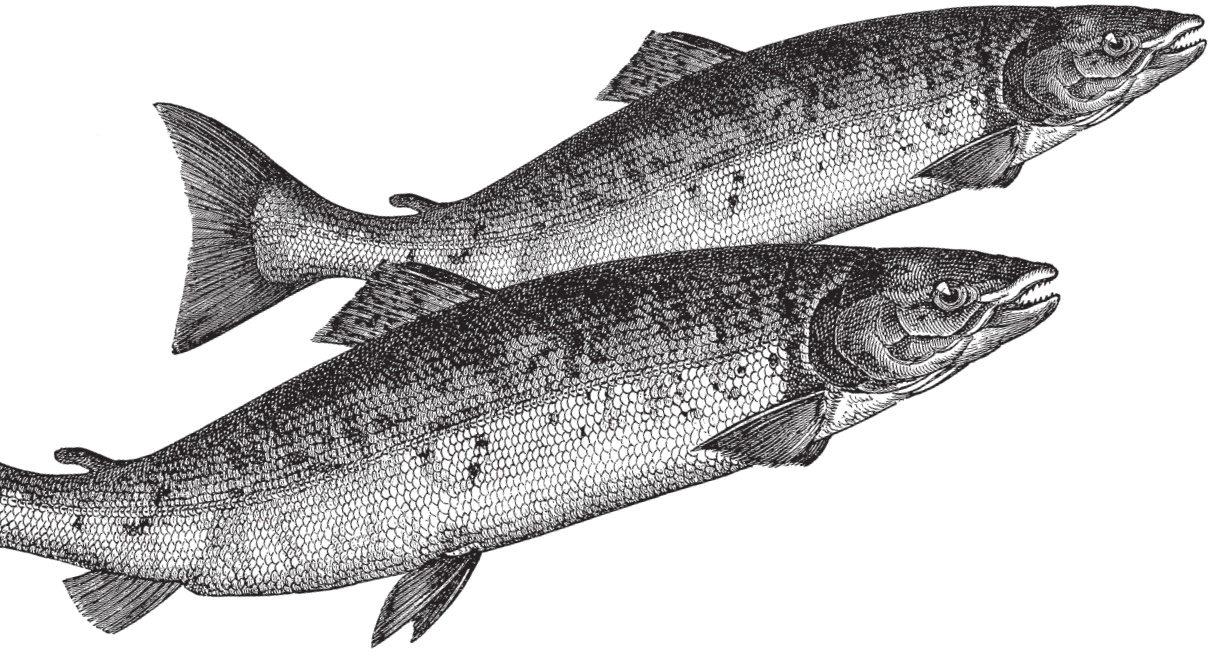
Визуальное представление веб-документов

3-е издание

CSS

каскадные таблицы стилей

Подробное руководство



Символ®

O'REILLY®

Эрик А. Мейер

Cascading Style Sheets

The Definitive Guide

Third Edition

Eric A. Meyer

O'REILLY®

CSS

каскадные таблицы стилей

Подробное руководство

Третье издание

Эрик Мейер

Символ®

Санкт-Петербург — Москва
2008

Эрик Мейер

CSS – каскадные таблицы стилей. Подробное руководство, 3-е издание

Перевод Н. Шатохиной

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*Б. Попов*
Редактор	*Ю. Бочина*
Корректор	*Л. Минина*
Верстка	*Д. Орлова*

Мейер Э.
CSS – каскадные таблицы стилей. Подробное руководство, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2008. – 576 с., ил.

ISBN-13: 978-5-93286-107-3
ISBN-10: 5-93286-107-X

Третье издание «CSS – каскадные таблицы стилей. Подробное руководство» показывает, как реализовать на практике все возможности каскадных таблиц стилей для стандартов CSS2 и CSS2.1. Множество примеров позволит научиться быстро и без усилий разрабатывать стилевое оформление веб-страниц, отвечающее современным требованиям.

Эрик Мейер, признанный эксперт по CSS, HTML и веб-стандартам, опираясь на свой богатейший опыт, рассматривает все свойства CSS и их взаимодействие, теги, атрибуты, реализации, поддержку различными броузерами, дает рекомендации разработчикам. Вы узнаете о сложном стилевом оформлении документов, пользовательском интерфейсе, верстке таблиц, о списках и генерируемом содержимом, о свободном перемещении и позиционировании, о семействах шрифтов и механизмах резервирования, о том, как работает модель блоков, о новых селекторах CSS3, поддерживаемых IE7, Firefox и другими броузерами. Книга поможет избежать распространенных ошибок, она является полным справочником по CSS и будет полезна как опытному веб-разработчику, так и новичку. От читателя потребуется только знание HTML 4.0.

ISBN-13: 978-5-93286-107-3
ISBN-10: 5-93286-107-X
ISBN 0-596-52733-0 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 29.02.2008. Формат 70×100¹/16. Печать офсетная.
Объем 36 печ. л. Тираж 2000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Посвящаю жене и дочери за все то счастье,
которое они дарят мне.

Оглавление

Введение

Если вы веб-дизайнер, разработчик веб-страниц и вас интересуют сложные стили оформления страниц, улучшение их восприятия и экономия времени и усилий, эта книга – для вас. Все, что надо для начала, – это неплохое знание HTML 4.0. Чем лучше вы знаете HTML, тем лучше вы подготовлены к чтению книги. Что касается остальных знаний и умений, для работы с этой книгой достаточно базового уровня.

Третье издание книги «CSS – каскадные таблицы стилей. Подробное руководство» посвящено стандартам CSS2 и CSS2.1 (вплоть до рабочего проекта, вышедшего 11 апреля 2006 года), последний из которых во многом представляет собой дополненную версию первого. Несмотря на то что некоторые части CSS3 получили статус предварительной рекомендации, в этом издании я решил их не рассматривать (за исключением некоторых селекторов CSS3), потому что реализация соответствующих модулей до сих пор не завершена или ее попросту нет. Я считаю, что сейчас важно сосредоточиться на поддерживаемых в настоящий момент и хорошо понятных уровнях CSS, а все грядущие возможности лучше оставить для последующих изданий.

Принятые обозначения

В этой книге действуют следующие соглашения о шрифтовом оформлении:

Курсив

Применяется для выделения новых терминов, URL, переменных в тексте, имен файлов и каталогов, команд, расширений файлов и путей доступа UNC.

`Моноширинный шрифт`

Предназначен для данных, выводимых в окне командной строки, примеров кода, ключей реестра.

`Моноширинный жирный`

Обозначает в примерах данные, вводимые пользователем.

`Моноширинный курсив`

Показывает переменные в примерах и ключах реестра. Также используется для выделения переменных или определяемых пользо-

вателем элементов в тексте, написанном курсивом (например, в пути или имени файла). Например, в пути *\Windows\имя_пользователя* замените текст имя_пользователя на реальное имя зарегистрированного в системе пользователя.

Так отмечаются подсказки, советы и примечания.

А это предупреждение и предостережение.

Соглашения по представлению свойств

В этой книге встречаются блоки разбора рассматриваемых свойств CSS. Они практически дословно воспроизведены из спецификаций CSS, поэтому необходимы некоторые разъяснения их синтаксиса.

По всей книге допустимые значения каждого свойства приводятся в соответствии со следующим синтаксисом:

Значение: **[** <длина> | thick | thin **]{1,4}**

Значение: **[**<имя_семейства> **,]*** <имя_семейства>

Значение: <url>**?** <цвет> **[** / <цвет> **]?**

Значение: <url> || <цвет>

Любые слова между символами «<» и «>» определяют тип значения или ссылку на другое свойство. Например, свойство font может принимать значения, которые на самом деле относятся к свойству font-family. На это указывает выражение <font-family>. Любые слова, представленные моноширинным шрифтом, представляют собой ключевые слова, которые должны применяться буквально, без кавычек. Прямая наклонная черта (/) и запятая (,) также должны использоваться буквально.

Выстраивание нескольких ключевых слов в некоторой последовательности также означает, что все они должны выполняться в данном порядке. Например, help me означает, что в свойстве эти ключевые слова должны быть расположены именно в таком порядке.

Если варианты разделены вертикальной чертой (X | Y), то следует выбрать какой-то один из них. Двойная вертикальная черта (X || Y) означает, что можно выбрать X, Y или оба элемента, и появляться они могут в любом порядке. Скобки [...] предназначены для группировки. Размещение рядом имеет больший приоритет, чем двойная вертикальная черта, которая в свою очередь имеет больший приоритет, чем одиночная вертикальная черта. Таким образом, запись «V W | X || Y Z» эквивалентна «[V W] | [X || [Y Z]]».

За каждым словом или заключенной в скобки группой может располагаться один из следующих модификаторов:

- Звездочка (*) указывает на то, что предшествующее ей значение или заключенная в скобки группа повторяются нуль или более раз. То есть запись `bucket*` означает, что слово `bucket` может повторяться любое количество раз, а может вообще отсутствовать. Верхней границы для возможного количества его применений нет.
- Плюс (+) указывает на то, что предшествующее значение или заключенная в скобки группа повторяются один или более раз, то есть запись `mop+` означает, что слово `mop` должно быть использовано хотя бы единожды или, возможно, несколько раз.
- Знак вопроса (?) указывает на то, что предшествующее значение или заключенная в скобки группа являются необязательными. Например, `[pine tree]?` означает, что слова `pine tree` могут отсутствовать (хотя в случае употребления они должны появляться строго в указанном порядке).
- Пара чисел в фигурных скобках {M,N} указывает на то, что предшествующее значение или заключенная в скобки группа повторяются не менее `M` и не более `N` раз. Например, `ha{1,3}` означает, что слово `ha` можно повторить один, два или три раза.

Вот некоторые примеры:

`give || me || liberty`

Должно присутствовать хотя бы одно из этих трех слов, и они могут располагаться в любом порядке. Например, `give liberty`, `give me`, `liberty me give` и `give me liberty` – все это действительные варианты.

`[ I | am ]? the || walrus`

Может присутствовать любое из слов `I` и `am`, но не оба сразу. Кроме того, можно вообще обойтись без этих слов. Также должны присутствовать слова `the` или `walrus` или оба в произвольном порядке. Таким образом, вы могли бы составить фразы `I the walrus`, `am walrus the`, `am the`, `I walrus`, `walrus the` и т. д.

`koo+ ka-choo`

Один или более экземпляров слова `koo` должны быть продолжены словом `ka-choo`. Следовательно, выражения `koo koo ka-choo`, `koo koo koo ka-choo` и `koo ka-choo` являются допустимыми. Несмотря на существующие ограничения, определяемые конкретной реализацией, число экземпляров `koo` стандартом никак не ограничено.

`I really{1,4}? [love | hate] [Microsoft | Netscape | Opera | Safari]`

Это универсальное средство выражения мнений веб-разработчика. Этот пример можно интерпретировать как `I love Netscape`, `I really love Microsoft` и аналогичные выражения. Слово `really` может отсутствовать или применяться от одного до четырех раз. Также можно

выбирать между `love` и `hate`, хотя здесь показан только пример со словом `love`.

`[[Alpha || Baker || Cray], ]{2,3} and Delphi`

Это более длинное и сложное выражение. Одним из возможных результатов может быть `Alpha, Cray, and Delphi`. Запятая здесь вставляется потому, что она указана в ограниченной скобками вложенной группе.

Использование примеров кода

Данная книга призвана помочь вам в вашей работе. В общем, вы можете использовать код из этой книги в своих программах и документации. Не надо обращаться в O'Reilly за разрешением на использование небольших частей кода, например при написании программы, в которой используется несколько блоков кода из этой книги. А вот продажа или распространение CD-ROM с примерами из книг O'Reilly требует специального разрешения. Вы можете свободно ссылаться на книгу и цитировать примеры кода, но для включения больших частей кода из этой книги в документацию вашего продукта требуется наше согласие.

Будем признательны, но не настаиваем на указании авторства. Обычно ссылка на источник включает название, автора, издателя и ISBN. Например: «CSS: Подробное руководство, Эрик А. Мейер. Copyright 2007 O'Reilly Media, Inc., 978-0-596-52733-4».

Если вам кажется, что использование вами примеров кода выходит за рамки законного использования или разрешений, оговоренных выше, не стесняйтесь, обращайтесь к нам по адресу *permissions@oreilly.com*.

Контактная информация

Сотрудники издательства O'Reilly тщательно проверили корректность информации, приведенной в данной книге, но не исключено, что некоторые возможности изменились (или даже остались ошибки!). Пожалуйста, сообщайте нам о любых найденных неточностях, а также присылайте ваши предложения для будущих изданий по адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США или Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

Для этой книги создана веб-страница, на которой представлены список опечаток, примеры и другая дополнительная информация. Страница расположена по адресу:

http://www.oreilly.com/catalog/csstdg3

Вопросы и комментарии по этой книге присылайте по электронной почте:

bookquestions@oreilly.com

Для получения более подробной информации о книгах, конференциях, ресурсах и сети O'Reilly посетите веб-сайт издательства:

http://www.oreilly.com

Safari® Enabled

Если на обложке книги есть пиктограмма «Safari® Enabled», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи лучших технических книг, копировать и использовать примеры кода, скачивать главы и быстро находить ответы, когда требуется самая точная и свежая информация. Она свободно доступна по адресу *http://safari. oreilly.com*.

Благодарности

Я бы хотел воспользоваться случаем и выразить благодарность всем, кто поддерживал меня во время долгого путешествия этой книги до полок магазинов.

Прежде всего хотелось бы поблагодарить всех сотрудников O'Reilly за все, что они делали в течение этих лет, периодически предоставляя мне возможность отдыхать, чтобы я мог написать достойную книгу. В этом третьем издании я выражаю благодарность Татьяне Апанди (Tatiana Apandi) за ее хорошее чувство юмора, терпение и понимание в случаях, когда я запаздывал со сроками.

Я также хотел бы выразить глубокую признательность моим техническим рецензентам. В первом издании это были Дэвид Бэрон (David Baron) и Ян Хиксон (Ian Hickson) с участием Берта Боса (Bert Bos) и Хекон Ли (Hekon Lie). Вторым изданием занимались Тантек Келик (Tantek Celik) и Ян Хиксон (Ian Hickson). Третье издание, которое вы держите в руках, редактировали замечательные люди – Даррелл Остин (Darrell Austin), Лайза Дейли (Liza Daly) и Нейл Ли (Neil Lee). Все они вложили немалый опыт и понимание, благодаря чему я смог учесть самые последние изменения, внесенные в CSS, а также избежать небрежных описаний и туманных пояснений. Ни одно из изданий этой книги, тем более данное, не получилось бы таким качественным, если бы не их коллективный вклад, но любые ошибки, которые вы найдете в тексте, – это, безусловно, моя вина, а не их. Это избитая фраза, но так оно и есть.

Также я хотел бы сказать спасибо всем, кто нашел опечатки и сообщил о них. Возможно, я не всегда оперативно отвечал на ваши письма, но читал все вопросы и замечания и в случае необходимости вносил коррективы. Продолжайте присылать свои отзывы, обратная связь и конструктивная критика лишь помогут сделать книгу лучше, как это было всегда.

Я также хочу выразить несколько личных благодарностей.

Коллективу WRUW, 91.1 FM Cleveland, спасибо вам за девять лет поддержки, замечательную музыку и бесшабашный юмор. Может быть, когда-нибудь я верну в ваш эфир звуки биг-бенда, а может, и нет, но в любом случае оставайтесь с нами.

Джеффри Зельдману (Jeffrey Zeldman) спасибо за то, что он замечательный коллега и партнер. И всему семейству Зельдманов спасибо за дружбу.

«Тетушке» Молли спасибо за то, что она всегда остается самой собой.

«Дядю» Джима благодарю за все – и в профессиональном, и в личном плане. Не будет преувеличением сказать, что я не достиг бы того, чего достиг, без вашего влияния, да и жизнь была бы намного более пресной, не будь вас рядом.

Всей команде кулинаров – Джиму, Женевьев, Джиму, Джини, Ферретт, Джен, Дженн и Молли – спасибо за великолепную стряпню и приятную беседу.

Всем, кого я должен был бы поблагодарить, но не сделал этого, мои извинения. И благодарности.

Моим жене и дочери безграничная благодарность за то, что они делают мои дни богаче, чем я заслуживаю, и за то, что они окружают меня такой любовью, которую я даже не надеюсь им вернуть. Хотя, конечно, я буду стараться.

– Эрик А. Мейер (Eric A. Meyer)
г. Кливленд-Хайтс, Огайо
1 августа 2006

1

CSS и документы

Каскадные таблицы стилей (CSS – Cascading Style Sheets) – мощный механизм управления представлением отдельных документов или их наборов. Очевидно, собственно каскадные таблицы стилей при отсутствии документа бесполезны, поскольку в них нет содержимого, которое надо представлять. Конечно, термин «документ» понимается здесь крайне широко. Например, Mozilla и родственные броузеры используют CSS, чтобы воздействовать на представление деталей интерфейса самого броузера. Но и в этом случае без «декораций» – кнопок, полей ввода адреса, диалоговых и обычных окон и т. д. – нет необходимости в CSS (или любой другой информации о представлении).

Веб спускается с Олимпа

В смутно припоминаемые (1990–1993) ранние годы Всемирной паутины HTML был довольно бедным языком. Он почти целиком состоял из структурных элементов, полезных для описания абзацев, гиперссылок, списков и заголовков. В нем не было ничего, даже отдаленно напоминавшего таблицы, фреймы или сложную разметку, – того, что считается абсолютно необходимым для создания веб-страниц. HTML изначально задумывался как структурный язык разметки, применяемый для описания различных частей документа. О том, как должны отображаться эти части, говорилось совсем немного. Язык не затрагивал описание внешнего вида – он был лишь небольшой схемой разметки.

Затем пришел Mosaic.

Мощь Всемирной паутины вдруг стала очевидной практически каждому, кто проводил в ней более 10 минут. Для перехода от одного документа к другому было достаточно указать курсором на выделенный специальным цветом фрагмент текста или даже изображение и щелкнуть кнопкой мыши. Кроме того, текст и графику можно было отобра-

жать на экране вместе, а для создания страницы нужен был только простой текстовый редактор. Все это находилось в свободном доступе, было открыто, и это здорово.

Веб-сайты начали возникать повсеместно. Это были личные дневники, сайты университетов, корпоративные сайты и т. д. Поскольку росло количество сайтов, росла и потребность в новых HTML-элементах, которые должны были выполнять определенные функции. Авторам потребовались средства, которые позволяли выделять фрагменты текста полужирным шрифтом или курсивом.

На тот момент HTML не имел возможности реализовать эти пожелания. Вы могли указать необходимость выделения фрагмента текста, но это не обязательно означало, что текст выделялся курсивом; вместо этого он мог быть выделен полужирным шрифтом или даже обычным шрифтом другого цвета в зависимости от броузера и его установок. Не было средств, которые бы гарантировали, что пользователь увидит то, что создал автор.

В результате этого давления в язык начали просачиваться такие элементы разметки, как теги `<FONT>` и `<BIG>`. Так структурный язык начал превращаться в язык представления.

Полная неразбериха

Прошли годы, и мы унаследовали все извивы этого беспорядочного процесса. Значительная часть средств HTML 3.2 и HTML 4.0 оказалась посвящена вопросам представления. Возможность изменять цвет и размер текста с помощью элемента `font`, назначать фоновые цвета и изображения для документов и таблиц, задавать атрибуты тега `table` (такие как `cellspacing`) и делать текст мигающим – все это наследие произнесенных в то время просьб «побольше управления!».

Чтобы увидеть результаты возникшей неразберихи, взгляните на разметку практически любого корпоративного веб-сайта. Поразительно, насколько объем разметки превышает объем полезной информации. Что еще хуже, разметка большинства сайтов практически полностью состоит из таблиц и элементов `font`, ни один из которых не помогает понять, что именно он представляет. С точки зрения структурированности эти страницы ненамного лучше, чем случайный набор строк или символов.

Давайте, например, предположим, что для заголовков страницы автор вместо тегов заголовка, таких как `h1`, применяет элементы `font`:

```
<font size="+3" face="Helvetica" color="red">Заголовок страницы</font>
```

Тег `font` никак не влияет на структуру документа. Это делает документ гораздо менее полезным. Какой, например, толк в теге `font` для броузера с возможностью синтеза речи? Если автор применяет заголовки, а не элементы `font`, броузер при воспроизведении текста может изменить

громкость или интонацию. Тег `font` не дает броузеру возможности узнать, что данный фрагмент чем-то отличается от остального текста.

Почему же авторы применяют для управления структурой и содержанием такие жесткие методы? Потому что они хотят, чтобы читатели видели страницу такой, какой она была создана. Использовать структурированную разметку HTML для них означает передать броузеру контроль за внешним видом страницы, что кажется недопустимым для плотно укомплектованных страниц, которые обрели популярность за эти годы. Но давайте взглянем на проблемы, возникающие при таком подходе:

- Неструктурированные страницы значительно усложняют индексирование содержимого. Мощный поисковый механизм позволяет вести поиск только по заголовкам страниц, или только по заголовкам разделов страницы, или только по тексту абзацев, или, возможно, только по специально отмеченным абзацам. Однако для реализации таких возможностей содержимое страниц должно соответствовать некому стандарту структурированной разметки. Именно этого не хватает большинству страниц. Обратите внимание, что Google при индексации страниц учитывает структуру разметки, поэтому структурированная страница повысит ваш ранг в Google.

- Недостаточная структурированность снижает удобство восприятия. Представьте, что вы слепы и при поиске полагаетесь на броузер с модулем синтеза речи. Что вы предпочтете: структурированную страницу, которая позволит вашему броузеру читать только заголовки разделов, и вы сможете выбрать раздел, о котором хотите услышать подробнее, или страницу, в которой структура отсутствует настолько, что ваш броузер вынужден читать весь текст подряд, не понимая, является ли он заголовком, текстом абзаца или помечен как важный? Давайте снова вспомним о Google – поисковой системе, которая сегодня является наиболее активным скрытым пользователем, имеющим миллионы друзей, которые принимают каждое из его предложений типа «где искать и что купить».

- Улучшенное представление страницы возможно только при наличии некоторой структуры документа. Представьте себе страницу, на которой показаны только заголовки разделов со стрелкой рядом с каждым из них. Пользователь сам выбирает, какой заголовок его интересует, и, щелкнув по нему, получает доступ к тексту раздела.

- Структурированная разметка упрощает обслуживание сайта. Сколько раз вы охотились за незначительными ошибками в чужом (или даже в собственном) HTML, которые портили вид страницы в том или ином броузере? Сколько времени вы потратили, создавая вложенные таблицы или элементы `font`, чтобы просто получить врезку с белыми ссылками? Сколько переходов строки вы вставили, пытаясь добиться необходимого расстояния между заголовком и текстом?

Применение структурированной разметки позволяет очистить код и упростить поиск в нем.

Тем не менее приходится признать, что полностью структурированный документ несколько некрасив. Из-за одного этого факта и сотни аргументов в пользу структурированной разметки не хватит, чтобы отговорить отдел маркетинга от использования того типа HTML, который был распространен в конце XX столетия и существует до сих пор. Поэтому нужен метод, позволяющий сочетать структурированную разметку с привлекательным представлением страницы.

CSS спешит на помощь

Конечно, проблема загрязнения HTML разметкой представления не осталась незамеченной консорциумом W3C (World Wide Web Consortium), который приступил к поиску быстрого решения. В 1995 году консорциум начал публикацию рабочего варианта стандарта, названного CSS. К 1996 году он получил статус рекомендации, такой же значимой, как и сам HTML. Вот причины этого.

Богатство стилей

Прежде всего, CSS обеспечивают более богатое представление документа, чем когда-либо, даже на пике своего репрезентативного пыла, позволял HTML. CSS позволяют задавать цвета текста и фона любых элементов, создавать рамки и увеличивать или уменьшать отступы вокруг элементов. Благодаря им можно сделать так, чтобы текст отображался прописными буквами, и добавить дополнительные элементы оформления (например, подчеркивание), разбивки и даже управлять тем, будет ли он отображаться вообще, а также они дают возможность реализовать многие другие эффекты.

Возьмем для примера первый (и основной) заголовок на странице, который обычно является заголовком самой страницы. Пример правильной разметки:

```
<h1>Паря над водой</h1>
```

Теперь предположим, что вы хотите, чтобы этот заголовок стал темно-красным, был набран определенным шрифтом курсивного начертания, подчеркнут и располагался на желтом фоне. Чтобы сделать все это с помощью HTML, понадобилось бы поместить тег `h1` в таблицу и нагрузить его массой других элементов, таких как `font` и `U`. CSS позволяет обойтись одним правилом:

```
h1 {color: maroon; font: italic 2em Times, serif; text-decoration: underline;
      background: yellow;}
```

Вот и все. Как видите, все, что вы делали в HTML, можно сделать в CSS. Однако не надо ограничивать себя только тем, что умеет HTML:

```
h1 {color: maroon; font: italic 2em Times, serif; text-decoration: underline;
      background: yellow url(titlebg.png) repeat-x;
      border: 1px solid red; margin-bottom: 0; padding: 5px;}
```

Теперь в качестве фона элемента h1 выступает изображение, которое повторяется в горизонтальном направлении, а вокруг элемента добавлена рамка, отстоящая от текста минимум на пять пикселов. Вы также удалили поле (пустое пространство) в нижней части элемента. Это те возможности, к которым HTML даже и близко не мог подобраться, и это всего лишь малая часть того, что могут CSS.

Простота применения

Если сила CSS не убедила вас, примите во внимание следующее: таблицы стилей могут существенно сократить объем работы разработчика веб-страниц.

Во-первых, таблицы стилей концентрируют команды, реализующие визуальные эффекты, в одном легкодоступном месте, а не разбрасывают их по всему документу. В качестве примера предположим, что в документе все заголовки h2 должны быть фиолетовыми. Работая с HTML, можно поместить тег font в каждый заголовок:

```
<h2><font color="purple">Это фиолетовый!</font></h2>
```

Это должно быть сделано для каждого заголовка второго уровня. Если в документе 40 заголовков, то надо вставить 40 элементов font, по одному для каждого заголовка! Слишком много труда требует один небольшой эффект.

Предположим, что вы вставили все эти элементы font. Вы закончили, вы счастливы, а затем решаете (или начальник решает за вас), что заголовки h2 должны быть темно-зелеными, а не фиолетовыми. Надо возвращаться назад и вновь заменять каждый из элементов font. Конечно, можно обратиться к командам поиска и замены, поскольку заголовки являются единственным текстом фиолетового цвета в вашем документе. Но если вы ввели и другие элементы font фиолетового цвета, вы *не можете* применить автоматический поиск и замену, потому что тогда будут изменены и эти элементы.

Вместо этого намного лучше было бы иметь одно правило:

```
h2 {color: purple;}
```

Его не только намного быстрее набрать на клавиатуре, но и намного проще изменять. Для перехода от фиолетового цвета к темно-зеленому достаточно изменить одно-единственное правило.

Вернемся к нагруженному стилями элементу h1 из предыдущего раздела:

```
h1 {color: maroon; font: italic 2em Times, serif; text-decoration: underline;
      background: yellow;}
```

Может показаться, что написать это правило намного сложнее, чем HTML, но представьте, что на странице расположена дюжина элементов h2, которые должны выглядеть так же, как h1. Сколько разметки потребуется для этих 12 элементов h2? Много. А применяя CSS, достаточно сделать следующее:

```
h1, h2 {color: maroon; font: italic 2em Times, serif;
          text-decoration: underline; background: yellow;}
```

Теперь стили применены и к элементам h1, и к элементам h2, а понадобилось для этого всего лишь три дополнительных нажатия на клавиши.

Если вы хотите изменить вид элементов h1 и h2, преимущества CSS даже еще более ощутимы. Представьте, сколько времени понадобилось бы, чтобы изменить разметку HTML для элементов h1 и h2, по сравнению с внесением следующих изменений в предыдущие стили:

```
h1, h2 {color: navy; font: bold 2em Helvetica, sans-serif;
          text-decoration: underline overline; background: silver;}
```

Если бы продолжительность каждого из этих двух подходов фиксировалась с помощью секундомера, держу пари, что разбирающийся в CSS автор без труда обогнал бы приверженца HTML.

Кроме того, большинство правил CSS сосредоточиваются в документе в одном месте. Можно распределить их по всему документу, группируя в ассоциированные стили или отдельные элементы, но как правило, намного эффективнее поместить все стили в одну таблицу стилей. Это позволяет разрабатывать (или изменять) внешний вид всего документа в одном месте.

Применение стилей к нескольким страницам

Но есть и еще кое-что! Можно не только централизовать всю информацию о стилях страницы в одном месте, но и создать таблицу стилей, которая может применяться ко многим страницам. Это реализуется путем сохранения таблицы стилей в отдельном документе, который затем импортируется любой использующей его страницей документа. Эта возможность позволит быстро создавать единообразный внешний вид всего веб-сайта. Для этого достаточно привязать одну таблицу стилей ко всем документам веб-сайта. Затем, если вам когда-нибудь захочется изменить внешний вид страниц сайта, надо будет лишь отредактировать один файл, и внесенные изменения распространятся на весь сервер автоматически!

Рассмотрим сайт, в котором все заголовки выделены серым шрифтом на белом фоне. Этот цвет они получают из таблицы стилей, которая гласит:

```
h1, h2, h3, h4, h5, h6 {color: gray; background: white;}
```

Предположим, этот сайт состоит из 700 страниц, каждая из которых использует таблицу стилей, оговаривающую, что заголовки должны

быть серыми. В некоторый момент разработчик веб-сайта принимает решение, что заголовки должны быть белыми на сером фоне. Итак, он редактирует таблицу стилей следующим образом:

```
h1, h2, h3, h4, h5, h6 {color: white; background: gray;}
```

затем таблица стилей сохраняется на диск и изменение выполнено. Это, конечно же, лучше, чем редактировать 700 страниц, чтобы просмотреть и изменить каждый заголовок таблицы и тег `font`, не так ли?

Каскадирование

И это еще не все! CSS также поддерживает средства разрешения конфликтов правил, называемые *каскадным включением* (*cascade*). Возьмем для примера предыдущий сценарий, в котором одна таблица стилей импортировалась в несколько веб-страниц. Теперь добавим набор страниц, которые ряд стилей используют совместно, но в то же время включают специализированные правила, применяемые только к конкретным страницам. В дополнение к уже существующей таблице стилей можно создать еще одну таблицу, импортируемую в эти страницы, или просто поместить специальные стили в страницы, которые в них нуждаются.

Например, требуется, чтобы на одной из 700 страниц заголовки были выделены желтым цветом на темно-синем фоне вместо желтого на сером. Тогда в этот отдельный документ можно ввести такое правило:

```
h1, h2, h3, h4, h5, h6 {color: yellow; background: blue;}
```

Благодаря каскадному включению это правило переопределит импортированное правило, реализующее желтые заголовки на сером фоне. Понимая и разумно применяя правила каскадирования, можно создавать сложные таблицы стилей, без труда изменяемые и объединяемые для обеспечения профессионального представления страниц.

От мощи каскадного объединения выигрывает не только автор. Вебсерферы (или *читатели*) могут в некоторых броузерах создавать собственные таблицы стилей (названные *таблицами стилей читателя* (*reader style sheets*)), которые будут каскадироваться со стилями автора, а также со стилями, используемыми броузером. Благодаря этому читатель-дальтоник может создать стиль, который выделяет гиперссылки:

```
a:link, a:visited {color: white; background: black;}
```

Таблица стилей читателя может включать все, что угодно: директиву увеличения размера текста для пользователя с ослабленным зрением, правила для удаления изображений, чтобы увеличить скорость чтения и просмотра, и даже стили для размещения любимой картинки пользователя в качестве фона каждого документа. (Это, конечно, не рекомендуется, но возможно.) Благодаря этому пользователи могут настраивать представление веб-документа, не отключая все стили автора.

CSS – замечательный инструмент для любого автора или читателя, поскольку обладает возможностями импортирования, каскадирования и реализации разнообразных эффектов.

Небольшой размер файла

Кроме визуальной мощи CSS и его способности расширять возможности как автора, так и читателя, в нем есть еще кое-что, что понравится вашим читателям. CSS способствует уменьшению размеров документов, сокращая таким образом время загрузки. Как? Я уже говорил, что для реализации визуальных эффектов многие страницы использовали таблицы и элементы `font`. К сожалению, оба эти метода создают дополнительную HTML-разметку, увеличивающую размеры файлов. Группируя информацию о визуальных стилях в одном месте и представляя эти правила посредством компактного синтаксиса, можно удалить элементы `font` и другие фрагменты обычной мешанины тегов. Таким образом, CSS уменьшит время загрузки к вящему удовольствию читателей.

Подготовка к будущему

Как я уже упоминал, HTML представляет собою структурный язык, тогда как CSS – это дополнение, стилистический язык. Осознавая это, W3C – орган, обсуждающий и утверждающий стандарты для Всемирной паутины, – начинает удалять стилистические элементы из HTML. Для создания эффектов, сейчас обеспечиваемых определенными HTML-элементами, могут применяться таблицы стилей. Так зачем тогда нужны эти HTML-элементы?

Таким образом, спецификация XHTML содержит ряд элементов, применять которые не рекомендуется (*deprecated*), поскольку со временем они будут полностью удалены из языка. В конце концов, они будут отмечены как устаревшие (*obsolete*), то есть от броузеров не будет требоваться и не будет поощряться их поддержка. К нерекомендуемым к применению элементам относятся: `<font>`, `<basefont>`, `<u>`, `<strike>`, `<s>` и `<center>`. С появлением таблиц стилей необходимость во всех этих элементах отпала. А со временем, возможно, количество нерекомендуемых элементов увеличится.

Мало того, существует вероятность того, что постепенно HTML будет вытеснен *расширяемым языком разметки* (eXtensible Markup Language – XML). XML намного сложнее HTML, но при этом он обладает значительно большими возможностями и гибкостью. Несмотря на это XML не предоставляет способа объявить стилевые элементы, такие как `<i>` или `<center>`. Вместо них, скорее всего, для определения внешнего вида XML-документов будут применяться таблицы стилей. Хотя таблицы стилей, применяемые с XML, могут и не быть таблицами CSS, по всей вероятности, они будут производными от CSS или чем-то очень близким к ним. Поэтому изучение CSS сейчас обеспечит авторам

большое преимущество, когда придет время сделать переход к Всемирной паутине на базе XML.

Итак, для начала очень важно понимать, как связаны друг с другом CSS и структура документа. CSS позволяют полностью изменить облик документа, но их возможности не безграничны. Начнем с изучения базовой терминологии.

Элементы

Элементы (*elements*) – это основа структуры документа. Нетрудно понять, какие элементы более всего используются в HTML: `p`, `table`, `span`, `a` и `div`. Каждый элемент документа играет определенную роль в его представлении. В терминах CSS (по крайней мере, для CSS2.1) это означает, что каждый элемент генерирует блок, в котором находится содержимое элемента.

Замещаемые и незамещаемые элементы

CSS определяется элементами, но не все элементы создаются одинаково. Например, изображения и абзацы – это элементы разных типов, так же как `span` и `div`. В CSS элементы разделяются на замещаемые и незамещаемые. Эти два типа элементов подробно обсуждаются в главе 7, где структуры блоков рассматриваются детально, а здесь я лишь вскользь коснусь их.

Замещаемые элементы

Замещаемыми (*replaced*) называются те элементы, содержимое которых замещается чем-то, что не содержится непосредственно в документе. Наиболее очевидный пример из XHTML – элемент `img`, замещаемый файлом изображения, который является внешним по отношению к документу. Кстати, `img` фактически не имеет содержимого, как видно из простого примера:

```
<img src="howdy.gif" />
```

Этот фрагмент разметки не имеет реального содержимого, а только имя элемента и атрибут. Данный элемент ничего не представляет, пока вы не укажете на внешнее содержимое (в данном случае изображение, заданное атрибутом `src`). Элемент `input` замещается кнопкой, переключателем или полем ввода текста в зависимости от его типа. Замещаемые элементы также генерируют блоки в своем визуальном представлении.

Незамещаемые элементы

Основная масса элементов HTML и XHTML – *незамещаемые* (*nonreplaceable*). Это означает, что их отображаемое агентом пользователя (обычно броузером) содержимое находится внутри генерируемого эле-

ментом блока. Например, элемент `<span>эй там</span>` – незамещаемый элемент, и агент пользователя (user agent) будет отображать текст «эй там». Это выполняется для абзацев, заголовков, ячеек таблиц, списков и почти всех остальных элементов XHTML.

Роль элементов в формировании представления

Кроме замещаемых и незамещаемых, в CSS2.1 специфицированы еще два базовых типа элементов: *блочные* (*block-level*) и *строковые* (*inline-level*) *элементы*. Эти типы хорошо знакомы тем авторам, которые имеют опыт работы с разметкой HTML или XHTML и ее отображением в веб-броузерах (рис. 1.1).

Блочные элементы

Блочные элементы (*block-level elements*) генерируют блок, который (по умолчанию) полностью заполняет область содержимого своего родительского элемента; расположение других элементов, помимо этого блока, недопустимо. Тем самым он генерирует «разрывы» до и после блока элемента. Самые известные блочные элементы HTML: `p` и `div`. Замещаемые элементы могут быть блочными, но обычно они таковыми не являются.

Элементы списка – особый случай блочных элементов. Помимо того, что их поведение аналогично поведению других блочных элементов, они генерируют маркер – как правило, буллет для ненумерованных списков и число для нумерованных – который «прикрепляется» к блоку элемента. Наличие маркера – это единственное отличие элементов списка от остальных блочных элементов.

Строковые элементы

Строковые элементы (*inline-level elements*) генерируют блок элемента в строке текста и не разрывают ее. Лучший пример строкового элемента – тег `a` в XHTML. Также можно упомянуть элементы `strong` и `em`. Они не создают разрывов текста, поэтому могут находиться внутри содержимого другого элемента, не нарушая его внешний вид.

Обратите внимание, что хотя термины «блок» и «строка» в XHTML имеют много общего, между блочными и строковыми элементами существует одно принципиальное различие. В HTML и XHTML блочные

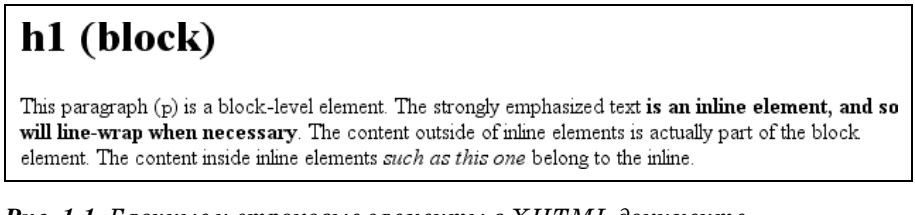


Рис. 1.1. *Блочные и строковые элементы в XHTML-документе*

элементы не могут происходить от строковых. В CSS нет ограничения на вложение ролей.

Посмотрим, как это работает, обратившись к CSS-свойству `display`.

Вы, вероятно, заметили, что оно имеет много значений, но здесь мы рассмотрим только три: `block`, `inline` и `list-item`. Все остальные значения подробно описаны в главе 2 и главе 7.

Сначала сосредоточимся на значениях `block` и `inline`. Рассмотрим следующую разметку:

```
<body>
<p>Это абзац со <em>строковым элементом</em> в нем.</p>
</body>
```

Здесь мы имеем два блочных элемента (`body` и `p`) и строковый элемент (`em`). Согласно спецификации XHTML `em` может происходить от `p`, но не наоборот. Обычно иерархия XHTML составляется таким образом, что строковые элементы могут происходить от элементов уровня блока, но не наоборот.

Напротив, в CSS нет подобных ограничений. Вы можете оставить существующую разметку, но изменить роли формирования представления этих двух элементов следующим образом:

```
p {display: inline;}
em {display: block;}
```

Этим вы заставите элементы генерировать контейнер блока внутри контейнера строки. Это совершенно законно и не нарушает ни одной спецификации. Проблема может возникнуть, если вы попытаетесь изменить порядок вложения элементов на обратный:

```
<em><p>Это абзац, ошибочно включенный в строковый элемент.</p></em>
```

Независимо от того, что вы делаете с ролями формирования представления посредством CSS, в XHTML это недопустимо.

display

Значения:	`none` | `inline` | `block` | `inline-block` | `list-item` | `run-in` | `table` | `inline-table` | `table-row-group` | `table-header-group` | `table-footer-group` | `table-row` | `table-column-group` | `table-column` | `table-cell` | `table-caption` | `inherit`
Исходное значение:	`inline`
Область применения:	все элементы
Наследование:	нет
Вычисляемое значение:	меняется для плавающих, абсолютно позиционированных и корневых элементов (см. CSS2.1, раздел 9.7); в противном случае – как задано

Изменение ролей формирования представления элементов может быть полезным в XHTML-документах и абсолютно необходимо в XML-документах. XML-документ не поддерживает каких-либо предопределенных ролей формирования представления, так что их определение остается в руках автора. Например, вы могли бы задаться вопросом, как будет представлен следующий фрагмент XML:

```
<book>
 <maintitle>Cascading Style Sheets: The Definitive Guide</maintitle>
 <subtitle>Second Edition</subtitle>
 <author>Eric A. Meyer</author>
 <publisher>O'Reilly and Associates</publisher>
 <pubdate>2004</pubdate>
 <isbn>blahblahblah</isbn>
</book>
<book>
 <maintitle>CSS2 Pocket Reference</maintitle>
 <author>Eric A. Meyer</author>
 <publisher>O'Reilly and Associates</publisher>
 <pubdate>2004</pubdate>
 <isbn>blahblahblah</isbn>
</book>
```

Поскольку по умолчанию установлено значение `display` — `inline`, содержимое будет отображаться как обычный текст строки, что показано на рис. 1.2. Такое представление практически бесполезно.

Свойство `display` позволяет определить базовую разметку:

```
book, maintitle, subtitle, author, isbn {display: block;}
publisher, pubdate {display: inline;}
```

Здесь определены пять блочных элементов и два строковых. Это означает, что каждый из блочных элементов будет интерпретироваться как элемент `div` в XHTML, а два строковых элемента будут трактоваться аналогично элементу `span`.

Эта фундаментальная возможность влиять на роли формирования представления делает CSS очень полезными в различных ситуациях. Вы можете в качестве отправной точки взять предыдущие правила, добавить несколько других стилей и получить результат, представленный на рис. 1.3.

Далее в этой книге мы исследуем различные свойства и значения, которые обеспечивают такое представление. Однако сначала нам надо рассмотреть способы связи CSS с документом. Несмотря на всю свою мощь, без установленных связей CSS не сможет воздействовать на до-

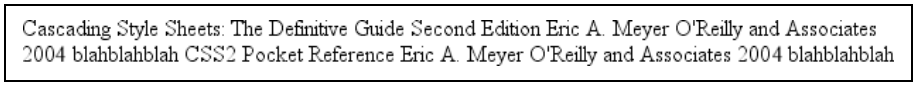


***Рис. 1.2.** Стандартное представление XML-документа*

кумент. Мы рассмотрим привязку в XHTML, поскольку она наиболее привычна.

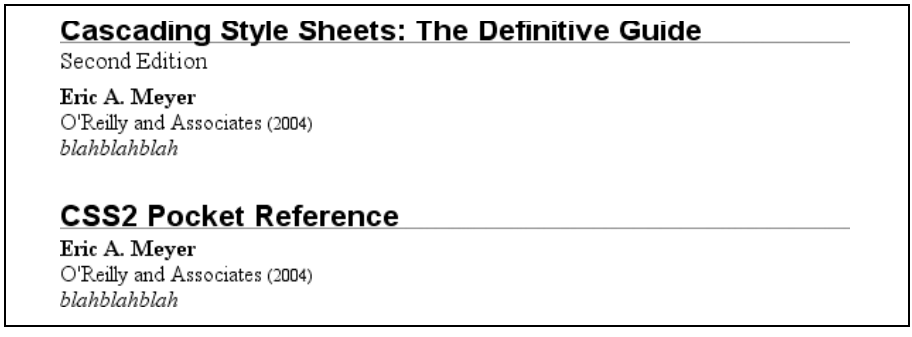


Рис. 1.3. *Представление XML-документа с использованием стилей*

Объединение CSS и XHTML

Я упомянул, что документы HTML и XHTML имеют схожую структуру, и этот момент следует повторить. Кстати, именно в этом состоит проблема со старыми веб-страницами: слишком часто мы забываем, что документы имеют внутреннюю структуру, которая совершенно отличается от визуальной. В погоне за созданием веб-страниц, выглядящих наилучшим образом, мы забываем, что страницы должны содержать информацию об их структуре.

Эта структура является неотъемлемой частью взаимоотношений между XHTML и CSS; без структуры подобная взаимосвязь вообще невозможна. Чтобы лучше понять это, давайте рассмотрим пример XHTML-документа и разберем его по частям:

```
<html>
<head>
<title>Eric's World of Waffles</title>
<link rel="stylesheet" type="text/css" href="sheet1.css" media="all" />
<style type="text/css">
@import url(sheet2.css);
h1 {color: maroon;}
body {background: yellow;}
/* Вот мои стили! Вау! */
</style>
</head>
<body>
<h1>Waffles!</h1>
<p style="color: gray;">The most wonderful of all breakfast foods is
the waffle--a ridged and cratered slab of home-cooked, fluffy goodness
that makes every child's heart soar with joy. And they're so easy to make!
Just a simple waffle-maker and some batter, and you're ready for a morning
of aromatic ecstasy!
```

```
</p>
</body>
</html>
```

Приведенная выше разметка показана на рис. 1.4.

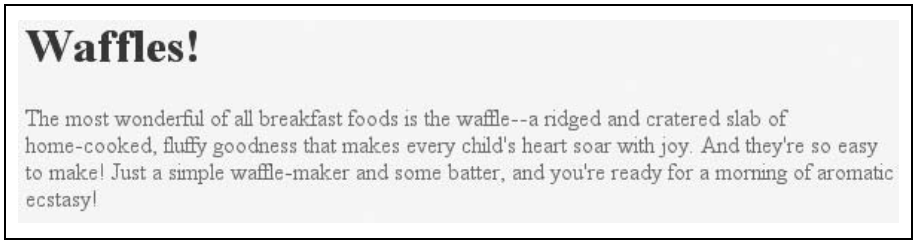


***Рис. 1.4.** Простой документ*

Теперь проанализируем различные способы, которыми этот документ связывается с CSS.

Тег link

Сначала рассмотрим применение тега `link`:

```
<link rel="stylesheet" type="text/css" href="sheet1.css" media="all" />
```

Тег `link` – это малозаметный, но совершенно полноценный тег, который годами маялся в спецификации HTML, ожидая достойного применения. Его основное назначение – предоставить авторам HTML возможность устанавливать связь между документом, содержащим тег `link`, и другими документами. В случае CSS он связывает таблицы стилей с документом; на рис. 1.5 таблица стилей *sheet1.css* связана с документом.

Таблицы стилей, не являющиеся частью HTML-документа, но используемые им, называют *внешними таблицами стилей* (*external style sheets*). Это название дано потому, что такие таблицы стилей являются внешними по отношению к HTML-документу. (Надо же!)

Чтобы внешняя таблица стилей была успешно загружена, тег `link` должен быть помещен внутрь элемента `head` и не может находиться внутри какого-либо другого элемента, так же как `title`. В результате веб-броузер отыщет и загрузит таблицу стилей, после чего будет использовать любые содержащиеся в ней стили для формирования представления HTML-документа, как показано на рис. 1.5.

А каков формат внешней таблицы стилей? Это просто список таких же правил, какие мы видели в предыдущем разделе и в примере XHTML-документа, но в данном случае правила сохраняются в отдельном файле. Просто помните, что в таблицу стилей не может быть включен ни XHTML, ни любой другой язык разметки, а только стилевые правила. Вот как может выглядеть содержимое внешней таблицы стилей:

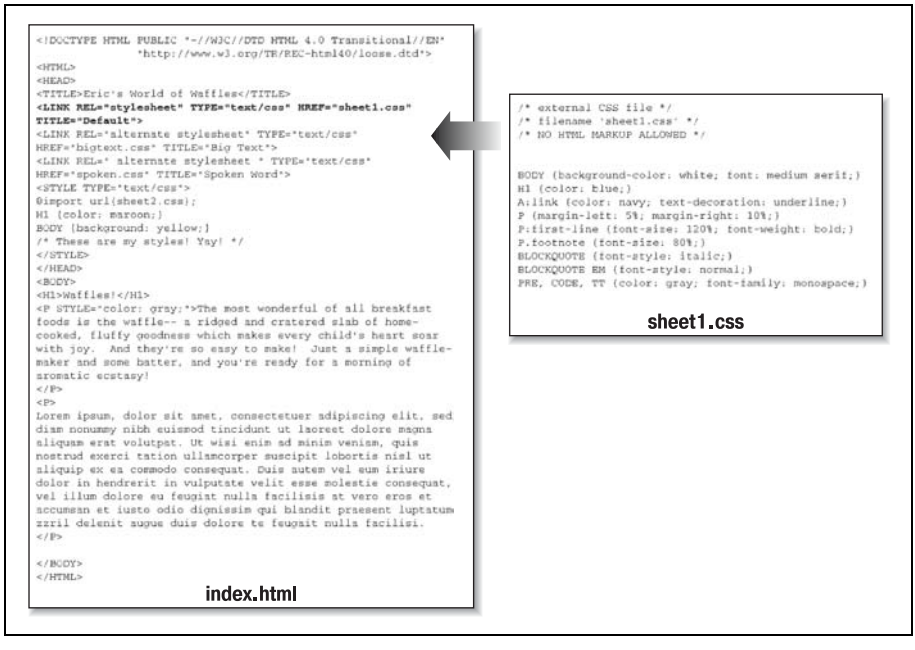


Рис. 1.5. *Применение внешней таблицы стилей к документу*

```
h1 {color: red;}
h2 {color: maroon; background: white;}
h3 {color: white; background: black;
 font: medium Helvetica;}
```

Вот и все: вообще никакой HTML-разметки или комментариев, только простые и понятные объявления стилей. Их сохраняют в простом текстовом файле и обычно дают расширение *.css*, как в *sheet1.css*.

Внешняя таблица стилей не может содержать какой-либо разметки документа, только правила и комментарии CSS (и те, и другие рассматриваются в этой главе позже). Наличие во внешней таблице стилей разметки может привести к тому, что некоторые ее части или вообще вся таблица будут проигнорированы.

Расширение имени файла не является обязательным, но некоторые более старые броузеры не смогут опознать файл как содержащий таблицу стилей, если он не заканчивается на *.css*, даже если вы *включили* верный тип элемента `text/css` в элемент `link`. Кстати, некоторые вебсерверы не будут передавать файл как `text/css`, если его имя не заканчивается на *.css*, хотя обычно с этим можно справиться, изменив файлы конфигурации сервера.

Атрибуты

Все остальные атрибуты и значения тега `link` довольно просты. Атрибут `rel` отвечает за установку взаимосвязи и в данном случае имеет значение `stylesheet`. Атрибуту `type` всегда присваивается значение `text/css`. Оно описывает тип данных, которые будут загружены с помощью тега `link`. Таким образом, веб-броузер знает, что таблица стилей – это таблица стилей CSS. Это определяет, как броузер будет обрабатывать импортируемые им данные. Однако в будущем возможно использование других языков описания стилей, поэтому важно объявлять, какой именно язык используется.

Далее расположен атрибут `href`. Значением этого атрибута является URL таблицы стилей. Он может быть как абсолютным, так и относительным в зависимости от ваших потребностей. В нашем примере URL относительный. Но он мог бы быть и абсолютным: `http://www.meyerweb.com/sheet1.css`.

И наконец, у нас есть атрибут `media`. В нашем случае он имеет значение `all`, т. е. эта таблица стилей должна использоваться для всех средств визуального представления. CSS2 определяет несколько допустимых значений для этого атрибута:

`all`

Указывается для всех устройств.

`aural`

Задают для синтезаторов речи, средств чтения с экрана и других аудио-представлений документа.

`braille`

Задают при выводе документа на устройствах отображения азбуки Брайля.

`embossed`

Указывается при печати документа азбукой Брайля.

`handheld`

Задается для переносных устройств, таких как карманные компьютеры или смартфоны.

`print`

Задается при распечатке документа на обычном принтере, а также при выводе в окне просмотра документа перед печатью.

`projection`

Указывают для проекционных устройств, таких как цифровой проектор, используемый для демонстрации слайдов, сопровождающей чтение доклада.

`screen`

Указывают при представлении документа в экранном устройстве, таком как монитор компьютера. Все веб-броузеры, выполняющие-

ся на подобных системах, являются экранными агентами пользователя.

`tty`

Применяется для устройств, использующих набор символов с фиксированной шириной, таких как телетайп.

`tv`

Задают при отображении документа на телевизоре.

Основная масса этих устройств не поддерживается ни одним из современных веб-броузеров. Три из наиболее широко распространенных: `all`, `screen` и `print`. Opera также поддерживает тип `projection`, что позволяет представлять документ в виде последовательности слайдов.

Таблицу стилей можно использовать не только для одного устройства, но и для нескольких, указав разделенный запятыми список этих устройств. Таким образом, например, вы можете применить связанную таблицу стилей как для экранного, так и для проекционного устройства:

```
<link rel="stylesheet" type="text/css" href="visual-sheet.css"
   media="screen, projection" />
```

Обратите внимание, что с документом может быть ассоциирована не одна связанная таблица стилей. В таких случаях в исходном представлении документа будут применяться только те теги `link`, атрибут `rel` которых имеет значение `stylesheet`. Таким образом, если бы вы захотели связать две таблицы стилей, *basic.css* и *splash.css*, это выглядело бы следующим образом:

```
<link rel="stylesheet" type="text/css" href="basic.css" />
<link rel="stylesheet" type="text/css" href="splash.css" />
```

При этом броузер будет загружать обе таблицы стилей, комбинируя правила обеих, и применять их все к документу. (Как именно происходит комбинирование таблиц, мы увидим в главе 3, но на данный момент давайте просто примем, что они комбинируются.) Например:

```
<link rel="stylesheet" type="text/css" href="basic.css" />
<link rel="stylesheet" type="text/css" href="splash.css" />

<p class="a1">Этот абзац будет серым, только если
применены стили из таблицы стилей 'basic.css'.</p>
<p class="b1">Этот абзац будет серым, только если
применены стили из таблицы стилей 'splash.css'.</p>
```

Единственный атрибут, который не приведен в примере разметки, но мог бы там находиться, – это `title`. Он редко используется, но в будущем может обрести важное значение, а его неправильное применение может иметь неожиданные последствия. Почему? Мы рассмотрим это в следующем разделе.

Альтернативные таблицы стилей

Существует возможность определения *альтернативных таблиц стилей* (*alternate style sheets*). Для этого атрибуту `rel` присваивается значение `alternate stylesheet`, и тогда таблицы стилей задействуются в представлении документа только в том случае, если их выбирает пользователь.

Для организации работы с альтернативными таблицами стилей броузер возьмет значения атрибутов `title` элементов `link` и создаст из них список альтернативных стилей. Так что вы можете написать следующее:

```
<link rel="stylesheet" type="text/css"
   href="sheet1.css" title="По умолчанию" />
<link rel="alternate stylesheet" type="text/css"
   href="bigtext.css" title="Крупный текст" />
<link rel="alternate stylesheet" type="text/css"
   href="zany.css" title="Сумасшедшие цвета!" />
```

В результате пользователи могут выбирать стили, и броузер переключится с первого (в данном случае помечен словами «По умолчанию») на любой другой выбранный пользователем стиль. На рис. 1.6 показан один из вариантов реализации этого механизма выбора.

Альтернативные таблицы стилей поддерживаются большинством броузеров, созданных компанией Netscape Communications, такими как Mozilla и Netscape 6+, и в Opera 7. Броузеры Internet Explorer не поддерживают альтернативные таблицы стилей, но в них того же эффекта можно достичь с помощью JavaScript.

Альтернативные таблицы стилей можно группировать, присваивая их атрибуту `title` одинаковые значения. Благодаря этому пользователь может выбирать разные представления сайта, как на экране, так и при печати. Например:

```
<link rel="stylesheet" type="text/css"
   href="sheet1.css" title="По умолчанию" media="screen" />
```

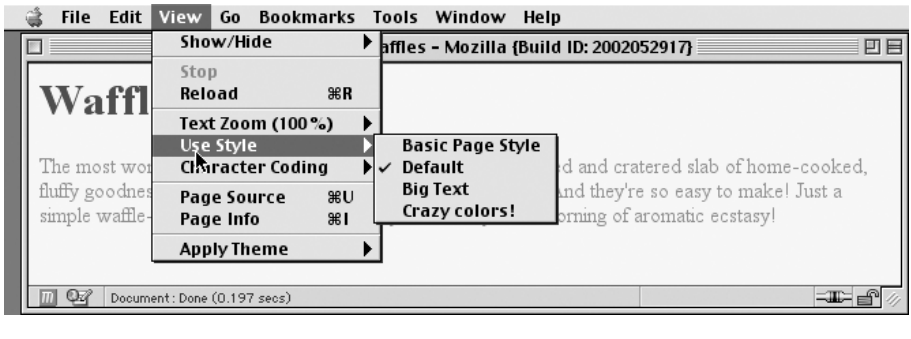


***Рис. 1.6.** Броузер, предлагающий выбрать альтернативные таблицы стилей*

```
<link rel="stylesheet" type="text/css"
   href="print-sheet1.css" title="По умолчанию" media="print" />
<link rel="alternate stylesheet" type="text/css"
   href="bigtext.css" title="Крупный текст" media="screen" />
<link rel="alternate stylesheet" type="text/css"
   href="print-bigtext.css" title="Крупный текст" media="print" />
```

Если в соответствующем агенте пользователя из списка альтернативных таблиц стилей выбран вариант «Крупный текст», то стиль документа при отображении на экране будет формироваться по таблице *bigtext.css*, а при выводе на печать – по *print-bigtext.css*. Ни в одном из устройств не будут использованы ни *sheet1.css*, ни *print-sheet1.css*.

Почему? Потому что если в теге `link` с атрибутом `rel`, имеющим значение `stylesheet`, указан заголовок, данная таблица стилей помечается как *предпочтительная таблица стилей* (*preferred style sheet*). Это значит, что из всех альтернативных таблиц стилей ее применение предпочтительнее, и именно она будет использоваться при первом выводе документа на экран. Однако если выбрать альтернативную таблицу стилей, предпочтительная таблица использоваться *не будет*.

Более того, если обозначить как предпочтительные несколько таблиц стилей, все они, кроме одной, будут проигнорированы. Рассмотрим:

```
<link rel="stylesheet" type="text/css"
   href="sheet1.css" title="Раскладка по умолчанию" />
<link rel="stylesheet" type="text/css"
   href="sheet2.css" title="Шрифт по умолчанию" />
<link rel="stylesheet" type="text/css"
   href="sheet3.css" title="Цвета по умолчанию" />
```

Все три элемента `link` теперь обозначены как предпочтительные таблицы стилей, благодаря наличию во всех трех атрибута `title`, но на самом деле только один из них выступает в этом качестве. Остальные два будут полностью проигнорированы. Какие именно из них? Точно ответить на этот вопрос невозможно, поскольку ни HTML, ни XHTML не предоставляют метода, позволяющего определить, какая из предпочтительных таблиц стилей должна быть проигнорирована, а какая нет.

Если просто не указать заголовок таблицы стилей, то она становится *постоянной таблицей стилей* (*persistent style sheet*) и всегда используется в представлении документа. Зачастую именно этого и хочет автор.

Элемент style

Элемент `style` – это единственный способ включения таблиц стилей и располагается он в самом документе:

```
<style type="text/css">
```

В элементе `style` всегда должен присутствовать атрибут `type`; если документ использует CSS, его значение должно быть `"text/css"`, так же как и для элемента `link`.

Элемент style должен всегда начинаться с <style type="text/css">, как показано в предыдущем примере. Далее следуют один или несколько стилей, и все завершается закрывающим тегом </ style>. Также элементу style можно присвоить атрибут media, допустимые значения которого не отличаются от тех, которые обсуждались ранее при рассмотрении связанных таблиц стилей.

Стили, включаемые между открывающим и закрывающим тегами style, называют *таблицей стилей документа* (*document style sheet*), или *вложенной таблицей стилей* (*embedded style sheet*), поскольку эта таблица стилей вложена в документ. В ней будут содержаться применяемые к документу стили, но она также может включать ссылки на внешние таблицы стилей с помощью директивы @import.

Директива @import

Теперь мы займемся тем, что находится внутри тега style. Для начала у нас есть аналог элемента link – директива @import:

```
@import url(sheet2.css);
```

Так же как и link, @import может указывать веб-броузеру на необходимость загрузки внешней таблицы стилей и использования ее стилей при формировании представления HTML-документа. Единственное основное отличие заключается в синтаксисе и размещении команды. Как видите, @import находится в контейнере style. Она должна располагаться перед всеми остальными правилами CSS, иначе она вообще не будет работать. Рассмотрим такой пример:

```
<style type="text/css">
@import url(styles.css); /* @import идет самой первой */
h1 {color: gray;}
</style>
```

В документе может быть несколько директив выражения @import, как и тегов link. Однако, в отличие от link, все указанные в директивах @import таблицы стилей будут загружены и использованы; с помощью @import невозможно назначить альтернативные таблицы стилей. Итак, исходя из следующей разметки:

```
@import url(sheet2.css);
@import url(blueworld.css);
@import url(zany.css);
```

все три внешние таблицы стилей будут загружены, и все их правила будут использоваться в визуальном представлении документа.

Многие старые броузеры не могут обрабатывать разнообразные формы директивы @import. Фактически это может применяться специально для «сокрытия» стилей от этих броузеров. Более подробную информацию можно найти по адресу *http://w3development.de/css/hide_css_from_browsers*.

Как и в случае элемента link, можно ограничить применение импортируемой таблицы стилей одним или несколькими устройствами, перечислив их после URL таблицы стилей:

```
@import url(sheet2.css) all;
@import url(blueworld.css) screen;
@import url(zany.css) projection, print;
```

Директива @import крайне полезна для переноса стилей между внешними таблицами стилей. Поскольку внешние таблицы стилей не могут содержать никакой разметки документа, элемент link применить нельзя, а директиву @import можно. Следовательно, можно себе представить внешнюю таблицу стилей такого содержания:

```
@import url(http://example.org/library/layout.css);
@import url(basic-text.css);
@import url(printer.css) print;
body {color: red;}
h1 {color: blue;}
```

Ну, может быть, не именно эти стили, но идея ясна. Обратите внимание на использование в примере как абсолютного, так и относительных URL. Можно применять любую форму URL, так же как и в link.

Также заметьте, что директивы @import находятся в начале таблицы стилей, как и в примере нашего документа. CSS требует, чтобы директива @import располагалась в таблице стилей раньше всех остальных правил. Директива @import, расположенная после правил (например, после body {color: red;}), будет проигнорирована соответствующими агентами пользователя.

Internet Explorer для Windows не игнорирует директивы @import, даже если они следуют после остальных правил. Поскольку другие броузеры игнорируют неверно размещенные директивы @import, очень легко ошибиться – поставить директивы @import не туда, куда надо, и таким образом изменить представление в других броузерах.

Действительные правила стилей

После инструкции @import в нашем примере мы находим несколько обычных правил стилей. На данный момент совершенно не важно, что они реально означают, хотя вы, вероятно, можете догадаться, что они задают красно-коричневый цвет для элементов h1 и желтый фон для элементов body:

```
h1 {color: maroon;}
body {background: yellow;}
```

Аналогичные правила встречаются в большинстве встроенных таблиц стилей – простых и сложных, коротких и длинных. Документ, в котором элемент style не содержит никаких правил, – редкость.

Обратная совместимость

Необходимо предупредить тех, кто беспокоится о том, чтобы сделать свои документы доступными для более старых броузеров. Вероятно, вы знаете, что броузеры игнорируют теги, которые не могут распознать; например, если веб-страница содержит тег `blooper`, броузеры полностью проигнорируют этот тег, потому что они его не знают.

То же касается и таблиц стилей. Если броузер не знает элементов `<style>` и `</style>`, он проигнорирует их. Однако это не означает, что будут проигнорированы и объявления, содержащиеся между этими тегами, потому что броузер интерпретирует их как обычный текст. Поэтому объявления стилей появятся в верхней части вашей страницы! (Конечно, броузер должен игнорировать текст, потому что он не является частью элемента `body`, но это происходит не всегда.)

Для решения этой проблемы рекомендуется заключать ваши объявления в тег комментария. В приведенном здесь примере начало тега комментария находится сразу после открывающего тега `style`, а конец комментария – сразу перед закрывающим тегом `style`:

```
<style type="text/css"><!--
@import url(sheet2.css);
h1 {color: maroon;}
body {background: yellow;}
--></style>
```

В результате старые броузеры проигнорируют объявления, так же как и теги `style`, потому что комментарии HTML не отображаются. Но те броузеры, которые понимают CSS, смогут прочитать таблицу стилей.

Комментарии CSS

В CSS также предусмотрены комментарии. Они очень похожи на комментарии в C/C++, поскольку ограничиваются символами `/*` и `*/`:

```
/* Это комментарий CSS1 */
```

Комментарии могут распространяться на несколько строк, как и в C++:

```
/* Это комментарий CSS1, и он может занимать
 несколько строк без всяких проблем. */
```

Важно помнить, что комментарии CSS не могут быть вложенными. Поэтому следующий пример не будет правильным:

```
/* Это комментарий, в котором мы находим
 другой комментарий, что НЕПРАВИЛЬНО
   /* Другой комментарий */
 и вновь первый комментарий */
```

Однако вряд ли вложенные комментарии кому-нибудь когда-нибудь понадобятся, поэтому данное ограничение не имеет особого значения.

Можно случайно получить «вложенные» комментарии, если временно закомментировать большой фрагмент таблицы стилей, уже содержащий комментарий. Поскольку в CSS вложенные комментарии не допускаются, «внешний» комментарий будет заканчиваться в точке окончания «внутреннего».

Если вы хотите поместить комментарии на одной строке с правилом, следует быть аккуратным. Например, здесь все правильно:

```
h1 {color: gray;}   /* Этот комментарий CSS растянулся на несколько строк, */
h2 {color: silver;} /* но поскольку он размещается параллельно со */
p {color: white;}   /* стилями, каждая строка должна быть */
pre {color: gray;}  /* разграничена знаками комментариев. */
```

Если в этом примере не ограничивать комментарий на каждой строке, большая часть таблицы стилей станет частью комментария и поэтому не будет работать:

```
h1 {color: gray;}   /* Этот комментарий CSS растянулся на несколько
h2 {color: silver;}  строк, но поскольку он не разграничен
p {color: white;}    знаками комментариев, последние три
pre {color: gray;}   стиля стали частью комментария. */
```

В этом примере только первое правило (`h1 {color: gray;}`) будет применено к документу. Остальные правила как часть комментария игнорируются механизмом визуализации броузера.

Продолжим разбор нашего примера, чтобы увидеть, как можно поместить информацию CSS в теги XHTML!

Подставляемые в строку стили

В тех случаях, когда вы хотите просто назначить несколько стилей отдельному элементу, не применяя встроенные или внешние таблицы стилей, задайте *подставляемый в строку стиль* (*inline style*) посредством HTML-атрибута `style`:

```
<p style="color: gray;">Самый прекрасный завтрак -
это вафля, рифленый кусочек домашнего воздушного совершенства...
</p>
```

Атрибут `style` может быть ассоциирован с любым тегом HTML, за исключением тех, которые располагаются вне элемента `body` (`head` или `title`, например).

Синтаксис атрибута `style` достаточно прост. Кстати, он очень похож на объявления, размещающиеся в контейнере `style`, за исключением того, что фигурные скобки заменяются двойными кавычками. Таким образом, согласно выражению `<p style="color: maroon; background: yellow;">` цвет текста станет красно-коричневым, а фон – желтым, но *только для этого абзаца*. Ни на какую другую часть документа это объявление не повлияет.

Заметьте, что в подставляемый в строку атрибут `style` можно ввести только блок описания, а не всю таблицу стилей. Следовательно, нельзя вставить в атрибут `style` ни директиву `@import`, ни включить какие-либо полные правила. Вы можете помещать в значение атрибута `style` только то, что может находиться в правиле между фигурными скобками.

Применение атрибута `style` в общем нежелательно. Кроме того, спецификацией XHTML 1.1 он отмечен как нерекомендуемый и вряд ли появится в других языках XML, кроме XHTML. При размещении стилей в атрибуте `style` некоторые из основных преимуществ CSS – способность организовывать централизованные стили, управляющие представлением всего документа или всех документов веб-сервера, – оказываются утраченными. Подставляемые в строку стили не намного лучше, чем тег `font`, хотя они обладают существенно большей гибкостью.

Заключение

Применяя CSS, можно полностью изменить способ представления элементов агентом пользователя. Это может быть сделано просто с помощью свойства `display` или по-другому – путем связывания таблиц стилей с документом. Пользователь никогда не будет знать, сделано ли это с помощью внешних или встроенных таблиц стилей или даже через подставляемый в строку стиль. В случае применения внешних таблиц стилей действительно важно, каким образом они делают возможным для авторов размещение всей информации о представлении сайта в одном месте и как они направляют туда все документы. Это не только облегчает обновление и эксплуатацию сайта, но помогает сохранить пропускную способность канала передачи данных, поскольку из документов удаляется вся информация о представлении.

Чтобы использовать CSS на полную мощность, авторам надо знать, как связать набор стилей с элементами документа. Чтобы полностью понять, как CSS со всем этим справляется, необходимо глубоко понимать, каким образом CSS выбирает части документа для стилевого оформления, что и обсуждается в следующей главе.