Содержание

Об авторах 11
Об изображении на обложке 12

Введение 13

Принятые соглашения 13
Синтаксис описания свойств CSS 14
Файлы примеров и цветные иллюстрации 17
Ждем ваших отзывов! 18
Благодарности 19

Глава 1. CSS и HTML-документы 21

История CSS 21
Элементы 23
Взаимодействие CSS и HTML 28
Таблицы стилей 37
Медиа-запросы 41
Запросы поддержки 47
Резюме 50

Глава 2. Селекторы 51

Базовые правила 51
Группирование 56
Селекторы идентификаторов и классов 61
Селекторы атрибутов 68
Структура документа 78
Селекторы псевдоклассов 87
Псевдоэлементы 119
Резюме 122

Глава 3. Приоритетность и каскадирование стилей 125

Приоритетность 125
Наследование 131
Каскадирование 134
Резюме 140

Глава 4. Значения и единицы измерения 141

Ключевые слова, строки и другие текстовые значения 141
Числовые и процентные значения 148
Длина и расстояние 149
Вычисляемые значения 159
Значения атрибутов 161

Цвета 161
Углы 171
Время и частота 172
Положение 173
Пользовательские переменные 173

Глава 5. Шрифты 177

Семейства шрифтов 177
Правило `@font-face` 182
Насыщенность шрифта 195
Размер шрифта 203
Начертание шрифта 214
Уплотнение и расширение текста 218
Кернинг шрифта 220
Варианты строчных букв 221
Особенности шрифта 224
Генерирование начертаний 226
Свойство `font` 228
Замена шрифтов 232
Резюме 234

Глава 6. Текстовые свойства 235

Отступы и выравнивание 235
Выравнивание по высоте 244
Интервалы между словами и символами 254
Регистр символов 258
Оформление текста 260
Оптимизация отображения текста 264
Текст, отбрасывающий тень 266
Обработка пробелов 267
Разрывы и переносы текстовых строк 271
Направление письма 277
Резюме 284

Глава 7. Основы визуального оформления документа 285

Контейнеры 285
Представление элемента 288
Строчные элементы 312
Резюме 342

Глава 8. Поля, отступы, границы и рамки 343

Контейнеры элемента 343
Поля 346
Границы 356
Внешний контур 395

Отступы 401
Резюме 410

Глава 9. Цвета, фон и градиенты 411

Цвета 411
Фон элемента 416
Градиенты 474
Тень элемента 509
Резюме 512

Глава 10. Обтекание и форма элемента 513

Обтекание 513
Отмена обтекания 530
Обтекание по форме элемента 534
Резюме 549

Глава 11. Позиционирование 551

Общие положения 551
Свойства смещения 553
Высота и ширина 556
Переполнение и обрезка содержимого 560
Сокрытие элемента 562
Абсолютное позиционирование 563
Фиксированное положение 581
Относительное позиционирование 583
Липкое позиционирование 585
Резюме 589

Глава 12 Гибкая верстка 591

Основы верстки flex-контейнеров 591
Flex-контейнеры 598
Упорядочение flex-элементов 616
Flex-контейнер 617
Выравнивание flex-элементов 618
Выравнивание flex-элементов вдоль поперечной оси 627
Свойство `align-self` 634
Выравнивание содержимого 635
Flex-элементы 641
Свойства форматирования отдельных flex-элементов 646
Свойство `flex` 646
Свойство `flex-grow` 648
Свойство `flex-shrink` 655
Свойство `flex-basis` 666
Свойство общего назначения `flex` 676
Порядок следования элементов 682

Глава 13. Верстка по сетке 687
Создание grid-контейнера 687
Основные определения 690
Размещение линий сетки 692
Привязка элементов к сетке 720
Поток grid-элементов 736
Автоматическое размещение линий сетки 741
Свойство настройки сетки общего назначения 744
Интервалы 748
Выравнивание grid-элементов 755
Размещение на слоях и порядок наложения 761
Резюме 763

Глава 14. Верстка таблиц 765
Форматирование таблиц 765
Границы ячеек таблицы 779
Размер таблицы 788
Резюме 799

Глава 15. Списки и генерируемое содержимое 801
Списки 801
Генерируемое содержимое 813
Шаблон счетчика 829
Резюме 851

Глава 16. Трансформации 853
Система координат 853
Трансформация элементов 857
Другие свойства трансформации 877
Резюме 889

Глава 17. Переходы 891
Переходы в CSS 891
Свойства настройки переходов 893
Независимый обратный переход 916
Анимируемые свойства и значения 920
Переход как эффект оформления 925
Печать переходов 925

Глава 18. Анимация 927
Определение ключевых кадров 928
Настройка анимации 929
Селекторы ключевых кадров 930
Анимация элементов 936
Анимация общего назначения 971

Приоритетность и порядок выполнения анимации 975
Эпилептические приступы и потеря ориентации при просмотре анимации 977
События анимации и вендорные префиксы 978
Печать анимации 980

Глава 19. Фильтры, смешивание цветов, обтравочные контуры и маски 981

CSS-фильтры 981
Наложение элементов и смешивание цветов 988
Наложение фоновых слоев 996
Обтравочные контуры и маскирование 1001
Маски 1008
Подгонка объектов 1023

Глава 20. Форматирование, зависящее от носителя 1027

Стилевое форматирование, зависящее от устройства 1027
Бумажные носители 1039
Резюме 1054

Приложение А. Анимируемые свойства 1055

Приложение Б. Базовые свойства 1061

Приложение В. Таблица соответствия цветов 1071

Предметный указатель 1075

ГЛАВА 3

Приоритетность и каскадирование стилей

В предыдущей главе рассматривались принципы иерархического структурирования документов и селекторы, указывающие на элементы, к которым применяются стилевые правила. Каждый документ состоит из большого количества иерархически упорядоченных элементов, что позволяет селекторам обращаться к ним, основываясь на информации о взаимосвязях элементов разных уровней, их атрибутах, контекстных данных и других сведениях. Иерархическая структура лежит в основе механизма стилевого оформления документа и является ключевым элементом в понимании одного из главных принципов его реализации — наследования свойств.

Под наследованием понимают частичную или полную передачу стилевого оформления элемента его потомкам. Определяя форматирование для каждого из элементов документа, пользовательский агент должен учитывать не только все унаследованные им свойства, но и *приоритетность* применения правил, а также их происхождение. Процесс получил название *каскадирование*, или каскадное применение стилей. В этой главе описана взаимосвязь трех главных принципов, влияющих на порядок применения стилевых правил ко всем элементам документа: приоритетность, наследование и каскадирование. Различие между наследованием и каскадированием проще всего понять на примере правила `h1 {color: red; color: blue;}`. Каскадирование применяется для определения конечного форматирования элемента `h1`, а наследование объясняет, почему у элемента `span`, вложенного в элемент `h1`, синий цвет.

Приоритетность

В главе 2 описано огромное количество способов представления элементов в стилевых правилах. Реальность такова, что внешний вид одного и того же элемента чаще всего назначается несколькими правилами, в которых используются разные селекторы. Рассмотрим три пары правил, каждая из которых определяет форматирование разных элементов.

```
h1 {color: red;}
body h1 {color: green;}

h2.grape {color: purple;}
h2 {color: silver;}
```

```
html > body table tr[id="totals"] td ul > li {color: maroon;}
li#answer {color: navy;}
```

Только одно из двух правил каждой пары применяется к целевым элементам. Но как узнать, какой цвет из двух возможных получит каждый элемент?

Порядок применения стилевых правил в документе определяется их *приоритетностью* (или *специфичностью*). Пользовательский агент определяет приоритетность селекторов всех без исключения правил, применяемых в документе, и включает их в соответствующие объявления. Если одному и тому же элементу одновременно назначаются противоречивые стилевые правила, то будет применяться правило с большей приоритетностью.

В документе порядок применения правил определяется не только приоритетностью селекторов. На него также оказывает влияние каскадирование, детально рассмотренное в разделе “Каскадирование”.

Приоритетность селектора определяется его компонентами и представляется четырехзначным выражением формата `0,0,0,0`. На приоритетность селектора оказывают влияние следующие факторы.

- Каждый идентификатор (атрибут `id`) увеличивает приоритетность на величину `0,1,0,0`.
- Селекторы классов, псевдоклассов и атрибутов добавляют к приоритетности величину `0,0,1,0`.
- За каждый селектор элементов и псевдоэлементов начисляется приоритетность `0,0,0,1`. В CSS2 селекторы псевдоэлементов в расчете приоритетности участие не принимали, но уже в CSS2.1 этот недостаток был устранен, и они вносят свой вклад в конечное значение.
- Комбинаторы и универсальные селекторы на значение приоритетности не влияют.

Проще всего научиться определять приоритетность правил на реальных примерах.

```
h1 {color: red;}                    /* приоритетность = 0,0,0,1 */
p em {color: purple;}               /* приоритетность = 0,0,0,2 */
.grape {color: purple;}             /* приоритетность = 0,0,1,0 */
*.bright {color: yellow;}           /* приоритетность = 0,0,1,0 */
p.bright em.dark {color: maroon;}   /* приоритетность = 0,0,2,2 */
#id216 {color: blue;}               /* приоритетность = 0,1,0,0 */
div#sidebar *[href] {color: silver;} /* приоритетность = 0,1,1,1 */
```

Если применить второе и пятое правило из приведенных выше примеров к элементу `em`, то он получит темно-бордовый цвет (`maroon`), поскольку у пятого правила значение приоритетности больше, чем у второго.

В качестве упражнения вычислим значение приоритетности для следующих правил и определим победителя в каждой из пар.

```
h1 {color: red;}          /* 0,0,0,1 */
body h1 {color: green;}   /* 0,0,0,2 (приоритетнее)*/

h2.grape {color: purple;} /* 0,0,1,1 (приоритетнее) */
h2 {color: silver;}       /* 0,0,0,1 */

html > body table tr[id="totals"] td ul > li {color: maroon;}
                          /* 0,0,1,7 */
li#answer {color: navy;}  /* 0,1,0,1 (приоритетнее) */
```

Правила с большей приоритетностью обозначены в комментариях. Обратите внимание на принципы сравнения значений. Во второй паре большей приоритетностью обладает правило `h2.grape`, поскольку значение `0,0,1,1` больше `0,0,0,1`. В третьей паре более приоритетным будет второе правило, так как `0,1,0,1` больше, чем `0,0,1,7`. Второй разряд всегда весомее первого, поэтому значение `0,0,1,0` всегда больше `0,0,0,13`.

Чем больше порядок разряда, тем весомее его вклад в общее значение приоритетности. Таким образом, `1,0,0,0` больше любых других значений, первый разряд которых содержит `0` (независимо от чисел в остальных разрядах). Значение `0,1,0,1` вполне закономерно превышает `0,0,1,7`, поскольку в его третьем разряде указано `1`, а в этой позиции второго значения находится `0`.

Приоритетность объявлений

Значение приоритетности селектора назначается всем объявлениям, к которым он имеет отношение. Рассмотрим следующий пример:

```
h1 {color: silver; background: black;}
```

Пользовательский агент представляет это правило в виде двух более простых правил.

```
h1 {color: silver;}
h1 {background: black;}
```

Оба правила имеют значение приоритетности `0,0,0,1`, которое применяется к каждому из объявлений. Подобный подход характерен для любых других сгруппированных объявлений. Проанализируем следующее правило:

```
h1, h2.section {color: silver; background: black;}
```

Пользовательский агент представляет его в виде такого набора правил.

```
h1 {color: silver;}             /* 0,0,0,1 */
h1 {background: black;}         /* 0,0,0,1 */
h2.section {color: silver;}     /* 0,0,1,1 */
h2.section {background: black;} /* 0,0,1,1 */
```

Разделение правила на составляющие компоненты позволяет правильно оценить его приоритетность при сбое одного из объявлений. В качестве примера рассмотрим приоритетность следующих правил.

```
h1 + p {color: black; font-style: italic;}              /* 0,0,0,2 */
p {color: gray; background: white; font-style: normal;} /* 0,0,0,1 */
*.aside {color: black; background: silver;}             /* 0,0,1,0 */
```

Результат их применения к документу, представленному приведенным ниже фрагментом HTML-кода, показан на рис. 3.1.

```
<h1>Greetings!</h1>
<p class="aside">
It's a fine way to start a day, don't you think?
</p>
<p>
There are many ways to greet a person, but the words are not as
important as the act of greeting itself.
</p>
<h1>Salutations!</h1>
<p>
There is nothing finer than a hearty welcome from one's fellow man.
</p>
<p class="aside">
Although a thick and juicy hamburger with bacon and mushrooms runs
a close second.
</p>
```

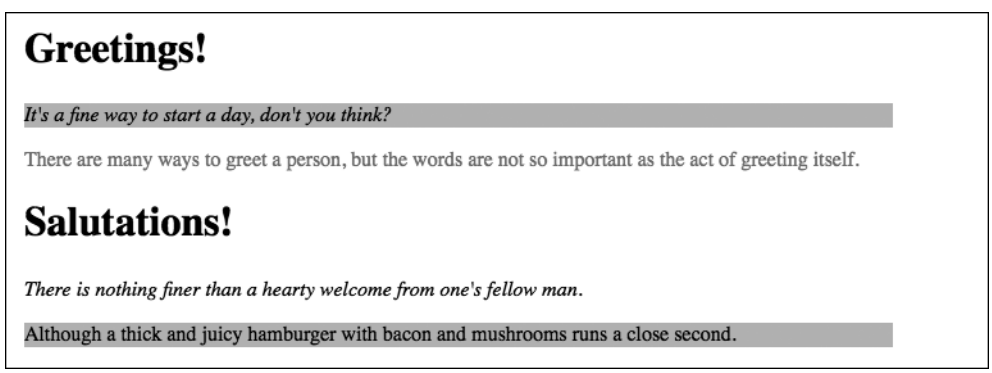

Рис. 3.1. Форматирование документа правилами с разной приоритетностью

Конечные стилевые правила, применяемые к элементам документа, определяются пользовательским агентом в результате сравнений значений их приоритетности. Каждому элементу назначается форматирование, имеющее наибольшую приоритетность. Подобный анализ необходимо проводить для всех без исключения объявлений, правил и селекторов. К счастью, эта задача решается пользовательским агентом полностью автоматически. От приоритетности правил также зависит порядок их каскадирования, о чем рассказано далее.

Приоритетность универсального селектора

При расчете значений приоритетности универсальный селектор не учитывается. Иными словами, приоритетность правил с универсальным селектором представлена

значением 0,0,0,0 и не равнозначна таковой в правилах с отсутствующей приоритетностью (рассмотрены в разделе "Наследование"). Таким образом, с помощью следующих двух правил абзацам, заключенным в элемент div, назначается черный цвет, а всем остальным элементам — серый.

```
div p {color: black;} /* 0,0,0,2 */
* {color: gray;}      /* 0,0,0,0 */
```

Как и следовало ожидать, добавление универсального селектора в правило, включающее селекторы других типов, не приводит к изменению значения его приоритетности. Исходя из этого, можно смело утверждать, что приведенные ниже правила обладают одинаковой приоритетностью.

```
div p         /* 0,0,0,2 */
body * strong /* 0,0,0,2 */
```

По сравнению с универсальными селекторами комбинаторы вообще не учитываются при вычислении уровня приоритетности — на порядок применения правил в документе они не влияют.

Приоритетность селекторов идентификаторов и атрибутов id

При определении приоритетности важно понимать различие между селекторами идентификаторов и селекторами атрибутов id. Вернемся к рассмотрению третьей пары правил, описанных выше.

```
html > body table tr[id="totals"] td ul > li {color: maroon;}
                                /* 0,0,1,7 */
li#answer {color: navy;}    /* 0,1,0,1 (приоритетнее) */
```

Легко заметить, что селектор идентификатора #answer (второе правило) добавляет к приоритетности правила значение 0,1,0,0, в то время как вклад в нее селектора атрибута [id="totals"] составляет всего 0,0,1,0. Таким образом, в результате применения следующих двух правил к элементу с атрибутом id, имеющим значение meadow, он окрашивается в зеленый цвет.

```
#meadow {color: green;}       /* 0,1,0,0 */
*[id="meadow"] {color: red;}  /* 0,0,1,0 */
```

Приоритетность встроенных стилей

До этого момента рассматривались правила, приоритетность которых представлялась значениями, начинающимися с 0 (в четвертом разряде). А все потому, что наибольшую приоритетность имеют объявления встроенных стилей. Рассмотрим следующий пример CSS-кода и фрагмент HTML-документа.

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

```
<h1 style="color: green;">The Meadow Party</h1>
```

Применение указанного правила к элементу h1 приводит к окрашиванию его текста зеленым цветом, независимо от форматирования, устанавливаемого другими

правилами. Заголовки первого уровня будут зелеными благодаря большей приоритетности — 1,0,0,0.

Буквально это означает, что применение к таким элементам правил других типов, например включающих селекторы атрибута id, не изменяет форматирование. Давайте чуть изменим последний пример, представив элемент h1 следующим образом.

```
h1#meadow {color: red;}

<h1 id="meadow" style="color: green;">The Meadow Party</h1>
```

Благодаря использованию встроенного стиля текст элементов h1 по-прежнему окрашивается зеленым цветом.

Важность стилей

Отдельные стили настолько важны, что должны применяться первостепенно, независимо от их базового уровня приоритетности. В CSS такие стили называются важными (по вполне очевидным причинам) и обозначаются с помощью специального ключевого слова !important, добавляемого в конец объявления, как раз перед точкой с запятой:

```
p.dark {color: #333 !important; background: white;}
```

При неправильном расположении ключевого слова !important в объявлении правило будет проигнорировано. Оно добавляется исключительно перед завершающей точкой с запятой, и ни в каком другом месте. К объявлению важных стилей нужно относиться крайне внимательно, особенно в свойствах, описываемых сразу несколькими ключевыми словами, например в таком случае:

```
p.light {color: yellow; font: smaller Times, serif !important;}
```

Если расположить ключевое слово !important в любом другом месте объявления свойства font, то пользовательский агент не сможет определить, к чему именно оно относится, и отменит сразу все правило.

Те читатели, которые имеют опыт программирования, будут крайне удивлены синтаксисом объявления важных стилей, поскольку восклицательный знак обычно обозначает логическое отрицание и в данном случае может трактоваться как “not important”. Как бы там ни было, в CSS он лишен данного смысла и всего лишь указывает на важность объявления. С этим ничего не поделаешь, но обязательно нужно учитывать данный нюанс при написании стилевых правил.

В действительности объявления, включающие ключевое слово !important, не изменяют приоритетность правила, а рассматриваются отдельно от них. Приоритетность важных объявлений устанавливается отдельно от приоритетности объявлений регулярных типов. Таким образом, противоречивость важных стилей устраняется отдельно от регулярных стилей, и наоборот. Но при возникновении конфликтной ситуации между ними предпочтение всегда отдается важным стилям.

На рис. 3.2 показан результат применения следующего форматирования к указанному фрагменту HTML-документа.

```
h1 {font-style: italic; color: gray !important;}
.title {color: black; background: silver;}
* {background: black !important;}

<h1 class="title">NightWing</h1>
```

Рис. 3.2. Важные стили характеризуются большей приоритетностью, чем регулярные

Детально важные стили и их назначение описаны в разделе “Каскадирование”.

Наследование

На порядок применения стилевых правил в документе оказывает влияние не только их приоритетность, но и еще одна ключевая концепция CSS: *наследование*. Наследование представляет собой механизм передачи стилевого форматирования от родительского элемента к его потомкам. Например, устанавливая цвет элементов `h1`, вы изменяете цвет текста не только заголовков первого уровня, но и всех его дочерних элементов.

```
h1 {color: gray;}

<h1>Meerkat <em>Central</em></h1>
```

В данном случае серый цвет назначается исходному тексту заголовка, а также тексту дочернего элемента `em`, наследующего свойство `color` от элемента `h1`. Если бы наследования не происходило, то у элемента `em` сохранялся бы цвет по умолчанию — черный. Для установки им одинакового цвета пришлось бы использовать два разных правила с одинаковыми значениями свойства `color`.

В качестве следующего примера рассмотрим неупорядоченный список, форматирование элемента `ul` которого определяется таким правилом:

```
ul {color: gray;}
```

Предполагается, что стилевое оформление элемента `ul` будет распространяться на все элементы списка и содержимое вложенных в них элементов. Благодаря принципу наследования наши ожидания полностью оправдываются, и неупорядоченный список приобретает вид, показанный на рис. 3.3.

- Oh, don't you wish
- That you could be a fish
- And swim along with me
- Underneath the sea

1. Strap on some fins
2. Adjust your mask
3. Dive in!

Рис. 3.3. Наследование стилей

С наследованием проще всего знакомиться, представив иерархическую структуру документа в графическом виде. На рис. 3.4 показана иерархическая структура простейшего документа, включающего всего два списка: неупорядоченный и упорядоченный.

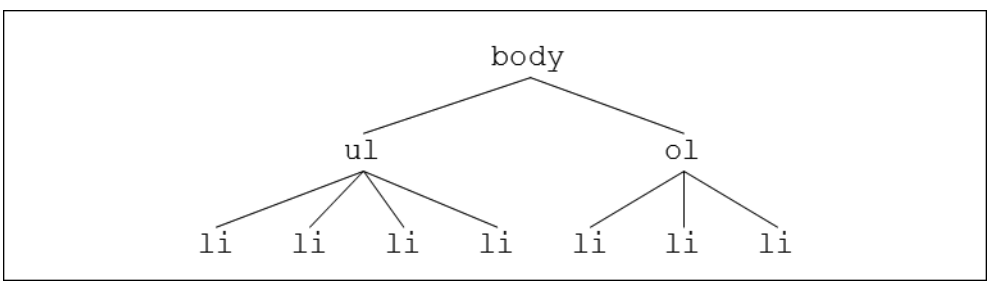

Рис. 3.4. Иерархическая структура документа

Исходно применяясь к элементу `ul`, объявление `color: gray` распространяется на все его дочерние элементы и далее — вниз по иерархической структуре — до последнего потомка. Стилевые правила никогда не наследуются снизу вверх — родительские элементы не могут заимствовать форматирование своих потомков.

В HTML существует всего одно исключение из правила наследования: стили фона могут передаваться вверх по иерархической структуре: от элемента `body` к корневому элементу (`html`), определяющему границы документа. Такая ситуация возможна только в случаях, когда фон назначен элементу `body`, а у элемента `html` он отсутствует.

Наследование — это один из фундаментальных принципов CSS, ставший настолько привычным, что воспринимается как нечто самой собой разумеющееся. Тем не менее с ним приходится считаться при стилевом оформлении всех без исключения документов.

Учтите, что наследованию подлежат далеко не все свойства, а только те из них, применение которых к вложенным элементам не приводит к получению противоречивых результатов. К таким свойствам, в частности, относится `border` (добавляющее границу к элементам). На рис. 3.5 наглядно показано, с чем связано такое ограничение. Если бы свойство `border` наследовалось, то документ выглядел бы более

загроможденным — по крайней мере, до тех пор пока задаваемое им форматирование не было бы отменено вручную.

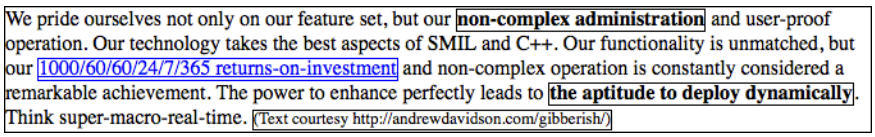

Рис. 3.5. Наследование некоторых стилей крайне нежелательно

Исходя из озвученных выше соображений, наследованию не подлежит большинство свойств форматирования блочных элементов — поля, отступы, границы и фон таких элементов не передаются их потомкам. И в самом деле, при наследовании отступа в 30 пикселей гиперссылками, добавленными в текст абзаца, он будет выглядеть совершенно нечитабельно.

Вторая особенность наследованных правил связана с их приоритетностью — для наследуемых элементов она попросту не учитывается. Данное ограничение кажется надуманным, но только не в ситуациях полного его отсутствия. Рассмотрим наглядный пример форматирования фрагмента HTML-документа двумя стилевыми правилами, результат применения которых показан на рис. 3.6.

```
* {color: gray;}
h1#page-title {color: black;}

<h1 id="page-title">Meerkat <em>Central</em></h1>
<p>
Welcome to the best place on the Web for meerkat information!
</p>
```

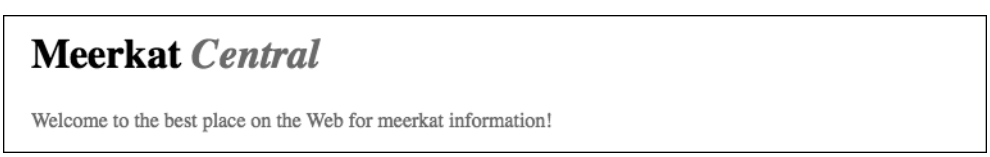

Рис. 3.6. Правила с нулевым значением приоритетности имеют преимущество перед правилами с отсутствующей приоритетностью

В данном случае к элементам применяется первое правило, а не второе, поскольку универсальный селектор добавляет к приоритетности нулевое значение, а наследуемые свойства при ее вычислении вообще не учитываются. Таким образом, элемент `em` окрашивается не в черный, а в серый цвет.

Рассмотренный пример как нельзя нагляднее иллюстрирует возможные последствия бесконтрольного использования универсального селектора в правилах. Представляя *все* элементы документа, он зачастую нарушает форматирование элементов с близкими родственными связями. Конечно, проблема решаема, но всегда проще ее предотвратить, чем устранять последствия включения универсального селектора в правила, в нем не нуждающиеся.

Зачастую отсутствие приоритетности у наследуемых правил становится причиной необычного поведения элементов. Следующее правило указывает окрашивать текст элементов с идентификатором `toolbar` в белый цвет и выводить его на черном фоне:

```
#toolbar {color: white; background: black;}
```

Такое форматирование сохраняется до тех пор, пока элементы, атрибут `id` которых имеет значение `toolbar`, содержат один только текст. При добавлении в них других элементов, например `a` (гиперссылок), к последним будет применяться собственный стиль, по умолчанию определяемый пользовательским агентом. В частности, в большинстве браузеров гиперссылки будут окрашиваться синим цветом, поскольку встроенные стили для элементов `a` в них, скорее всего, задаются следующим правилом:

```
a:link {color: blue;}
```

Чтобы исправить ситуацию, в таблице стилей нужно предусмотреть специальное правило для гиперссылок:

```
#toolbar {color: white; background: black;}
#toolbar a:link {color: white;}
```

Применяя отдельное правило к гиперссылкам, можно добиться результата, показанного на рис. 3.7.

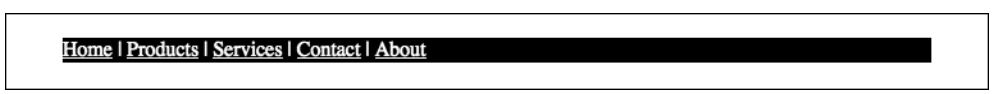

Рис. 3.7. Непосредственное стилевое форматирование вложенных элементов

Альтернативный способ применения к гиперссылкам общего цветового решения заключается в использовании значения `inherit`, описанного в главе 2.

```
#toolbar {color: white; background: black;}
#toolbar a:link {color: inherit;}
```

Результат применения этих правил такой же, как и предыдущем случае (см. рис. 3.7), поскольку теперь цвет гиперссылок наследуется у родительского элемента.

Каскадирование

Мы умышленно не рассматривали ситуации, в которых к элементу применяются сразу два правила с одинаковым значением приоритетности. Настало время разобраться, как браузер поступает в подобных случаях, предотвращая возникновение конфликтных ситуаций. Рассмотрим такой пример.

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

Какой цвет получит элемент, ведь оба правила имеют одинаковую приоритетность, а потому и шансы на применение? Разумеется, элемент не может окрашиваться сразу двумя цветами, но как узнать, какой из них будет назначен браузером?

Ответ кроется в самом названии технологии CSS (Cascading Style Sheets — каскадные таблицы стилей), в котором ключевым словом является "каскад". Под каскадом в данном случае понимается одновременное применение разных стилевых правил к элементам документа, как в результате сравнения значений приоритетности, так и с учетом принципа наследования. Каскадирование заключается в выполнении следующих действий.

1. Поиск всех правил, имеющих отношение к форматированию отдельно взятого элемента.
2. Определение *важности* каждого из правил, применяемых к элементу. Наибольшую важность имеют правила, объявления которых включают ключевое слово `!important`.
3. Определение *происхождения* стилей, влияющих на внешний вид элемента. Существуют три основных источника происхождения стилей: автора (разработчика), пользователя и браузера. В общем случае наивысший приоритет имеют стили автора. Тем не менее пользовательские стили, содержащие в объявлении ключевое слово `!important`, превалируют над стилями автора, в том числе особой важности (также объявляемые с ключевым словом `!important`). Стили браузера имеют самый низкий приоритет, уступая пальму первенства стилям автора и пользователя.
4. Определение приоритетности для объявлений стилей, применяемых к элементу. Чем больше значение приоритетности, тем бо́льшую важность имеет стиль для элемента.
5. Определение порядка объявления стилей, применяемых для форматирования элемента. Чем позже в коде объявлен стиль, тем выше его важность с точки зрения порядка следования. Правила импортируемой таблицы стилей рассматриваются пользовательским агентом перед правилами импортирующей таблицы стилей.

Чтобы понять, каким образом работает каскадирование, необходимо детально проанализировать каждое из действий, рассмотрев их на реальных примерах.

Некоторые модули CSS добавляют в объектную модель дополнительные (к трем базовым) источники происхождения стилей. Речь идет о переходах и анимации, происхождение которых не рассматривается в примерах данной главы, но детально описано в последующих главах.

Определение важности и происхождения стилей

При назначении элементу сразу нескольких стилей самым важным считается стиль, в объявлении которого имеется ключевое слово `!important`.

```
p {color: gray !important;}

<p style="color: black;">Well, <em>hello</em> there!</p>
```

Несмотря на то что цвет элемента также указывается во встроенном стиле, благодаря ключевому слову `!important` он устанавливается с помощью внешнего правила. В результате текст абзаца, а потому и вложенного в него элемента `em` (согласно принципу наследования) окрашивается серым цветом.

Для назначения форматирования с помощью встроенного стиля его также нужно снабдить ключевым словом `!important`. Таким образом, для окрашивания абзацев и всех их дочерних элементов черным цветом приведенный выше код нужно представить в следующем виде.

```
p {color: gray !important;}

<p style="color: black !important;">Well, <em>hello</em> there!</p>
```

Если стилевые правила имеют одинаковую важность, то в расчет принимается их происхождение. В общем случае стили автора (разработчика) считаются приоритетнее стилей пользователя. Рассмотрим следующие одинаковые стили с разным происхождением:

```
p em {color: black;}  /* правило из таблицы стилей автора */
p em {color: yellow;} /* правило из таблицы стилей пользователя */
```

В данном случае текст, представленный полужирным начертанием, имеет черный цвет, а не желтый, так как регулярный стиль автора приоритетнее регулярного стиля пользователя. Тем не менее при добавлении в стили ключевого слова `!important` ситуация изменяется на противоположную:

```
p em {color: black !important;}  /* правило из таблицы стилей автора */
p em {color: yellow !important;}  /* правило из таблицы стилей пользователя */
```

Теперь текст с полужирным начертанием окрашивается желтым цветом, а не черным.

Последними в документе применяются стили пользовательского агента, заданные в нем по умолчанию. Они имеют наименьшую важность и назначаются элементам, для которых не определены даже пользовательские стили. Стили пользовательского агента заменяются любыми другими в первую очередь (например, стилями автора, устанавливающими форматирование элементов `a`, — для отображения текста гиперссылок белым цветом вместо синего, заданного по умолчанию).

Подытожив рассмотренные выше принципы, можно выделить пять уровней важности стилей, применяемых в документе. Ниже они перечислены в порядке от наиболее до наименее важного.

1. Стиль пользователя с добавлением ключевого слова `!important`.
2. Стиль автора с добавлением ключевого слова `!important`.
3. Стиль автора.
4. Стиль пользователя.
5. Стиль пользовательского агента.

Разработчикам стоит принимать в расчет все перечисленные выше уровни важности, кроме последнего, так как создаваемые ими стили будут иметь однозначный приоритет только над стилями пользовательского агента.

Определение значения приоритетности

В конфликтных ситуациях решение о том, какой из стилей, имеющих одинаковую важность и происхождение, применять к элементу, принимается пользовательским агентом, исходя из значения их приоритетности.

```
p#bright {color: silver;}
p {color: black;}

<p id="bright">Well, hello there!</p>
```

Применение указанных стилей к абзацу приводит к окрашиванию текста серебристым цветом, как показано на рис. 3.8. А все потому, что значение приоритетности у селектора `p#bright` (`0,1,0,1`) больше, чем у селектора `p` (`0,0,0,1`), даже несмотря на то что в таблице стилей он указывается первым.

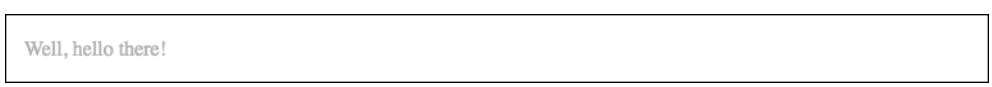

Рис. 3.8. К элементу применяется стиль с большим значением приоритетности

Определение порядка объявления стилей

На последнем этапе — при совпадении важности, происхождения и приоритетности правил, применяемых к элементу, — определяется порядок их объявления в таблице стилей. Вернемся к рассмотрению исходного примера, в котором цвет заголовка первого уровня определяется сразу двумя правилами.

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

В данном случае все элементы `h1` получают синий, а не красный цвет, поскольку при равенстве всех остальных критериев (важность, происхождение и приоритетность) предпочтение отдается правилу, добавленному в таблицу стилей последним.

А как поступает пользовательский агент при равенстве всех критериев, включая порядок указания правил в таблице стилей? Что если конфликтующие правила принадлежат к одной и той же внешней таблице стилей, как показано ниже?

```
@import url(basic.css);
h1 {color: blue;}
```

Легко заметить, что ситуация будет конфликтной только при включении правила `h1 {color: red;}` в таблицу стилей `basic.css`. В подобных случаях пользовательский агент рассматривает содержимое внешнего файла как вставленное в месте объявления (расположения команды `@import`). Исходя из этого, большей приоритетностью обладают любые правила, указанные после команды `@import`. Не стоит забывать, что речь идет о ситуациях совпадения у правил всех остальных критериев. Рассмотрим следующий пример.

```
p em {color: purple;} /* правило из внешней таблицы стилей */
p em {color: gray;}   /* правило, включенное в документ */
```

К элементу `p` применяется второе правило, как объявляемое в таблице стилей последним.

Порядок объявления правил важен при назначении стилей гиперссылкам, характеризующимся несколькими состояниями. Рекомендуется придерживаться следующего порядка стилизации состояний гиперссылок: LVFHA (Link-Visited-Focus-Hover-Active — гиперссылка, посещенная гиперссылка, в фокусе, наведен указатель, активная гиперссылка).

```
a:link {color: blue;}
a:visited {color: purple;}
a:focus {color: green;}
a:hover {color: red;}
a:active {color: orange;}
```

Как известно, у этих правил одинаковое значение приоритетности: `0,0,1,1`. Поскольку важность и происхождение у них тоже одинаковые, то к гиперссылке будет применяться последнее из правил, соответствующих текущему состоянию гиперссылки. В частности, активная гиперссылка (та, на которой щелкнули мышью или выбрали с помощью клавиатуры) описывается сразу четырьмя селекторами состояний: `:link`, `:focus`, `:hover` и `:active`. При расположении правил, задающих форматирование гиперссылок в каждом из состояний, в последовательности LVFHA активную гиперссылку будет представлять только последний селектор, как и предполагается в большинстве случаев.

Отказавшись от общепринятой формулы, правила форматирования состояний гиперссылок можно выстроить в алфавитном порядке.

```
a:active {color: orange;}
a:focus {color: green;}
a:hover {color: red;}
a:link {color: blue;}
a:visited {color: purple;}
```

Если придерживаться указанного порядка включения правил в таблицу стилей, то селекторы :hover, :focus и :active вполне очевидно останутся невостребованными, поскольку располагаются перед селекторами :link и :visited. Учитывая, что каждая гиперссылка находится либо в посещенном, либо в непосещенном состоянии, последние два правила всегда будут иметь более высокий приоритет, чем первых три.

Попробуем определить, какой еще порядок объявления правил стилевого оформления гиперссылок может заинтересовать разработчиков. Предложенный далее вариант предполагает, что наведение указателя мыши приводит к изменению внешнего вида только непосещенных гиперссылок. Посещенные гиперссылки при наведении на них указателя мыши сохраняют прежнее форматирование. Кроме того, в активное состояние переводятся как непосещенные, так и посещенные гиперссылки.

```
a:link {color: blue;}
a:hover {color: red;}
a:visited {color: purple;}
a:focus {color: green;}
a:active {color: orange;}
```

Учтите, что конфликтные ситуации возникают тогда, когда внешний вид гиперссылок во всех состояниях определяется общими свойствами. Если каждому из состояний гиперссылки соответствуют разные свойства, то порядок объявления стилевых правил не играет особой роли. Например, следующие правила можно объявлять произвольным образом, что никак не скажется на воспроизводимых ими эффектах.

```
a:link {font-weight: bold;}
a:visited {font-style: italic;}
a:focus {color: green;}
a:hover {color: red;}
a:active {background: yellow;}
```

Легко заметить, что порядок расположения селекторов :link и :visited друг относительно друга не принципиален. С точки зрения конечного результата формулы LVFHA и VLFHA абсолютно одинаковы.

Чтобы полностью избежать конфликтных ситуаций, в стилевых правилах нужно использовать селекторы псевдоклассов. Следующие объявления можно располагать в любом порядке, не беспокоясь о нарушении форматирования гиперссылок.

```
a:link {color: blue;}
a:visited {color: purple;}
a:link:hover {color: red;}
a:visited:hover {color: gray;}
```

Так как правила задают форматирование для уникальных состояний гиперссылок, между собой они не конфликтуют. Таким образом, изменение порядка их расположения в таблице стилей не приводит к нарушениям в оформлении документов. Несмотря на то что последние два правила имеют одинаковую приоритетность, между собой они не пересекаются. И это понятно, поскольку гиперссылка не может одновременно находиться и в посещенном, и в непосещенном состоянии, а потому

форматирование в каждом из случаев задается отдельно. Если же селекторами псевдоклассов снабдить правила, определяющие форматирование активных состояний, то порядок их следования снова становится очень важным.

```
a:link {color: blue;}
a:visited {color: purple;}
a:link:hover {color: red;}
a:visited:hover {color: gray;}
a:link:active {color: orange;}
a:visited:active {color: silver;}
```

Правила для активных состояний, располагаемые перед правилами форматирования гиперссылок, на которые наведен указатель мыши, пользовательским агентом не рассматриваются. Причина конфликта — совпадение значений приоритетности у стилей. Чтобы предотвратить возникновение конфликтной ситуации, в селекторы игнорируемых правил нужно добавить уточняющие псевдоклассы.

```
a:link:hover:active {color: orange;}
a:visited:hover:active {color: silver;}
```

Приоритетность обоих селекторов увеличивается до значения `0,0,3,1` — между собой они не конфликтуют, поскольку описывают взаимоисключающие состояния. Одновременно гиперссылка может представляться только одним из двух селекторов, а ее внешний вид — задаваться соответствующим правилом.

Нестилевое оформление документа

Оформление документа может выполняться без использования инструментов CSS — например, с помощью элемента `font`. Пользовательским агентом такое форматирование рассматривается как имеющее нулевую приоритетность, уступая пальму первенства стилям автора и пользователя, но только не стилям пользовательского агента. В CSS3 нестилевое форматирование вообще приравнивается к оформлению документа стилями пользовательского агента (предположительно добавленное в конец таблицы стилей, но в спецификации это в явном виде не оговаривается).

Резюме

Каскадирование является краеугольным камнем CSS и устанавливает принципы стилевого форматирования документа в любых конфликтных ситуациях. Оно основывается на сопоставлении значений приоритетности селекторов и механизме наследования стилевых правил.