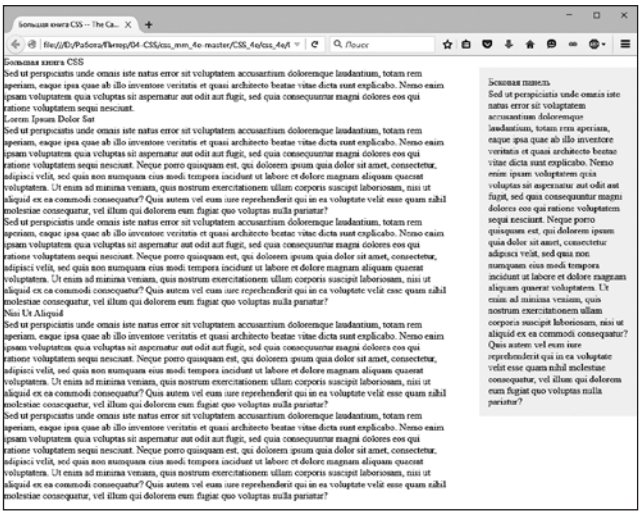

Рис. 5.5. Сброс стандартных стилей на этой странице устраняет небольшие различия в том, как разные браузеры отображают основные HTML-элементы. Устраняются и любые различия в отображении элементов. Ваша задача — взять пустой холст и отформатировать элементы нужным образом

```
body {
  color: #B1967C;
  font-family: "Palatino Linotype", Baskerville, serif;
  padding-top: 115px;
  background: #CDE6FF url(images/bg_body.png) repeat-x;
  max-width: 800px;
  margin: 0 auto;
}
h1 {
  font-size: 3em;
  font-family: "Arial Black", Arial, sans-serif;
  margin-bottom: 15px;
}
```

Первый стиль добавляет фоновое изображение и цвет для страницы, а также устанавливает для нее фиксированную ширину. Сохранив этот файл и просмотрев страницу `cascade.html` в браузере (рис. 5.6), вы заметите, что эти атрибуты не наследуются другими элементами — то же изображение, например, не повторяется за элементами заголовков или абзацев.

Свойства `font-family` и `color`, с другой стороны, наследуются, так что другие элементы на странице теперь используют шрифт Arial и имеют коричневый цвет. Тем не менее вы увидите, что, хоть верхний заголовок такого же цвета, как и остальной текст на странице, у него другой шрифт — вот наглядный пример каскадности в действии. Для форматирования элемента `h1` не было назначено никакого цвета, так что заголовок наследует коричневый цвет, который был применен к элементу `body`. Но, поскольку стиль элемента `h1` определяет шрифт, он замещает унаследованный шрифт от стиля `body`.

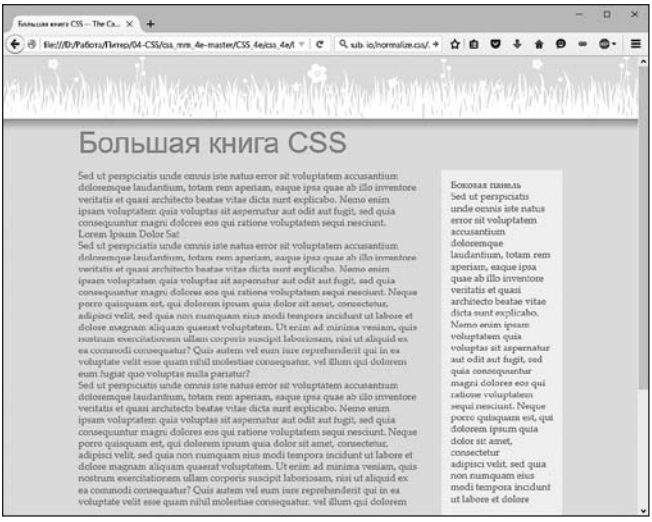

Рис. 5.6. Наследование и каскадность в действии: элемент h1 в верхней части этой страницы наследует цвет шрифта из форматирования элемента body, но получает размер и шрифт из стиля элемента h1

Комбинирование стилей

В этом примере мы создадим два стиля. Один стиль будет форматировать все заголовки второго уровня веб-страницы, а другой — более специфичный стиль — будет повторно форматировать те же заголовки, но в крупной, главной колонке веб-страницы.

1. В файле `styles.css` добавьте следующий стиль в конец таблицы стилей:

```
h2 {
  font-size: 2.2em;
  color: #AFC3D6;
  margin-bottom: 5px;
}
```

Этот стиль меняет цвет и увеличивает размер шрифта элемента `h2`, а также добавляет немного пространства под ним. Если вы просмотрите страницу в браузере, то увидите, что заголовки `h2` из основной колонки и те же заголовки из правой колонки похожи друг на друга.

Далее вы создадите стиль для форматирования *только* тех заголовков второго уровня, которые находятся в главной колонке.

2. Вернитесь в HTML-редактор к файлу `styles.css`, щелкните кнопкой мыши сразу после стиля элемента `h2` и нажмите клавишу Enter, чтобы перейти на новую строку. Добавьте следующий стиль:

```
.main h2 {
  color: #E8A064;
  border-bottom: 2px white solid;
```

```
  background: url(images/bullet_flower.png) no-repeat;
  padding: 0 0 2px 80px;
}
```

Вы только что создали селектор потомков, описанный ранее в книге, форматирующий все элементы h2, расположенные *внутри* элемента с классом main. Две колонки текста на этой странице заключаются в элементы div с разными именами классов. У большей, расположенной слева колонки класс носит имя main, поэтому такой особый стиль будет применяться только к элементам h2 внутри этого раздела div.

Рассматриваемый стиль похож на тот, который вы создали в практикуме в главе 2 в шаге 17. Он добавляет подчеркивание и значок с изображением цветка к заголовку. Этот стиль также определяет оранжевый цвет шрифта.

3. Сохраните таблицу стилей и снова просмотрите страницу в браузере (рис. 5.7).

   Вы заметите, что у всех заголовков второго уровня (у двух в основной колонке и у одного в боковой) одинаковый размер шрифта, но у тех двух, которые расположены в основной колонке, также есть подчеркивающая линия и изображение цветка.

   Поскольку стиль .main h2 специфичнее простого стиля h2, то при возникновении каких-либо конфликтов между двумя стилями (в данном случае в значении свойства color) свойства стиля .main h2 приоритетнее. Таким образом, хотя шрифту заголовков второго уровня в основной колонке передается синий цвет стиля h2, оранжевый цвет более специфичного стиля .main h2 приоритетнее. Однако, поскольку для стиля .main h2 не указан размер шрифта или нижнего поля, заголовки в основной колонке получают эти свойства от стиля h2.

Преодоление конфликтов

Поскольку свойства каскадных таблиц стилей конфликтуют, когда несколько стилей применяются к одному и тому же элементу, иногда вы можете обнаружить, что ваши страницы выглядят не так, как вы этого ожидали.

Когда это происходит, вам нужно установить, почему это произошло, и внести изменения в селекторы таблиц стилей, чтобы каскадность приводила к нужному результату.

1. Вернитесь к HTML-редактору и открытому файлу styles.css.

   Сейчас вы создадите новый стиль для форматирования исключительно абзацев в основной колонке.

2. Добавьте следующий стиль в конец таблицы стилей:

```
.main p {
  color: #616161;
  font-family: "Palatino Linotype", Baskerville, serif;
  font-size: 1.1em;
  line-height: 150%;
  margin-bottom: 10px;
  margin-left: 80px;
}
```

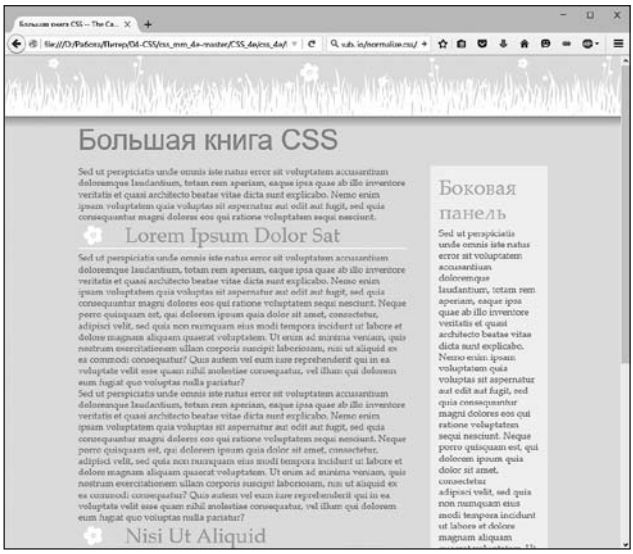

Рис. 5.7. Стили h2 и .main h2 применяются к заголовкам второго уровня в левой колонке этой страницы, причем стиль .main h2 — только к ним

Просмотрев страницу в браузере, вы увидите, что изменился цвет, размер и шрифт текста абзацев, строки растянулись (свойство `line-height`) и изменились отступы абзацев слева и снизу.

Далее вы отформатируете первый абзац более крупным шрифтом с полужирным начертанием, чтобы привлечь к нему внимание. Самый простой способ отформатировать один-единственный абзац — создать класс и применить его к этому абзацу.

3. Добавьте последний стиль в конце таблицы стилей:

```
p.intro {
  color: #6A94CC;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 1.2em;
  margin-left: 0;
  margin-bottom: 15px;
}
```

Этот стиль изменяет цвет, шрифт и размер, а также незначительно регулирует отступы. Все, что вы должны сделать, — применить класс к HTML-коду.

4. Откройте файл `cascade.html` в HTML-редакторе. Найдите элемент `p`, расположенный после загловка `<h1>Большая книга CSS</h1>` и строки `<div class="main">`, и добавьте следующий атрибут класса:

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

5. Просмотрите страницу в браузере.

И... абзац никак не изменился. Согласно правилам каскадности, `.intro` — простой класс, а `.main p` — селектор потомков и состоит из имен класса и тега. Они

добавлены для создания более специфичного стиля, поэтому его свойства решают любой конфликт между ним и стилем `.intro`.

Для того чтобы заработал стиль `.intro`, необходимо немного «укрепить» его, наделив этот селектор большей специфичностью.

6. Вернемся к файлу `styles.css` в HTML-редакторе и изменим имя стиля с `.intro` на `p.intro`.

   Убедитесь в том, что между `p` и `.intro` нет пробела. По сути, вы создали связь: `.main p` — один класс и один селектор тега и `p.intro` — один тег и один класс. Оба имеют уровень специфичности 11, но поскольку `p.intro` появляется в таблице стилей после `.main p`, именно этот селектор побеждает и его свойства применяются к абзацу. (Чтобы преодолеть конфликт, можно создать и еще более специфичный стиль — `.main .intro`.)

7. Просмотрите страницу в браузере (рис. 5.8).

   Теперь в абзаце цвет шрифта сменился на голубой, шрифт изменился и стал крупнее, а также отсутствует отступ слева. Если бы у вас не было четкого понимания принципа каскадности, вы бы ломали голову над тем, почему этот класс не работал в первый раз.

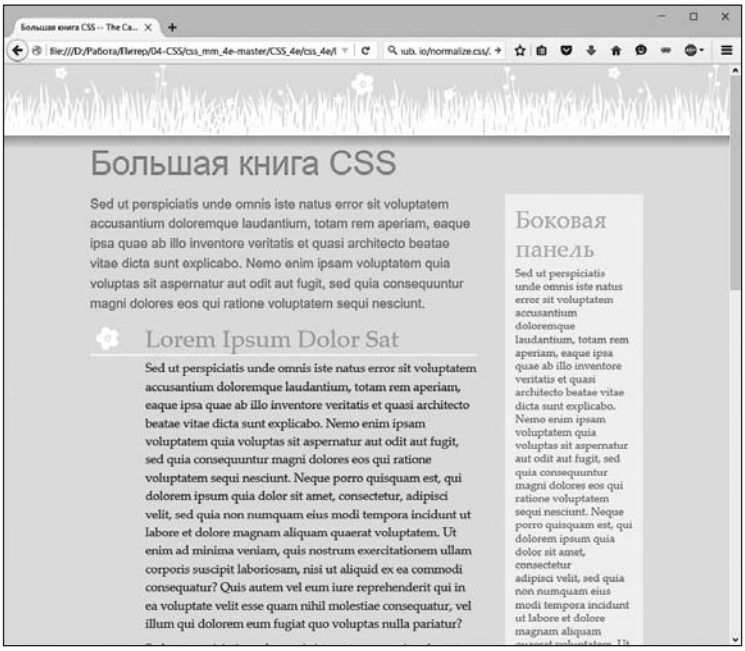

Рис. 5.8. Даже на простой странице с небольшим набором стилей внешний вид элементов зачастую является комбинацией свойств различных стилей

В этой и четырех предыдущих главах мы рассмотрели основы каскадных таблиц стилей. Теперь, в части II, пришло время применить полученные знания для создания настоящих полноценных дизайнов веб-страниц.