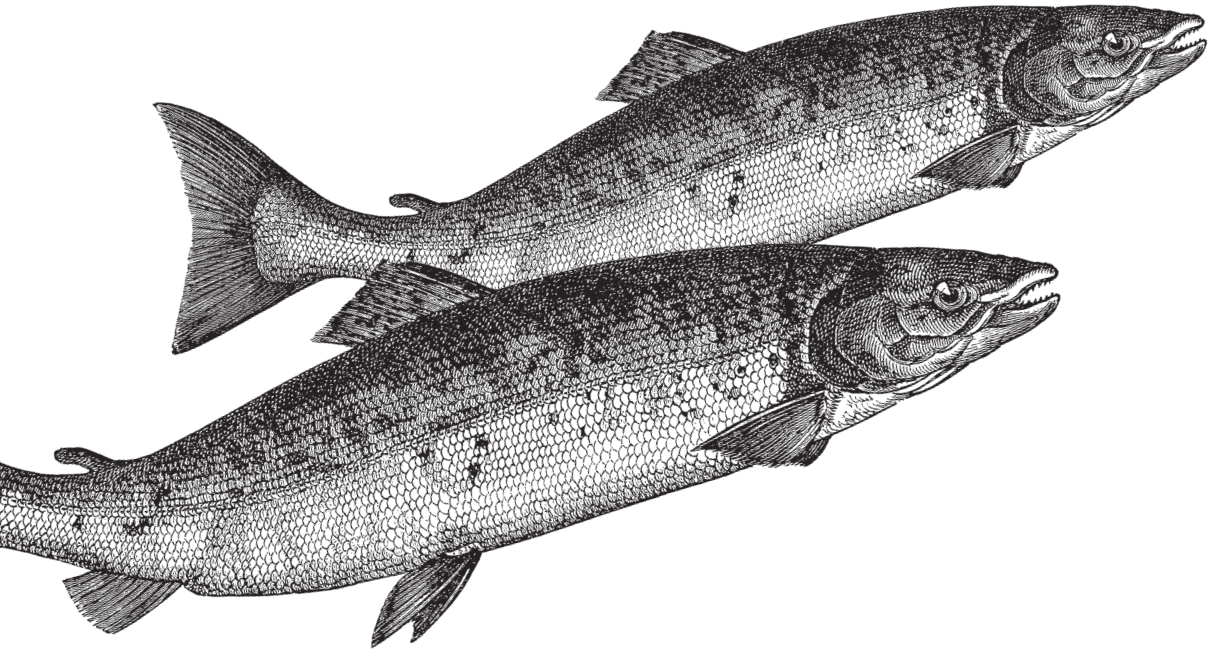
Визуальное представление веб-документов

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

4

Значения и единицы измерения

В этой главе мы коснемся элементов, составляющих основу практически всего, что позволяют делать CSS, – *единиц измерений* (*units*), применяемых во всех свойствах для задания цвета, расстояний и размеров. Без единиц измерения нельзя объявить о том, что текст абзаца должен быть фиолетовым, или что вокруг изображения должно быть пустое пространство в 10 пикселов, или что текст заголовка должен иметь определенный размер. Поняв основные принципы, изложенные в этой главе, вы намного быстрее изучите и начнете использовать все остальное в CSS.

Числа

В CSS существует два типа чисел: *целые* (*integer*) и *вещественные* (*real*). Эти типы чисел в большинстве случаев служат базой для всех остальных типов значений, но иногда в качестве значений свойств могут выступать числа других форматов.

В CSS2.1 вещественное число определяется как целое, за которым могут следовать десятичная точка и дробная часть. Таким образом, все эти числа являются действительными числовыми значениями: `15.5`, `-270.00004` и `5`. И целые, и вещественные числа могут быть как положительными, так и отрицательными, хотя свойства могут (и часто так оно и есть) ограничивать диапазон принимаемых ими чисел.

Процентные значения

Процентное значение (*percentage value*) – это вычисляемое вещественное число, за которым следует знак процента (`%`). Процентные значения практически всегда выражены относительно другого значения, которым может быть все что угодно, включая значение другого свойст-

ва того же элемента, значение, унаследованное от родительского элемента, или значение элемента-предка. Любое свойство, принимающее значения, задаваемые в процентах, определяет свои ограничения на допустимый диапазон процентных значений и точность представления относительных процентных значений.

Цвет

Один из первых вопросов, задаваемых каждым начинающим автором веб-страниц: «Как устанавливать цвета на странице?». HTML предоставляет на выбор два варианта: взять один из немногочисленных именованных цветов, такой как `red` или `purple`, либо оперировать таинственными шестнадцатеричными кодами. Оба этих способа описания цветов сохранились и в CSS, но появились и некоторые другие, как я полагаю, более понятные методы.

Именованные цвета

Тем, кого удовлетворяет небольшой базовый набор цветов, проще всего указать имя цвета. CSS достаточно логично называет все эти предоставляемые на выбор цвета *именованными цветами* (*named colors*).

Вопреки тому, во что нас пытаются заставить поверить некоторые создатели броузеров, имеется очень небольшое количество действительных ключевых слов для обозначения именованных цветов. Например, нельзя выбрать «перламутровый» цвет, поскольку для него ключевое слово отсутствует. Что касается CSS2.1, спецификация CSS определяет 17 цветов. Это 16 цветов, описанных в HTML 4.01, плюс оранжевый (`orange`):

aqua	fuchsia	lime	olive	red	white
black	gray	maroon	orange	silver	yellow
blue	green	navy	purple	teal	

Итак, скажем, вы хотите, чтобы все заголовки первого уровня были красно-коричневыми. Лучше всего объявить их так:

```
h1 {color: maroon;}
```

Просто и понятно, не так ли? На рис. 4.1 показаны еще несколько примеров:

```
h1 {color: gray;}
h2 {color: silver;}
h3 {color: black;}
```

Конечно, встречаются (а может быть, вы даже сами использовали) названия цветов, отсутствующие в приведенном выше списке. Например, если задать:

```
h1 {color: lightgreen;}
```

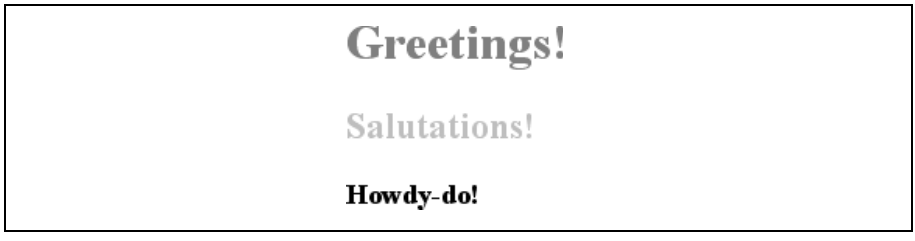


Рис. 4.1. *Именованные цвета*

Скорее всего, все элементы `h1` и в самом деле станут светло-зелеными, несмотря на то, что ключевого слова `lightgreen` нет в списке именованных цветов CSS2.1. Дело в том, что большинство веб-броузеров распознают до 140 цветов, включая стандартные 17. Эти дополнительные цвета определены в цветовой спецификации CSS3, которая в данной книге не рассматривается. Эти стандартные цвета (на момент написания книги), вероятно, надежнее, чем более длинный список из 140 или около этого цветов, потому что коды цветов для этих 17 определены в CSS2.1. Расширенный список из 140 цветов, появившийся в CSS3, базируется на стандартных значениях X11 RGB, которые используются десятилетиями, поэтому они, вероятно, будут поддерживаться очень хорошо.

К счастью, в CSS существуют более точные способы задания цветов. Их преимущество в том, что они позволяют определить любой цвет цветового спектра, а не только 17 (или 140) именованных цветов.

Цветовая модель RGB

Компьютеры создают цвета путем комбинирования различных уровней красного, зеленого и синего. Такую комбинацию часто называют *цветовой моделью RGB*. Кстати, если открыть старый ЭЛТ-монитор компьютера и залезть в проекционный кинескоп поглубже, можно обнаружить три электронных пушки. (Однако вы их лучше не ищите, если не хотите лишиться гарантии на монитор.) Эти пушки испускают в каждую точку экрана пучки электронов с различной интенсивностью. В этих точках яркости каждого из лучей комбинируются, образуя все цвета, которые вы видите. Каждая точка называется *пикселом* (*pixel*). К данному термину мы вернемся в этой главе несколько позже. Даже несмотря на то, что в большинстве современных мониторов электронные пушки уже не используются, их цветовой вывод все равно основан на комбинации трех цветов.

Исходя из того, как происходит формирование цвета на мониторе, необходим прямой доступ к этим цветам, чтобы была возможность определять собственные сочетания трех цветов. Это непросто, но реально, и результат стоит того, потому что в этом случае ресурсы создания цветов практически не ограничены. Существует четыре способа формирования цвета.

Функциональные RGB-цвета

Существует два типа кодов цветов, основанных на *функциональном формате записи RGB* (*functional RGB notation*), в противоположность шестнадцатеричной нотации. Обобщенный синтаксис этого типа кодировки цвета – `rgb(color)`, где `color` представляет собой комбинацию трех процентных значений или целых чисел. Допустимый диапазон процентных значений – от `0%` до `100%`, а диапазон целых значений – от `0` до `255`.

Следовательно, код, задающий белый и черный цвета с помощью процентных значений, будет таким:

```
rgb(100%, 100%, 100%)
rgb(0%, 0%, 0%)
```

А вот те же цвета, представленные записью из целых чисел (integer-triplet notation):

```
rgb(255, 255, 255)
rgb(0, 0, 0)
```

Предположим, требуется, чтобы элементы `h1` были закрашены произвольным оттенком красного – чем-то средним между красным и красно-коричневым. Цвет `red` эквивалентен записи `rgb(100%, 0%, 0%)`, тогда как `maroon` равен `(50%, 0%, 0%)`. Можно попробовать получить цвет, промежуточный между этими двумя, посредством такой записи:

```
h1 {color: rgb(75%, 0%, 0%);}
```

При этом красный компонент цвета становится светлее, чем `maroon`, но темнее, чем `red`. Если же надо получить бледно-красный цвет, то можно повысить уровни зеленого и синего:

```
h1 {color: rgb(75%, 50%, 50%);}
```

Самый близкий к этому эквивалент цвета при записи с помощью целых чисел:

```
h1 {color: rgb(191, 127, 127);}
```

Самый простой способ визуализировать соответствия цвет-код состоит в том, чтобы создать таблицу кодов для оттенков серого. Кроме того, на иллюстрации в книге можно изобразить только шкалу оттенков серого (рис. 4.2):

```
p.one {color: rgb(0%, 0%, 0%);}
p.two {color: rgb(20%, 20%, 20%);}
p.three {color: rgb(40%, 40%, 40%);}
p.four {color: rgb(60%, 60%, 60%);}
p.five {color: rgb(80%, 80%, 80%);}
p.six {color: rgb(0, 0, 0);}
p.seven {color: rgb(51, 51, 51);}
p.eight {color: rgb(102, 102, 102);}
p.nine {color: rgb(153, 153, 153);}
p.ten {color: rgb(204, 204, 204);}
```

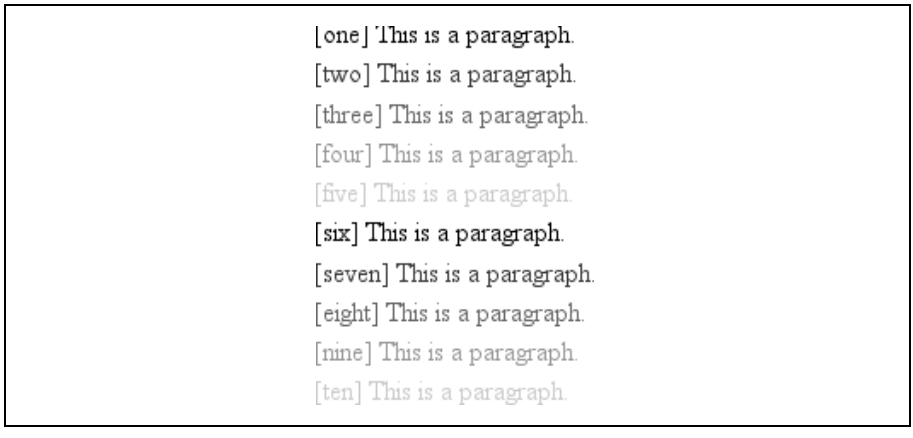


***Рис. 4.2.** Текст, представленный в оттенках серого*

Конечно, поскольку мы работаем с оттенками серого, все три компонента RGB-кода в каждом выражении одинаковые. Если бы любое из них отличалось от других, начал бы проявляться цвет. Если, например, `rgb(50%, 50%, 50%)` превратить в `rgb(50%, 50%, 60%)`, получился бы серый цвет с небольшой синевой.

Процентные значения можно записывать дробными числами. Например, по каким-либо причинам вы хотите определить цвет, содержащий точно 25,5% красного, 40% зеленого и 98,6% синего:

```
h2 {color: rgb(25.5%,40%,98.6%);}
```

Если агент пользователя игнорирует десятичные точки (а некоторые так и поступают), он округлит значение до ближайшего целого, и в результате будет объявлено `rgb(26%, 40%, 99%)`. Запись в виде трех целых ограничивает нас, естественно, целыми значениями.

Значения, которые выходят за рамки допустимого диапазона для каждого из форматов записи, «урезаются» до ближайшей границы диапазона. Это значит, что значение, которое больше `100%` или меньше `0%`, будет по умолчанию приведено к этим допустимым границам. Таким образом, следующие объявления были бы интерпретированы так, как показано в комментариях:

```
P.one {color: rgb(300%,4200%,110%);}   /* 100%,100%,100% */
P.two {color: rgb(0%,-40%,-5000%);}    /* 0%,0%,0% */
p.three {color: rgb(42,444,-13);}      /* 42,255,0 */
```

Может показаться, что преобразования между процентными значениями и целыми числами выполняются произвольным образом, но на самом деле для их вычисления существует простая формула. Если известно процентное значение каждого из уровней RGB, то для того чтобы получить окончательные значения, достаточно составить пропорцию с числом 255. Скажем, есть цвет, содержащий 25% красного,

37,5% зеленого и 60% синего. Умножьте каждое из этих процентных значений на 255 и разделите результат на 100, после чего получите 63,75; 95,625 и 153. Затем округлите числа до ближайших целых: `rgb(64, 96, 153)`.

Конечно, если процентные значения известны, то не составит труда преобразовать их в целые. Запись в виде целых удобнее тем, кто работает с такими программами, как Photoshop, отображающими в диалоговом окне «Info» целые значения, или тем, кто настолько близко знаком с деталями формирования цветов, что привык «мыслить» числами 0–255. В последнем случае, вероятно, удобнее оперировать шестнадцатеричными числами, что мы и обсудим в следующем разделе.

Шестнадцатеричные RGB-цвета

CSS позволяет определять цвет с помощью шестнадцатеричной записи, такой близкой авторам веб-страниц, работающим с HTML:

```
h1 {color: #FF0000;}   /* делаем заголовки H1 красными */
h2 {color: #903BC0;}   /* делаем заголовки H2 темно-фиолетовыми */
h3 {color: #000000;}   /* делаем заголовки H3 черными */
h4 {color: #808080;}   /* делаем заголовки H4 умеренно-серыми */
```

Шестнадцатеричная форма записи («hex notation») применяется в компьютерном мире уже довольно давно, и программисты обычно знают ее. Такая распространенность шестнадцатеричной системы счисления, возможно, и послужила причиной ее использования при задании цветов в HTML старой школы. Эта практика была просто перенесена в CSS.

Вот как это делается: цвет задается посредством объединения трех шестнадцатеричных чисел в диапазоне от `00` до `FF`. Обобщенный синтаксис для этой формы записи – `#RRGGBB`. Обратите внимание, что здесь между числами нет ни пробелов, ни запятых, ни каких-либо других разделителей.

Шестнадцатеричная форма записи с математической точки зрения эквивалентна записи целыми числами, обсуждаемой в предыдущем разделе. Например, запись `rgb(255, 255, 255)` эквивалентна `#FFFFFF`, а `rgb(51, 102, 128)` – это то же самое, что и `#336680`. Выбирайте любую форму записи, в большинстве агентов пользователя они интерпретируются одинаково. При наличии калькулятора, умеющего осуществлять преобразования между десятичными и шестнадцатеричными числами, переход от одних к другим не составит труда.

Для шестнадцатеричных чисел, составленных из трех согласованных пар символов, CSS допускает более короткую запись. Обобщенный синтаксис для этой формы записи – `#RGB`.

```
h1 {color: #000;}   /* делаем заголовки H1 черными */
h2 {color: #666;}   /* делаем заголовки H2 темно-серыми */
h3 {color: #FFF;}   /* делаем заголовки H3 белыми */
```

Как видите, в каждом значении цвета присутствует только три символа. Однако для записи шестнадцатеричных чисел в диапазоне от `00` до `FF` необходимо по два символа для каждого, а у вас всего три символа. Как же работает этот метод?

Секрет в том, что броузер удваивает каждый символ. Следовательно, запись `#F00` эквивалентна `#FF0000`, `#6FA` будет аналогична `#66FFAA`, а `#FFF` будет преобразована в `#FFFFFF`, что соответствует `white`. Очевидно, что не каждый цвет может быть представлен таким образом. Умеренно-серый, например, надо записывать в стандартной шестнадцатеричной форме как `#808080`. Это значение не может быть представлено в краткой форме, ближайший эквивалент – `#888`, что аналогично `#888888`.

Сводим вместе все формы записи цветов

В табл. 4.1 представлен обзор некоторых из рассмотренных нами цветов. Возможно, ключевые слова, обозначающие цвета, не будут распознаваться броузерами, и поэтому цвета должны задаваться с помощью RGB или шестнадцатеричных кодов (просто для надежности). Кроме того, для некоторых цветов здесь вообще не представлены укороченные шестнадцатеричные коды. В таких случаях длинные (шестиразрядные) коды не могут быть представлены в краткой форме, потому что символы в них не дублируются. Например, значение `#880` разворачивается в `#888800`, а не в `#808000` (в другом представлении – `olive`). Поэтому для кода `#808000` краткой версии не существует, и соответствующая ей запись таблицы пуста.

Таблица 4.1. Эквивалентные записи цветов

Цвет	Процентные соотношения	Числовая	Шестнадцатеричная	Краткая шестнадцатеричная
red	rgb(100%,0%,0%)	rgb(255,0,0)	#FF0000	#F00
orange	rgb(100%,40%,0%)	rgb(255,102,0)	#FF6600	#F60
yellow	rgb(100%,100%,0%)	rgb(255,255,0)	#FFFF00	#FF0
green	rgb(0%,50%,0%)	rgb(0,128,0)	#008000	
blue	rgb(0%,0%,100%)	rgb(0,0,255)	#0000FF	#00F
aqua	rgb(0%,100%,100%)	rgb(0,255,255)	#00FFFF	#0FF
black	rgb(0%,0%,0%)	rgb(0,0,0)	#000000	#000
fuchsia	rgb(100%,0%,100%)	rgb(255,0,255)	#FF00FF	#F0F
gray	rgb(50%,50%,50%)	rgb(128,128,128)	#808080	
lime	rgb(0%,100%,0%)	rgb(0,255,0)	#00FF00	#0F0
maroon	rgb(50%,0%,0%)	rgb(128,0,0)	#800000	
navy	rgb(0%,0%,50%)	rgb(0,0,128)	#000080	
olive	rgb(50%,50%,0%)	rgb(128,128,0)	#808000	

Таблица 4.1 (продолжение)

Цвет	Процентные соотношения	Числовая	Шестнадцатеричная	Краткая шестнадцатеричная
purple	rgb(50%, 0%, 50%)	rgb(128, 0, 128)	#800080	
silver	rgb(75%, 75%, 75%)	rgb(192, 192, 192)	#C0C0C0	
teal	rgb(0%, 50%, 50%)	rgb(0, 128, 128)	#008080	
white	rgb(100%, 100%, 100%)	rgb(255, 255, 255)	#FFFFFF	#FFF

Цвета безопасной веб-палитры

Цвета безопасной веб-палитры («web-safe» colors) – это такие цвета, которые обычно не подлежат смешиванию в 256-цветных компьютерных системах. Цвета безопасной веб-палитры могут быть выражены RGB-кодами, кратными 20%, или 51, или шестнадцатеричному коду 33. К безопасным кодам также относятся 0% или 0. Таким образом, если RGB-цвета задаются в виде процентных соотношений, то все коды должны или содержать 0%, или быть числом, кратным 20, например rgb(40%, 100%, 80%) или rgb(60%, 0%, 0%). Если используются RGB-коды по шкале 0–255, то значения должны быть или равны 0, или кратны 51, как в rgb(0, 204, 153) или rgb(255, 0, 102).

При шестнадцатеричной форме записи любой триплет с кодами 00, 33, 66, 99, CC и FF считается принадлежащим к безопасной веб-палитре, например #669933, #00CC66 и #FF00FF. Это значит, что безопасными укороченными шестнадцатеричными кодами являются 0, 3, 6, 9, C и F; поэтому #693, #0C6 и #F0F – примеры цветов из безопасной веб-палитры.

Единицы измерения длины

Многие свойства CSS, такие как отступы, зависят от мер длины для обеспечения правильного представления различных элементов страниц. Поэтому неудивительно, что в CSS для измерения длины существует множество способов.

Все единицы измерения длины могут быть представлены как положительные или отрицательные числа (хотя некоторые свойства будут принимать только положительные значения), непосредственно за которыми следует обозначение единицы измерения. Числа могут быть и вещественными, т. е. содержать дробную часть, например 10,5 или 4,561. Все значения длины сопровождаются двухбуквенной аббревиатурой, представляющей единицы измерения, например in (дюймы) или pt (пункты). Единственное исключение из этого правила – нулевая длина (0), для которой не надо указывать единицы измерения.

Различают два типа единиц измерения длины: *абсолютные единицы измерения* (*absolute length units*) и *относительные* (*relative length units*).

Абсолютные единицы измерения длины

Начнем с абсолютных единиц измерения, потому что они проще для понимания, несмотря на тот факт, что они практически не применяются в разработке веб-страниц. Вот пять типов абсолютных единиц измерения:

Дюймы (`in`)

Как вы, возможно, поняли, это условное обозначение дюймов на линейках, с которыми имеют дело жители США. (Тот факт, что эти единицы измерения попали в спецификацию, несмотря на то, что практически весь мир пользуется метрической системой, может рассматриваться как свидетельство распространения интересов США на Интернет. Но не будем углубляться в виртуальную социально-политическую теорию.)

Сантиметры (`cm`)

Обозначает сантиметры, которые можно найти на линейках по всему миру. В одном дюйме 2,54 сантиметра, а один сантиметр равен 0,394 дюйма.

Миллиметры (`mm`)

Для тех американцев, которым тяжело привыкнуть к метрической системе, сообщаю, что в сантиметре 10 миллиметров, следовательно, один дюйм равен 25,4 миллиметра, а миллиметр равен 0,0394 дюйма.

Пункты (`pt`)

Пункты – это стандартная типографская единица измерения, которая десятилетиями используется в принтерах, наборных машинах и в программах обработки текстов. Традиционно дюйму соответствуют 72 пункта (пункты появились еще до широкого распространения метрической системы). Следовательно, 12 пунктов соответствуют одной шестой дюйма. Например, запись `p {font-size: 18pt;}` эквивалентна `p {font-size: 0.25in;}`.

Пики (`pc`)

Пики – это еще один типографский термин. Пика эквивалентна 12 пунктам, т. е. в дюйме 6 пик. А одна пика равна одной шестой доле дюйма. Например, выражение `p {font-size: 1.5pc;}` задает такую же высоту текста, как и объявления, приведенные выше.

Конечно, с этими единицами измерения удобно работать, только если броузер знает все характеристики монитора, на котором отображается страница, принтера или любого другого агента пользователя, который может быть применен. В веб-броузере представление зависит от размера монитора и разрешения, установленного на нем. И вы как автор не можете влиять на эти факторы. Вы можете только надеяться, что размеры будут хотя бы согласованными, т. е. что длина в `1.0in` будет в два раза больше `0.5in`, как показано на рис. 4.3.

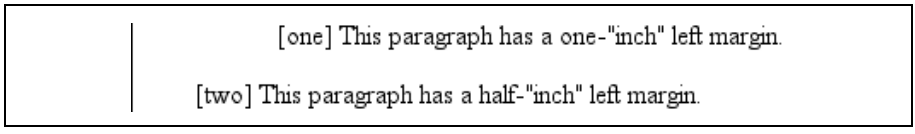


Рис. 4.3. *Задание левого отступа в абсолютных единицах длины*

Работа с абсолютными длинами

Если разрешение экрана монитора составляет 1024 пиксела в ширину и 768 пикселов в высоту, размеры экрана составляют 14,22 дюйма в ширину и 10,67 дюйма в высоту, а область отображения занимает весь экран монитора, то размеры каждого пиксела составят $^{1}/_{72}$ дюйма в ширину и высоту. Как вы, возможно, догадались, вероятность реализации этого сценария очень и очень мала (вы когда-нибудь видели монитор с такими размерами?). Таким образом, в большинстве мониторов реальное количество пикселов на дюйм (ppi – pixels per inch) превышает 72, причем иногда существенно, достигая 120 ppi и более.

В Windows драйвер видеоадаптера можно настроить таким образом, чтобы отображение элементов соответствовало реальным размерам. Для этого нажмите Пуск (Start)→Настройки (Settings)→Панель управления (Control Panel). В панели управления дважды щелкните по значку Экран (Display). Выберите вкладку Параметры (Settings) и нажмите Дополнительно (Advanced), чтобы открыть диалоговое окно (на разных ПК оно может быть разным). В разделе Размер шрифта (Font Size) выберите Другие (Other) и затем, приложив линейку к экрану, перемещайте бегунок до тех пор, пока деления на экранной линейке не совпадут с делениями на физической. Щелкайте по кнопке OK до тех пор, пока не закроете все диалоговые окна. Вот вы и произвели настройку.

В операционной системе, под управлением которой работают Макинтоши, нет возможности произвести такую настройку. В Mac Classic OS (т. е. любой версии до OS X) задано соотношение между экранными пикселами и абсолютными размерами: предполагается, что монитор имеет разрешающую способность 72 пиксела на дюйм. Это предположение совершенно неверно, но оно встроено в операционную систему и поэтому практически неустранимо. В результате во многих веб-броузерах, работающих на Classic Mac, любое заданное в пунктах значение будет эквивалентно такой же длине в пикселах: текст размером в `24pt` будет в высоту составлять 24 пиксела, и текст размером в `8pt` будет 8 пикселов в высоту. К сожалению, это слишком мало, и текст такого размера разобрать невозможно (рис. 4.4).

В OS X принято более близкое к Windows значение ppi: 96ppi. Это ничуть не более правильно, но по крайней мере приемлемо для компьютеров, работающих под Windows.

Classic Mac – замечательная иллюстрация необходимости избегать применения пунктов при разработке для Всемирной паутины. «Эмы» (em),

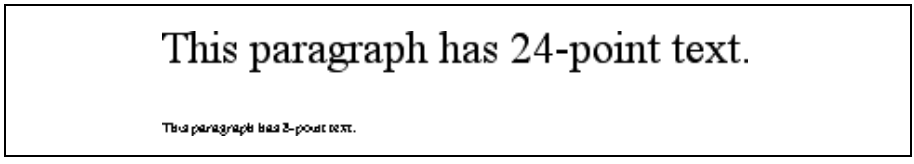


Рис. 4.4. *Очень мелкий текст очень трудно читать*

процентные соотношения и даже пикселы – все эти единицы более предпочтительны, чем пункты, когда дело касается отображения в броузере.

Начиная с Internet Explorer 5 для Macintosh и таких, основанных на технологии Gecko, броузерах, как Netscape 6+, сам броузер имеет настройки для установки значения ppi. Можно выбрать стандартное для Макинтоша соотношение 72ppi, обычное соотношение для Windows в 96ppi или значение, которое соответствует соотношению ppi вашего монитора. Последний вариант аналогичен описанной выше схеме настройки в Windows, где для сравнения с линейкой применяется подвижная шкала, благодаря чему удается получить точное соответствие размеров, отображаемых вашим монитором, и реальных физических размеров.

Теперь абстрагируемся от практических реализаций и сделаем весьма сомнительное предположение о том, что ваш компьютер знает достаточно о своей системе отображения, чтобы точно воспроизводить реальные размеры. В таком случае, объявив `p {margin-top: 0.5in;}`, можно было бы гарантировать, что верхний отступ каждого абзаца будет составлять половину дюйма. Независимо от размера шрифта или любых других обстоятельств отступ в верхней части абзаца будет равен половине дюйма.

Абсолютные единицы измерения намного удобнее при определении таблиц стилей для печатных документов, где измерения в дюймах, пунктах и пиках – обычное дело. Как вы увидели, пытаться применять абсолютные измерения в веб-разработке в лучшем случае рискованно, так что давайте вернемся к более полезным единицам измерения.

Относительные единицы измерения длины

Относительные единицы измерения получили такое название потому, что они измеряются относительно других единиц измерения. Измеряемое ими фактическое (или абсолютное) расстояние может меняться под действием внешних факторов, таких как разрешение экрана, ширина области просмотра, предпочтительные настройки пользователя и масса других параметров. Кроме того, для некоторых относительных единиц измерения их размер практически всегда измеряется относительно использующего их элемента и соответственно будет меняться от элемента к элементу.

Существует три относительных единицы измерения длины: `em`, `ex` и `px`. Первые две обозначают «em-высоту» и «x-высоту» и представляют со-

бой обычные типографские меры длины, но те, кто знаком с типографией, заметят, что в CSS эти единицы обрели неожиданный смысл. Последний тип единиц измерения длины – px, что обозначает «пикселы». Пиксел – это одна из точек, которые можно увидеть на мониторе компьютера, если внимательно присмотреться. Это значение определено как относительное, потому что зависит от разрешения устройства отображения. Данного вопроса мы скоро коснемся.

Единицы измерения em и ex

Сначала, однако, рассмотрим em и ex. В CSS один «em» – это значение свойства font-size заданного шрифта. Если для элемента font-size равен 14 пикселам, тогда для него же 1em равен 14 пикселам.

Очевидно, что это значение может меняться от элемента к элементу. Например, возьмем h1, размер шрифта которого составляет 24 пиксела, элемент h2, размер шрифта которого составляет 18 пикселов, и абзац со шрифтом в 12 пикселов. Если задать левый отступ в 1em для всех трех элементов, то отступ слева для них будет 24, 18 и 12 пикселов соответственно:

```
h1 {font-size: 24px;}
h2 {font-size: 18px;}
p {font-size: 12px;}
h1, h2, p {margin-left: 1em;}
small {font-size: 0.8em;}

<h1>Left margin = <small>24 pixels</small></h1>
<h2>Left margin = <small>18 pixels</small></h2>
<p>Left margin = <small>12 pixels</small></p>
```

При задании размера шрифта значение em вычисляется относительно размера шрифта родительского элемента, как показано на рис. 4.5.

С другой стороны, величина ex опирается на высоту буквы x нижнего регистра выбранного шрифта. Поэтому, если в двух абзацах размер текста составляет 24 пункта, но для каждого абзаца выбран свой шрифт, то значение ex для каждого из них может быть различным. Дело в том, что высота буквы «x» в разных шрифтах разная, что показано на рис. 4.6. В примерах текст имеет высоту 24 пункта, и, следовательно, значение em каждого примера составляет 24 пункта, но x-высота (высота глифов строчных букв) каждого из них разная.

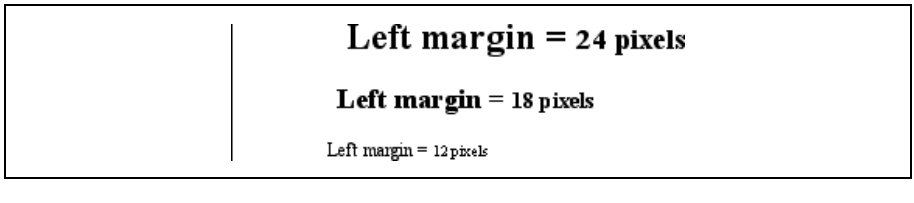


Рис. 4.5. *Применение единицы измерения длины* em *для определения размеров отступов и шрифтов*

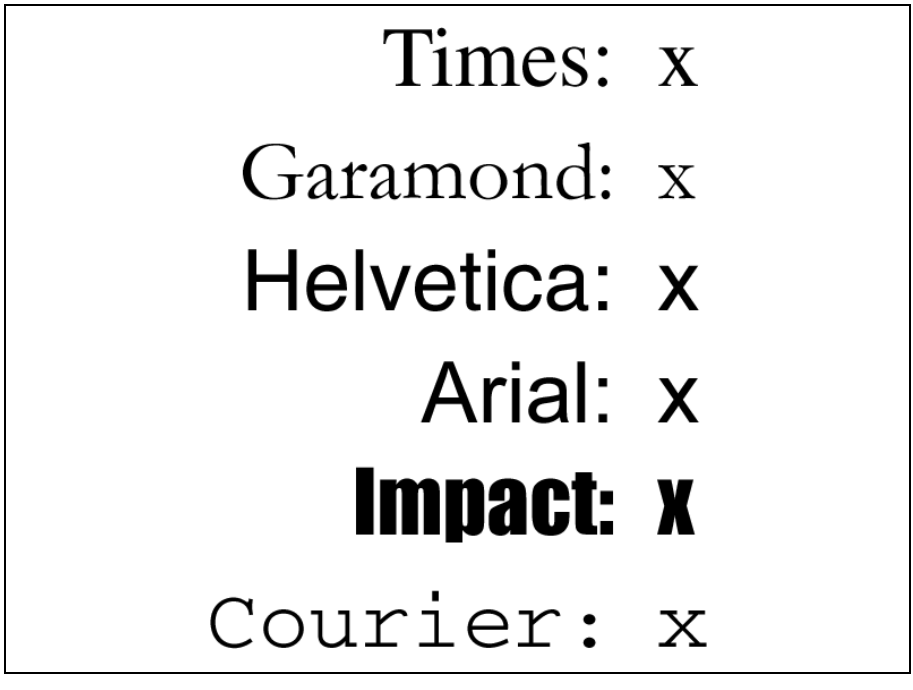


Рис. 4.6. *Разные x-высоты*

Практические трудности с применением em и ex

Конечно, все, о чем я только что говорил, сплошная теория. Я показал, что *должно* происходить, но на практике многие агенты пользователя вычисляют значение `ex` путем деления значения `em` на два. Почему? Большинство шрифтов не имеют встроенного значения высоты `ex`, и вычислить его очень сложно. Поскольку строчные буквы многих шрифтов примерно вполовину ниже прописных, удобно считать, что `1ex` эквивалентен `0.5em`.

Некоторые броузеры, включая Internet Explorer 5 для Mac OS, на самом деле пытаются определять x-высоту заданного шрифта, генерируя внутри строчную букву «x» и определяя ее высоту в пикселах, чтобы сравнить со значением свойства `font-size`, используемым для создания этого символа. Не самый идеальный метод, но это намного лучше, чем просто приравнивать `1ex` к `0.5em`. Мы, специалисты-практики CSS, можем надеяться, что со временем количество агентов пользователя, работающих с реальными значениями `ex`, возрастет и упомянутое выше упрощение с половиной `em` канет в лету.

Измерение длин в пикселах

На первый взгляд с пикселами все довольно просто. Если присмотреться к монитору достаточно пристально, то можно увидеть, что его

изображение представляет собой сетку из крошечных точек. Каждая точка – это пиксел. Определим элемент, состоящий из некоторого количества пикселов в ширину и высоту, как в следующей разметке:

```
<p>
The following image is 20 pixels tall and wide: <img src="test.gif"
  style="width: 20px; height: 20px;" alt="" />
</p>
```

Он будет занимать по высоте и ширине именно это количество точек на экране монитора (рис 4.7).

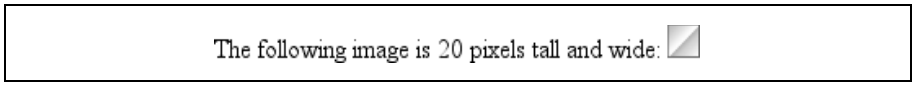


***Рис. 4.7.** Задаем длину в пикселах*

К сожалению, в применении пикселов есть потенциальный недостаток. Если размер шрифта задается в пикселах, то пользователи Internet Explorer для Windows до версии IE7 не смогут изменять размер текста с помощью команды меню Text Size своего броузера. Это неудобство может стать существенным, если текст слишком мелкий и его неудобно читать. Если выбрать более гибкие единицы измерения, например `em`, то пользователь сможет менять размер текста. (Те, кто очень трепетно относятся к своему дизайну, могут, конечно, назвать *это* недостатком.)

С другой стороны, измерения в пикселах идеальны для задания размеров изображений, которые уже представлены определенным количеством пикселов в ширину и высоту. Кстати, единственный случай, при котором окажется неудобным выражать размер изображения в пикселах, это если вам требуется масштабировать пикселы относительно некоторого текста. Этот превосходный и подчас очень полезный подход сможет предоставить реальные преимущества при работе с векторными, а не с растровыми изображениями. (С переходом к масштабируемой векторной графике Scalable Vector Graphics он получит большее распространение в будущем.)

Теория пикселов

Так почему же все-таки пикселы определены как относительные единицы измерения длины? Я говорил, что пикселы – это крошечные цветные точки на экране монитора. Но сколько таких точек в дюйме? Это может показаться нелогичным, но немного терпения.

При рассмотрении пикселов спецификация CSS рекомендует агентам пользователя в случаях, когда разрешающая способность устройства отображения значительно отличается от 96 ppi, изменить масштаб пиксела на эталонную величину. В CSS2 эталонная величина пиксела равна 90 ppi, а в CSS2.1 предлагается 96 ppi – величина, свойственная

компьютерам под управлением Windows и принятая современными броузерами для Макинтоша, такими как Safari.

В общем, если объявлено что-то вроде `font-size: 18px`, веб-броузер практически наверняка возьмет размер пикселов на мониторе (в конце концов, они уже там), но для других устройств отображения, таких как принтеры, агенту пользователя придется менять масштаб длин, выраженных в пикселах, на что-то более удобное. Иначе говоря, программа, отвечающая за вывод на печать, должна определить, сколько точек содержится в пикселе, и для этого она может взять в качестве эталона величину 96 ppi.

Один из примеров трудностей, связанных с определением размера пиксела, можно найти в ранней реализации CSS1. Internet Explorer 3.x при распечатке документа предполагал, что `18px` – это то же самое, что и 18 точек, что на принтере с разрешением 600dpi составляет 18/600 или 3/100 дюйма, или, если вам так больше нравится, `0.03in`. Для текста это совсем маленький размер!

Из-за этой потенциальной возможности масштабирования пикселы определены как относительные единицы измерения, даже несмотря на то, что в веб-разработке они ведут себя аналогично абсолютным единицам.

Что делать?

Имея в виду все связанные с этим вопросом трудности, лучше всего, вероятно, использовать относительные единицы измерения, в первую очередь, `em` и при необходимости – `px`. Поскольку `ex` в большинстве применяемых сейчас броузеров, как правило, вычисляется как часть `em`, ее достоинства в настоящий момент обсуждать не будем. Если реальное измерение x-высоты будут поддерживать большее количество агентов пользователя, возможно, `ex` возьмет свое. В общем, `em` является более гибкой единицей измерения, потому что соотнесена с размером шрифта, так что относительное расположение элементов будет оставаться постоянным.

В ряде случаев при работе с элементами удобнее иметь дело с пикселами, например, для задания ширины рамки или позиционирования элементов. Все зависит от ситуации. Например, в проектах, где по традиции фрагменты дизайна друг от друга отделяются при помощи GIF-изображений, применение отступов, длина которых выражена в пикселах, имеет аналогичный результат. При преобразовании этих величин в `em` они бы увеличивались или уменьшались в соответствии с изменениями размера текста, что иногда хорошо, а иногда – нет.

URL

Те, у кого есть опыт создания веб-страниц, наверняка хорошо знакомы с URL (или в CSS2.1 – URI). Общий формат обращения к нужному объ-

екту, например, в операторе `@import`, применяемом при импортировании внешней таблицы стилей, таков:

```
url(protocol://server/pathname)
```

Приведенный выше пример определяет то, что известно как *абсолютный URL* (*absolute URL*). Под абсолютным я подразумеваю URL, который будет работать везде, независимо от того, где (или вернее, на какой странице) он находится, потому что он определяет абсолютное месторасположение в веб-пространстве. Скажем, есть сервер *www.waffles.org*. На этом сервере есть каталог под названием `pix`, и в этом каталоге есть файл изображения *waffle22.gif*. В этом случае абсолютный URL этого файла будет записан так:

```
http://www.waffles.org/pix/waffle22.gif
```

Этот URL будет действительным независимо от того, располагается ли содержащая данный URL страница на сервере *www.waffles.org* или *web.pancakes.com*.

Другой тип URL – *относительный* URL, названный так потому, что он определяет местоположение относительно содержащего этот URL документа. Если вы ссылаетесь на относительное местоположение, например файл, находящийся в том же каталоге, что и ваша веб-страница, тогда общий формат такой:

```
url(pathname)
```

Эта форма записи годится, только если файл находится на том же сервере, что и страница, содержащая URL. Для примера представим себе, что веб-страница расположена по адресу *http://www.waffles.org/syrup.html*, и на этой странице необходимо показать изображение *waffle22.gif*. Для этого случая URL будет таким:

```
pix/waffle22.gif
```

Этот путь действителен, потому что веб-броузер знает, что он должен начать поиск с места, где находится веб-документ, и затем добавить относительный URL. В этом случае путь *pix/waffle22.gif* добавляется к имени сервера *http://www.waffles.org/*, что аналогично *http://www.waffles.org/pix/waffle22.gif*. Практически во всех ситуациях можно применять абсолютный URL вместо относительного. Не имеет значения, какой из URL выбран, если он правильно определяет местоположение.

В CSS относительные URL определяются относительно положения таблицы стилей, а не HTML-документа, который ее использует. Например, у вас есть внешняя таблица стилей, которая импортирует другую таблицу стилей. Если для импорта второй таблицы стилей вы используете относительный URL, он должен быть задан относительно первой таблицы стилей. В качестве примера рассмотрим HTML-документ, находящийся по адресу *http://www.waffles.org/toppings/tips.html*, в котором есть ссылка на таблицу стилей *http://www.waffles.org/styles/basic.css*:

```
<link rel="stylesheet" type="text/css"
    href="http://www.waffles.org/styles/basic.css">
```

В файле *basic.css* находится инструкция `@import`, ссылающаяся на другую таблицу стилей:

```
@import url(special/toppings.css);
```

Эта инструкция `@import` заставит броузер искать таблицу стилей по адресу *http://www.waffles.org/styles/special/toppings.css*, а не *http://www.waffles.org/toppings/special/toppings.css*. Если таблица стилей расположена по второму адресу, то директива `@import` в файле *basic.css* должна выглядеть так:

```
@import url(http://www.waffles.org/toppings/special/toppings.css);
```

Netscape Navigator 4 интерпретирует относительные URL относительно HTML-документа, а не таблицы стилей. Если на вашу веб-страницу заходит много посетителей с NN4.x или вы хотите гарантировать, что NN4.x сможет найти все ваши таблицы стилей и фоновые изображения, проще сделать все URL абсолютными, поскольку их Navigator обрабатывает правильно.

Обратите внимание, что между `url` и открывающей круглой скобкой не должно быть пробела:

```
body {background: url(http://www.pix.web/picture1.jpg);}   /* верно */
body {background: url  (images/picture2.jpg);}           /* НЕВЕРНО */
```

Если вставить здесь пробел, все объявление будет признано недействительным и, следовательно, проигнорировано.

Ключевые слова

Ключевые слова существуют с тех пор, как в качестве значений используются некоторые определенные слова. Очень распространенный пример – ключевое слово `none`, отличающееся от `0` (нуля). Так, чтобы удалить подчеркивание ссылок в HTML-документе, можно написать:

```
a:link, a:visited {text-decoration: none;}
```

Аналогично, если бы потребовалось подчеркнуть ссылки, можно было бы указать ключевое слово `underline`.

Если свойство допускает применение ключевых слов, то его ключевые слова определены только для этого свойства. Если одно и то же слово задано как ключевое для двух свойств, то действие ключевого слова для одного свойства никак не будет связано с его действием в рамках другого свойства. В качестве примера возьмем слово `normal`. Определенное для свойства `letter-spacing`, оно означает нечто совершенно отличное от того, что задает `normal` для свойства `font-style`.

Ключевое слово inherit

Существует одно ключевое слово, которое используется всеми свойствами CSS2.1: `inherit`. Оно делает значение свойства таким же, как и у его родительского элемента. В большинстве случаев не надо задавать наследование, потому что большая часть свойств реализует его по умолчанию; однако `inherit` может быть крайне полезным.

Например, рассмотрим следующие стили и разметку:

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

<div id="toolbar">
<a href="one.html">Один</a> | <a href="two.html">Два</a> |
<a href="three.html">Три</a>
</div>
```

Сам элемент `div` будет белым на синем фоне, а вот ссылки будут оформлены согласно предпочтительным настройкам броузера. Скорее всего, это будет синий текст на синем фоне с белыми вертикальными полосками между ними.

Можно было бы написать правило, явно задающее белый цвет текста ссылок панели инструментов, но можно обратиться к более надежному способу и применить `inherit`. Для этого достаточно добавить в таблицу стилей следующее правило:

```
#toolbar a {color: inherit;}
```

Это заставит ссылки руководствоваться унаследованным значением `color` вместо применяемых по умолчанию стилей агента пользователя. Обычно непосредственно назначенные стили замещают унаследованные стили, но `inherit` может изменить это поведение на обратное.

Единицы измерения CSS2

В добавление к рассмотренным в рамках CSS2.1, рассмотрим несколько дополнительных единиц измерений, которые содержит CSS2. Все они имеют отношение к звуковым таблицам стилей (применяемым теми броузерами, которые могут воспроизводить речевой сигнал). Эти единицы измерения не были включены в CSS2.1, но поскольку они могут стать частью будущих версий CSS, мы кратко обсудим их здесь:

Угловые величины

Предназначены для определения местоположения, из которого должен исходить данный звук. Существует три типа единиц измерения углов: градусы (`deg`), грады (`grad`) и радианы (`rad`). Например, прямой угол может быть задан как `90deg`, `100grad` или `1.57rad`; в любом случае значения переводятся в градусы в диапазоне от 0 до 360. Это также выполняется и для отрицательных значений, которые допускаются. Величина `-90deg` эквивалентна `270deg`.

Значения времени

Определяют задержки между элементами речи. Они могут быть выражены или в миллисекундах (`ms`), или в секундах (`s`). Таким образом, значения `100ms` и `0.1s` эквивалентны. Значения времени не могут быть отрицательными, поскольку CSS не предполагает создания парадоксов.

Значения частот

Служат для задания частоты звуков, генерируемых броузером с возможностью синтеза речи. Значения частот могут быть выражены в герцах (`Hz`) или мегагерцах (`MHz`) и не могут быть отрицательными. Сокращения единиц измерения частот нечувствительны к регистру, поэтому записи `10MHz` и `10mhz` эквивалентны.

На момент написания данной книги единственным известным агентом пользователя, поддерживающим эти значения, является реализация звуковых таблиц стилей *Emacspeak*. Звуковые стили подробно рассмотрены в главе 14.

Кроме этих значений существуют также старые добрые друзья под новыми именами. *URI* (*Uniform Resource Identifier – универсальный идентификатор ресурса*) является разновидностью другого имени – URL (Uniform Resource Locator – универсальный адрес ресурса). Спецификации CSS2 и CSS2.1 требуют, чтобы URI объявлялись в форме `url(...)`, так что в этом практически ничего не изменилось.

Заключение

Единицы измерения и значения охватывают очень широкий спектр понятий: от единиц измерения длин до специальных ключевых слов, которые описывают эффекты (такие как `underline` (подчеркивать)), от названий цветов до местоположений файлов (таких как изображения). Большей частью, единицы измерения – это та область, которую агенты пользователя обрабатывают практически полностью правильно, за исключением нескольких небольших ошибок и индивидуальных особенностей. Например, неспособность Navigator 4.x правильно интерпретировать относительные URL измучила многих авторов и привела к повсеместному переходу к абсолютным URL. Цвета – это еще одна сфера, где агенты пользователя практически всегда ведут себя хорошо, кроме нескольких небольших случайностей, происходящих то тут, то там. Однако причуды единиц измерения длин далеки от того, чтобы считать их простыми ошибками, это интересная проблема, над которой должен потрудиться каждый автор.

Все эти единицы измерения имеют свои преимущества и недостатки в зависимости от обстоятельств, в которых они применяются. Мы уже видели некоторые из этих обстоятельств, а нюансы будут обсуждаться далее, начиная с CSS-свойств, которые определяют способы изменения отображения текста.