Роман Клименко

на 100%

ВЕБ-МАСТЕРИНГ:

Изучаем HTML5, CSS3, JavaScript, PHP, CMS, AJAX, SEO

Эта книга поможет вам:

- Освоить новейшие веб-технологии
- Создавать динамические сайты
- Научиться эффектным технологиям JavaScript
- Сопровождать и продвигать сайты в Интернете

ПИТЕР

ББК 32.988.02-018
УДК 004.72
К49

Клименко Р. А.

К49 Веб-мастеринг на 100%. — СПб.: Питер, 2013. — 512 с.: ил.

ISBN 978-5-496-00079-6

Данная книга предназначена для тех, кто хочет научиться веб-мастерингу и стать специалистом по созданию веб-сайтов на профессиональном уровне. В издании описываются самые популярные и востребованные веб-технологии — HTML5, CSS3, JavaScript, jQuery, Ajax, PHP, а также приемы работы с системой управления содержимым сайта CMS Drupal и секреты раскрутки сайта (SEO). С помощью этих средств вы сможете создавать сайты любого назначения: от «визиток» и блогов до интерактивных интернет-магазинов и порталов с непрерывно обновляемыми новостями. Прочитав эту книгу, вы станете настоящим веб-мастером, готовым к работе над любыми проектами на 100%.

ББК 32.988.02-018
УДК 004.72

ISBN 978-5-496-00079-6

Краткое содержание

Оглавление

Введение

Лет десять назад создание сайтов было несложным занятием. Достаточно было потратить несколько вечеров на изучение базовых возможностей HTML, и дело было сделано — вы могли смело называть себя веб-мастером. Однако за последние десять лет в сайтостроении многое изменилось. Сайты стали намного красивее и интерактивнее. Возможности разработчиков возросли. Но вместе с тем сложность создания сайтов возросла во много раз. Прогресс не стоит на месте, и количество знаний, которыми должен обладать разработчик, увеличилось многократно.

Теперь для разработки полноценного сайта недостаточно знать HTML. Чтобы создать красивый и функциональный сайт, нужно разбираться в таких технологиях, как CSS, JavaScript, PHP. Необходимо уметь работать с графическим пакетом Adobe Photoshop и желательно иметь навыки создания баннеров и flash-объектов. А чтобы не провести остаток жизни за созданием своего сайта, желательно также уметь работать с какой-либо CMS (Content Management System) — системой управления сайтом.

Чтобы досконально изучить все перечисленные технологии, потребуется не один месяц и не одна книга. Но практика показывает, что дотошно изучать эти технологии не нужно. HTML, CSS, JavaScript, PHP... — в своей повседневной жизни рядовой веб-программист использует лишь малую часть всех их возможностей. Более того, на свете есть очень и очень мало людей, которые наизусть помнят все возможности тех же HTML и CSS.

В современном мире профессионализм веб-разработчика оценивается не количеством его навыков, а умением выходить из нестандартных ситуаций, знанием изюминок программирования. Что толку от того, что вы можете без запинки назвать все теги языка HTML, если вы не способны реализовать вывод трех колонок макета сайта, одинаково отображающийся в популярных браузерах. Какой смысл от всех ваших познаний в CSS, если вы не можете воплотить градиентный текст, задуманный дизайнером в макете.

Данная книга не заменит десятка книг с полным описанием синтаксиса и команд языков HTML, JavaScript и PHP. Но она поможет вам стать профессиональным веб-разработчиком. С ее помощью вы изучите все, что необходимо в повседневной работе веб-разработчика. Вы овладеете тонкостями разработки сайта: правильным позиционированием объектов при верстке, размещением элементов дизайна внизу экрана, работой с нестандартными шрифтами, созданием теней и градиентного текста, закруглением уголков элементов HTML-макета и многим другим.

После изучения основных технологий, применяемых веб-разработчиками при создании сайта, мы рассмотрим, как работать с CMS Drupal: создадим тему оформления для данной CMS на основе верстки, выполненной нами в предыдущих главах. После чего разработаем настоящий сайт, и даже полноценный интернет-магазин.

В конце книги мы поговорим о раскрутке созданного сайта в поисковых системах.

На странице книги на сайте издательства «Питер» (http://www.piter.com) вы можете скачать дополнительные материалы к книге.

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты halickaya@minsk.piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

Глава 1

HTML

Подготавливаем шаблон

Структура документа HTML

Работаем с макетом

Контейнеры

Работаем с текстом

Изображения и ссылки

Списки

Таблицы

Форма и ее элементы

Используем Flash

Нововведения HTML5

Первый язык, который мы с вами изучим, называется HTML (HyperText Markup Language, язык разметки гипертекста). Он дает возможность указать, в какой части веб-страницы будет находиться тот или иной элемент: текст, таблица, картинка.

Раньше HTML применялся для форматирования текста и изменения цвета элементов страницы. Но сейчас для этих целей служат каскадные таблицы стилей (Cascading Style Sheets, CSS), о которых будет рассказано в главе 2.

После прочтения этой и следующей глав вы сможете называть себя верстальщиком сайтов, то есть человеком, выполняющим верстку HTML-документа.

Верстка — это создание структуры HTML-документа таким образом, чтобы во всех браузерах веб-страница выглядела точно так же, как и на предоставленном дизайнером макете.

Макет — разработанный в формате PSD вид будущей HTML-страницы, определяющий расположение элементов и изображения, которые должны на ней отображаться. Реже макеты создают в векторных форматах AI и CDR. Еще реже в качестве макета выступает JPG-картинка или PDF-документ.

Подготавливаем шаблон

Самое сложное в любом деле — начать. При разработке сайта начало всегда одинаковое — нужно создать текстовый документ с расширением HTML, HTM или PHP, после чего внести в него общий каркас HTML-документа. Ведь HTML-документ — это не что иное, как обычный текстовый документ, но только со специальным расширением. Вы легко можете создать его в любом текстовом редакторе, даже в Блокноте.

Расширение HTML-документа

Между расширениями HTML и HTM нет разницы, поэтому вы можете использовать любое из них. А вот расширение PHP можно применять только в том случае, если вы создаете сайт на каком-либо хостинге (локальном или хостинге в Интернете) с установленным языком PHP.

Расширение PHP мы будем применять после главы 4, посвященной языку PHP. До этого все наши веб-документы будут иметь расширение HTML.

Имя HTML-документа

В принципе, HTML-документ может иметь любое имя, состоящее из латинских букв и цифр (но первой в имени должна идти буква). Но если вы хотите, чтобы HTML-документ открывался при вводе в адресной строке браузера адреса вашего сайта, нужно дать HTML-документу специальное имя.

Например, вы приобрели домен example.com, после чего разместили на нем набор HTML-документов. Какой HTML-документ будет открываться, если посетитель введет в адресной строке браузера URL-адрес http://example.com (то есть адрес сайта без указания документа, который хочет просмотреть)?

В этом случае веб-сервер попытается открыть HTML-документ с именем index.html или main.html. Чаще всего самый главный HTML-документ называют index.html.

Вы, наверное, знакомы с таким понятием, как «домашняя страница». На многих сайтах есть логотип со ссылкой на нее, или «хлебные крошки»[1], которые начинаются с нее. Как правило, домашняя страница — это и есть файл index.html, index.htm или index.php.

Итак, создайте текстовый документ с расширением HTML.

Версии HTML

Первая версия языка HTML — HTML 2.0 — была утверждена еще в 1995 году. Процесс создания сайтов на ней не подчинялся никаким правилам — разработчики могли использовать любой регистр, и сайты на HTML 2.0 грешили огромным количеством ошибок.

В 1997 году появились сразу две новые версии языка HTML — 3.2 и 4.

Еще через два года, в 1999 году, возникла версия HTML 4.01, содержащая большое количество нововведений.

А на следующий год была выпущена версия XHTML 1.0, которая долгое время использовалась для создания сайтов. И лишь недавно появилась HTML5 — самая последняя версия языка разметки. Версия HTML5 является попыткой объединить XHTML 1.0 и HTML 4.01.

Чем же отличались старые версии HTML от более новых? Естественно, в новых версиях появлялись новые теги и атрибуты. Однако это не все изменения.

---

[1] Навигационная цепочка, в которой перечислены ссылки на все разделы сайта, начиная с главной страницы, которые нужно было последовательно посетить, чтобы попасть на открытую в данный момент страницу сайта.

Иногда использовавшиеся ранее теги или атрибуты признавались устаревшими и не рекомендуемыми для применения. От этого теги/атрибуты не теряли своей функциональности — иначе пострадала бы обратная совместимость. Но применять их не рекомендовалось.

Чаще всего теги и атрибуты признавались устаревшими по той причине, что для них появилась более подходящая замена. Так стали устаревшими атрибуты изменения цвета элемента, а также тег `FONT` — на их место пришли каскадные таблицы стилей. Но изредка замены для устаревшего тега не находилось. Например, устаревший тег `CENTER` еще долгое время жил за гранью закона — разработчики вынуждены были его использовать, не находя адекватной замены.

Иногда теги, которые ранее были признаны устаревшими, в новой версии HTML опять становились рекомендуемыми для применения. Как и любая развивающаяся система, язык HTML старался соответствовать текущему времени и текущим нуждам разработчиков.

Но основная тенденция развития языка HTML заключалась в стандартизации синтаксиса и верстки HTML-документа. Из-за многочисленных условностей при верстке допускалось большое количество ошибок. Было принято решение бороться с этим. В результате синтаксис языка HTML стал более строгим. Но об этом мы поговорим далее в книге.

Вернемся к нашим дням. На данный момент последняя версия HTML имеет номер 5. HTML5 отличается от XHTML 1.0 только наличием новых тегов. Причем эти теги не поддерживаются старыми версиями браузеров. Под старыми версиями имеются в виду версии браузеров, которые были выпущены год и более назад. В Интернете существует огромное количество пользователей, которые до сих пор используют старые версии браузеров. По этой причине новые теги HTML5 пока лучше не применять. А если учесть, что абсолютное большинство заказчиков еще в 2010 году требовало от разработчиков сайтов совместимости верстки с браузером Internet Explorer 6.0 (то есть чтобы в устаревшем браузере Internet Explorer 6.0 верстка отображалась точно так же, как и в более новых версиях браузера), то время возможностей HTML5 наступит еще очень нескоро.

Именно поэтому в данной книге мы начнем изучение HTML с возможностей языка XHTML. И только в последних главах начнем применять возможности HTML5.

Выбираем DOCTYPE

Каждый HTML-документ должен начинаться со строки `DOCTYPE`. Она говорит браузеру, какую версию HTML вы планируете использовать при создании HTML-страницы.

Это очень важный момент, поскольку строка DOCTYPE влияет на большое количество мелочей: от поведения различных тегов до размера границ, расстояния по умолчанию, высоты и ширины тегов. Поэтому указывать строку DOCTYPE следует осознанно. Но поскольку мы с вами только начали изучение языка HTML, нам остается лишь выбрать DOCTYPE последней версии HTML и учиться верстать для данной версии.

Если вы не укажете строку DOCTYPE или сделаете это неверно, различные браузеры будут вести себя по-разному. Браузер Opera будет считать, что вы верстаете в самой новой версии HTML, а браузеры Internet Explorer и Mozilla, наоборот, — что для самой старой версии. Соответственно, ваша веб-страница будет по-разному выглядеть в разных браузерах, что категорически недопустимо.

Итак, откройте в Блокноте HTML-файл, который мы создали ранее. И в первой строке этого файла введите строку:

```
<!DOCTYPE HTML>
```

Это DOCTYPE для HTML5. Именно его мы и будем использовать в дальнейшем.

Помимо данного DOCTYPE, можно встретить следующие:

- <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd"> — DOCTYPE для стандарта HTML 4.01 Strict (строгий);
- <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> — DOCTYPE для стандарта HTML 4.01 Transitional (переходный);
- <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd"> — DOCTYPE для стандарта HTML 4.01 Frameset (с фреймами);
- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> — DOCTYPE для стандарта XHTML 1.0 Strict (строгий);
- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> — DOCTYPE для стандарта XHTML 1.0 Transitional (переходный);
- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd"> — DOCTYPE для стандарта XHTML 1.0 Frameset (с фреймами);
- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"> — DOCTYPE для стандарта XHTML 1.1.

Как видите, запись DOCTYPE для HTML5 самая простая: в ней нет никаких непонятных адресов. И для него, как и для XHTML 1.1, нет разновидностей Transitional, Frameset и Strict.

От Transitional, Frameset и Strict также зависит, как будут выводиться те или иные элементы языка HTML, а также какие из этих элементов будут считаться устаревшими.

- Разновидность Transitional, как следует из названия, является переходным вариантом от прошлой версии языка HTML к новой. Если в новой версии HTML какие-то теги были признаны устаревшими, то в стандарте Transitional они по-прежнему остаются рекомендуемыми для применения. То есть при использовании варианта Transitional сайт, скорее всего, будет выводиться так же, как и при применении более старой версии HTML.
- Разновидность Frameset аналогична Transitional, кроме того, в Frameset также разрешено применять теги для создания фреймов.
- Разновидность Strict используется в том случае, если код документа полностью соответствует выбранной версии HTML. В Strict оформление и содержание полностью разделены между HTML и CSS.

Если по каким-то причинам вы не можете применить <!DOCTYPE HTML>, то лучше всего использовать DOCTYPE предыдущей версии HTML: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">.

Запоминать, как пишется DOCTYPE, не стоит. Лучше создать шаблон HTML-документа с нужным DOCTYPE, после чего использовать этот шаблон в своих проектах.

Подготавливаем HTML-шаблон

После того как вы определились с нужным DOCTYPE, следует составить базовую структуру HTML-документа. В HTML все теги представляют собой иерархическую систему. Они могут быть вложены только в определенные теги.

Так, самым первым тегом в HTML-документе должен быть HTML, объявление которого идет сразу после DOCTYPE. В этом теге могут находиться только теги HEAD или BODY. Сам документ должен располагаться внутри тега BODY, а информация, описывающая документ, — в теге HEAD.

Если вы не создаете страницу с фреймами, то базовая структура всегда будет одинакова. Для HTML5 базовая структура HTML-документа представлена в листинге 1.1.

Листинг 1.1. Базовая структура HTML-документа версии HTML5

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
</head>
<body>
</body>
</html>
```

Для XHTML базовая структура отличается наличием других атрибутов (листинг 1.2).

Листинг 1.2. Базовая структура HTML-документа версии XHTML 1.0 Strict

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
</head>
<body>
</body>
</html>
```

Сохраняем HTML-документ

Есть одна особенность, о которой не следует забывать, если вы создаете HTML-документ в кодировке UTF-8, то есть если вы указали в коде следующий тег:

```
<meta http-equiv="content-type" content="text/html;
charset=UTF-8" />
```

Если ваша HTML-страница должна работать в кодировке UTF-8, то и HTML-документ нужно сохранить в этой кодировке. Для этого в Блокноте выберите пункт меню Файл ▸ Сохранить как и в раскрывающемся списке Кодировка диалога Сохранить как укажите пункт UTF-8 (рис. 1.1).

Если вы этого не сделаете, то русские символы в вашем документе при просмотре через браузер будут отображаться в виде «кракозябров».

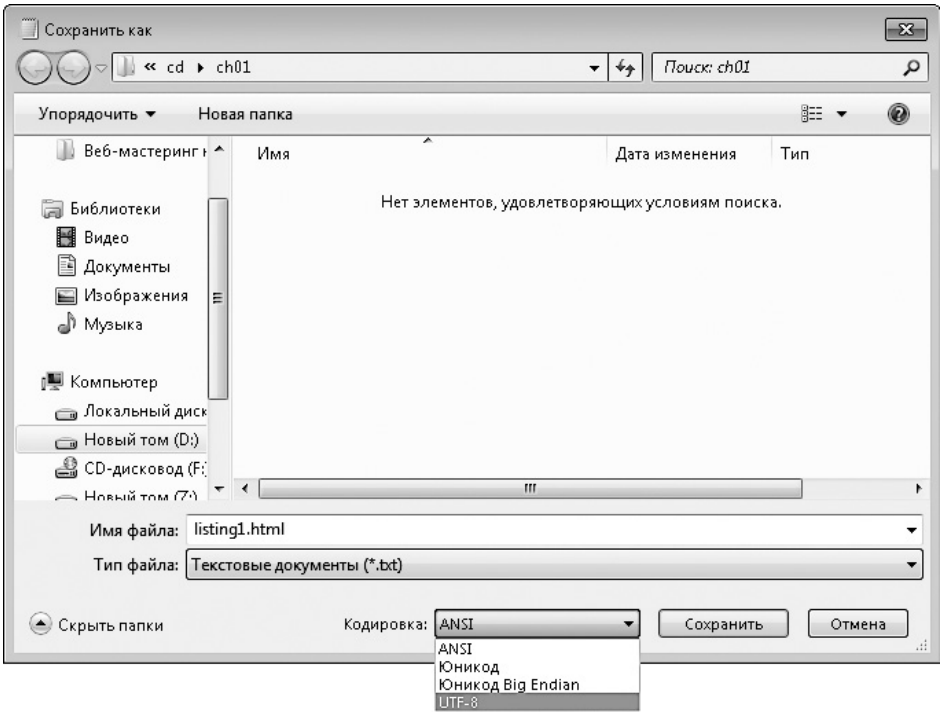


Рис. 1.1. Сохраняем HTML-документ в кодировке UTF-8

Просматриваем наш документ

После того как HTML-документ сохранен, его можно просмотреть в любом доступном браузере (рис. 1.2). Если вы верстаете на заказ, проверять правильное отображение веб-страницы следует во всех популярных браузерах: Opera, Mozilla Firefox, Safari, Google Chrome и Internet Explorer версий 6–9, поскольку одна и та же страница может выглядеть по-разному, в зависимости от браузера, в котором вы ее просматриваете.

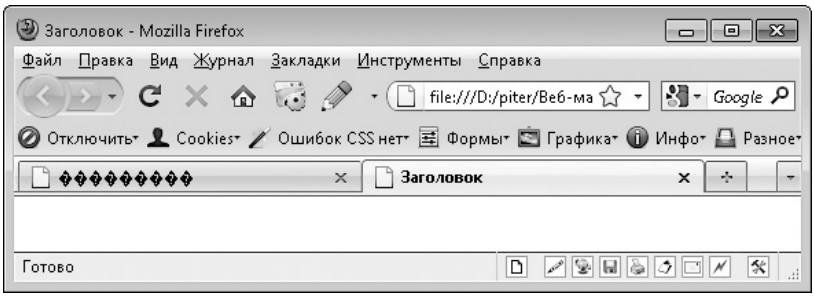


Рис. 1.2. Браузер с двумя открытыми HTML-документами: второй документ был сохранен в кодировке UTF-8, а для первого выбор кодировки при сохранении не был сделан

ПРИМЕЧАНИЕ

В приложении 2 приведена информация о том, какие теги работают в популярных версиях браузеров.

Сайты с браузерами для Windows. Скачать версии популярных браузеров можно по следующим ссылкам:

- Opera — http://www.opera.com/download/;
- Safari — http://www.apple.com/ru/safari/download/;
- Google Chrome — http://www.google.ru/chrome;
- Mozilla Firefox — http://www.mozilla.org/ru/firefox/new/;
- Internet Explorer — http://windows.microsoft.com/ru-RU/internet-explorer/downloads/ie.

Internet Explorer. Поскольку этот браузер является наиболее популярным, верстальщику приходится проверять корректное отображение веб-страницы не только в последней его версии, но и в более ранних, вплоть до версии 6.

Однако на одном компьютере может быть установлена только одна версия браузера Internet Explorer. По этой причине поступают следующим образом. На компьютер устанавливают последнюю версию браузера Internet Explorer (на момент написания книги это версия 9), после чего дополнительно устанавливают специальную программу, содержащую более ранние версии браузера Internet Explorer и позволяющую запускать эти версии. Наиболее популярна бесплатная программа Utilu IE Collection, скачать которую можно со страницы http://finalbuilds.edskes.net/iecollection.htm.

Во время инсталляции программы Utilu IE Collection выберите браузеры Internet Explorer 6 и выше. Более ранние версии браузера можно не устанавливать.

Браузер для других ОС. Выше мы рассмотрели только популярные браузеры для ОС Windows. А ведь есть и другие операционные системы, в которых используются либо совершенно другие браузеры, либо те же браузеры, но отображающие вебстраницу совершенно по-другому.

К счастью, обычно заказчики не просят проверять корректное отображение вебстраницы во всех популярных браузерах для всех операционных систем. Если вам попадется «неправильный» заказчик, то на помощь вам могут прийти специальные онлайн-сервисы, позволяющие увидеть, как смотрится страница в различных версиях многих браузеров для разных операционных систем.

Примером такого сервиса является сайт http://browsershots.org. Помимо всего прочего, перед созданием скриншотов он позволяет выбрать разрешение экрана, глубину цвета, использование JavaScript и Flash в браузере.

Конечно, вместо использования подобных сервисов вы можете установить у себя на компьютере не только операционную систему Windows, но и другие популярные операционные системы: Linux и Mac OS.

Какой браузер выбрать. Как правило, веб-страницы создают в одном из популярных браузеров, время от времени проверяя их корректное отображение во всех остальных браузерах. И может возникнуть вопрос: в каком из браузеров лучше всего создавать веб-страницы. Конечно, это вопрос: предпочтений. Но на мой взгляд, идеальным для разработки сайтов является браузер Mozilla Firefox. Для такой оценки есть следующие причины:

- наличие консоли ошибок (Инструменты ▸ Консоль ошибок), позволяющей сразу увидеть проблемы в CSS- и JavaScript-коде открытой страницы;
- возможность просмотреть исходный код не только всей страницы, но и любой ее части;
- возможность быстро изучить список всех изображений, используемых на странице, а также сохранить любое из них (Инструменты ▸ Информация о странице, вкладка Мультимедиа);
- наличие расширения Web Developer для браузера Mozilla Firefox (это расширение мы рассмотрим далее в книге).

Редакторы для верстки

Конечно, сверстать HTML-документ можно и в Блокноте... Раньше это считалось верхом профессионализма. Сейчас же это считается верхом глупости.

Для разработки HTML-документов существует множество специализированных программ. Основное достоинство таких программ — они подсвечивают HTML-код и позволяют сразу определить, где в коде находится ошибка. Кроме того, многие подобные программы содержат интерактивную справку, позволяющую быстро освежить в своей памяти назначение того или иного тега/атрибута/параметра. Это позволяет минимизировать количество ошибок в HTML-документе и упростить его создание. Поэтому использовать для создания HTML-документа Блокнот крайне не рекомендуется.

Также категорически запрещается использовать для верстки программу Microsoft Word и другие текстовые процессоры. Подобные программы идеально подходят для создания текстовых документов, но совершенно не годятся для HTML-документов. Дело в том, что текстовые процессоры автоматически производят множество функций, полезных при создании обычного текстового документа. Например, они автоматически заменяют одни кавычки на другие. И уже это приведет к неработоспособности кода.

Рассмотрим некоторые редакторы для верстки.

Notepad++. Популярной бесплатной заменой Блокнота является программа Notepad++, которую можно скачать с сайта http://notepad-plus-plus.org/. Одна из функций данной программы — подсвечивание синтаксиса различных языков программирования, в том числе HTML, CSS, JavaScript, PHP (рис. 1.3). Благодаря этому программа Notepad++ может быть хорошей альтернативой Блокноту.

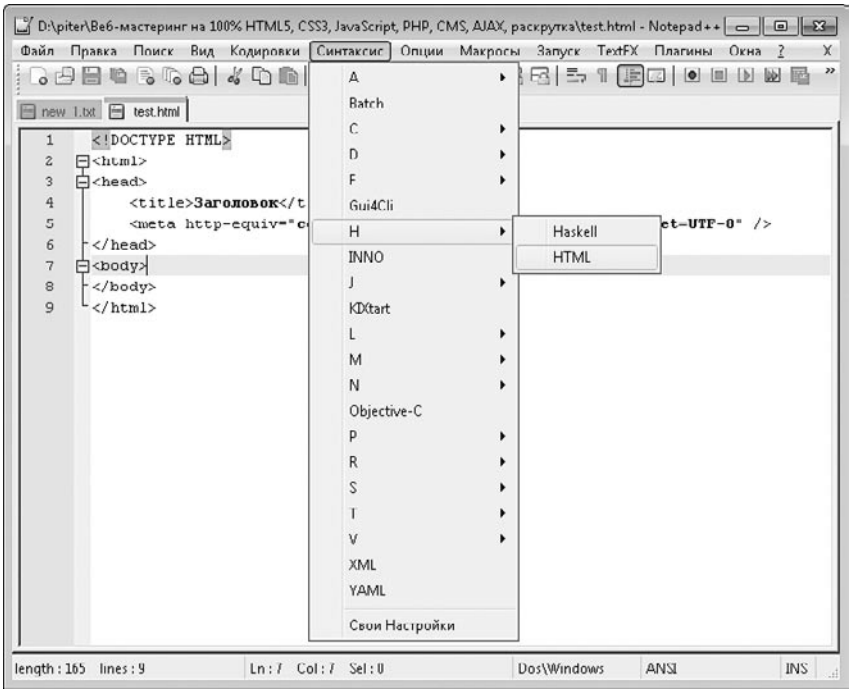


Рис. 1.3. Программа Notepad++

phpDesignerPro. Если вы решили заняться версткой профессионально, вам понадобится что-то более функциональное, чем Notepad++. А именно — специализированная программа для верстки и программирования. К сожалению, такие приложения платные. Но они стоят тех денег, которые за них просят.

В качестве примера рассмотрим возможности платной программы phpDesignerPro, которую можно скачать с сайта http://www.mpsoftware.com. Ознакомительная версия данной программы работает 21 день.

Это приложение содержит шаблоны самых разных документов, в том числе HTML-документа (рис. 1. 4). Поэтому вам больше не нужно будет хранить сделанный нами ранее шаблон HTML-документа. Достаточно просто создать в программе новый HTML-документ.

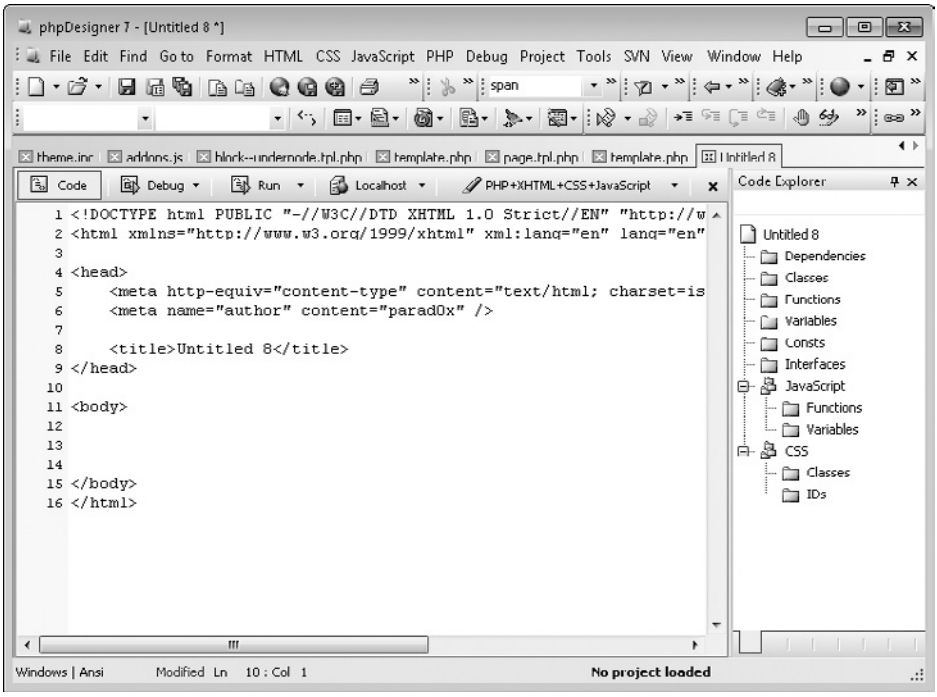


Рис. 1.4. Создание HTML-документа начинается со специального шаблона

Программа phpDesignerPro умеет подсвечивать синтаксис таких языков программирования, как PHP, XHTML, HTML5, CSS3, JavaScript, XML, Perl, VBScript, SQL, Java, и других, с которыми верстальщик и разработчик сайтов никогда не столкнутся.

При этом синтаксис не только подсвечивается. Программа умеет также автоматически дополнять незаконченные теги, атрибуты, свойства и параметры CSS, а также функции PHP и jQuery (рис. 1.5).

Функция автоматического дополнения связана с функцией подсказки (рис. 1.6). Достаточно выбрать один из пунктов в списке автодополнения, и для него отобразится краткое описание, список входящих параметров и тип возвращаемого значения. Данная функция работает с языком программирования PHP, о котором вы узнаете далее.

Благодаря встроенному визуальному режиму вы можете сразу увидеть, как выглядит ваш HTML-документ в различных популярных браузерах (рис. 1.7). Сами браузеры также должны быть установлены на вашем компьютере. И в настройках программы phpDesignerPro должны быть указаны правильные пути к этим браузерам на компьютере.

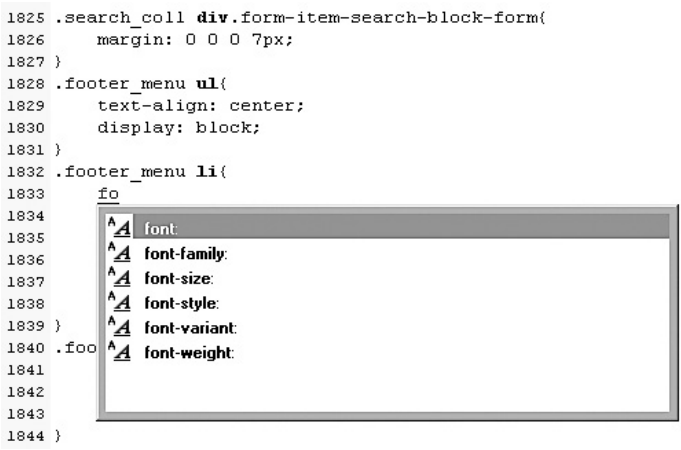


Рис. 1.5. Автоматическое дополнение параметров

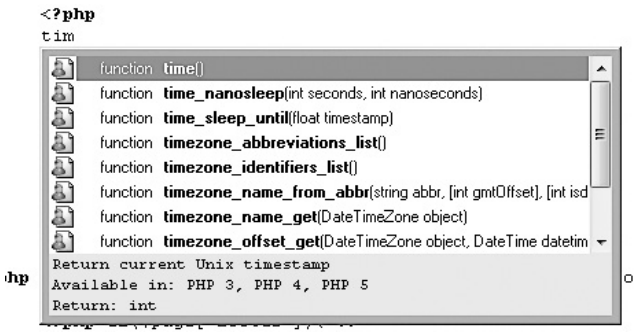


Рис. 1.6. Функция подсказки

Это лишь небольшая часть возможностей программы phpDesignerPro. Данная программа содержит огромное количество функций, вплоть до отладчика и FTP-клиента, автоматически загружающего HTML-документ после редактирования на выбранный сервер. Но все эти возможности не имеет смысла рассматривать в начале книги.

Другие программы. Будущий разработчик сайтов может также остановить свой выбор на следующих текстовых и визуальных редакторах HTML:

- Adobe Dreamweaver (http://www.adobe.com/cfusion/tdrc/index.cfm?product=dreamweaver) — визуальный редактор для создания веб-сайтов и приложений;
- Hotdog (http://www.sausage.com/hotdog-professional.html) — данная программа поможет создать не только HTML-документ, но и программировать на Flash, PHP, ASP, SQL; она также умеет оптимизировать и анимировать GIF-изображения (баннеры), может создавать файлы справки (CHM);

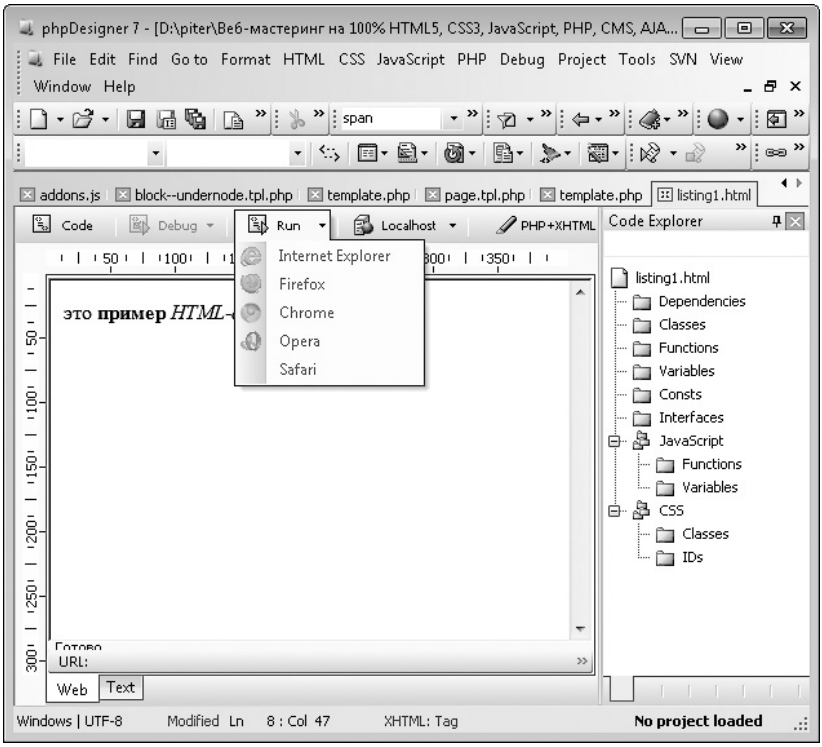


Рис. 1.7. Просмотр HTML-документа в одном из популярных браузеров

- HTML Pad (http://www.soft.necromancers.ru/prog_4764.html) — поддерживает HTML, JavaScript, VBScript, SSI, ASP и Perl, умеет создавать макросы;
- PSPad (http://www.pspad.com/ru/download.php) — еще одна бесплатная замена Блокнота, которой известен синтаксис HTML и CSS;
- HtmlReader (http://manticora.ru/download.htm) — бесплатный редактор, позволяющий просматривать, редактировать и создавать HTML-документы;
- EditPlus (http://www.editplus.com) — еще один редактор, подсвечивающий синтаксис HTML, PHP, Java, C/C++, CSS, ASP, Perl, JavaScript, VBScript, Python and Ruby on Rails.

Валидаторы

Помимо редакторов HTML, существуют другие классы программ, помогающих разрабатывать сайты.

В первую очередь следует упомянуть валидаторы — сервисы Интернета, проверяющие HTML-документ на ошибки и следование правилам выбранного `DOCTYPE`.

Как правило, большинство ошибок в HTML-документе не сказываются на его отображении в браузерах. Валидатор предназначен прежде всего для разработчиков сайтов.

Наиболее популярным валидатором является сервис http://validator.w3.org (рис. 1.8). Он позволяет проверить корректность сайта в Интернете, HTML-файла либо непосредственно HTML-кода.

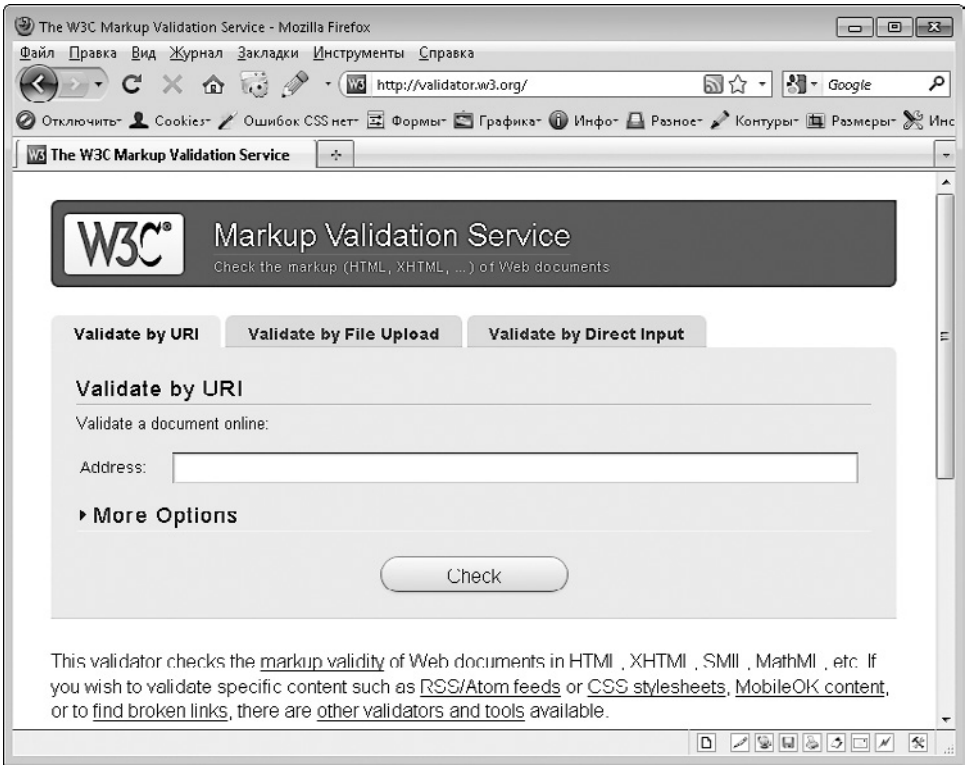


Рис. 1.8. Валидатор HTML

По умолчанию DOCTYPE и кодировка сайта определяются автоматически. Но вы можете указать их вручную, воспользовавшись раскрывающейся областью More Options (Больше параметров).

Если в результате проверки будут обнаружены проблемы, сервис сообщит о количестве найденных ошибок и неточностей и подробно опишет каждую из них, но, к сожалению, только на английском языке (рис. 1.9).

Если же ни одной проблемы найдено не будет, появится соответствующее сообщение и предложение установить на вашем сайте значок, который будет информировать посетителей о ваших знаниях и умениях (рис. 1.10).

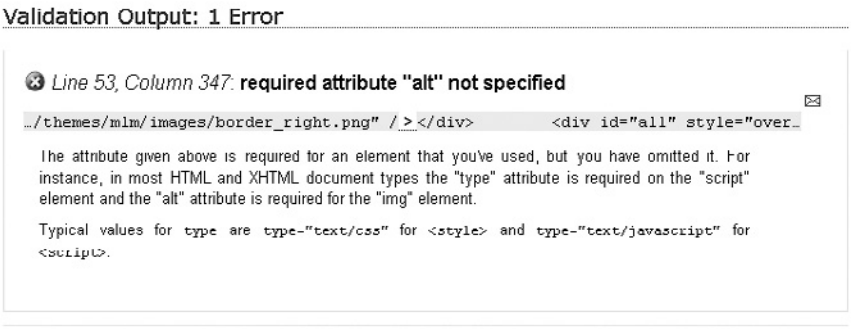


Рис. 1.9. Список ошибок, найденных валидатором

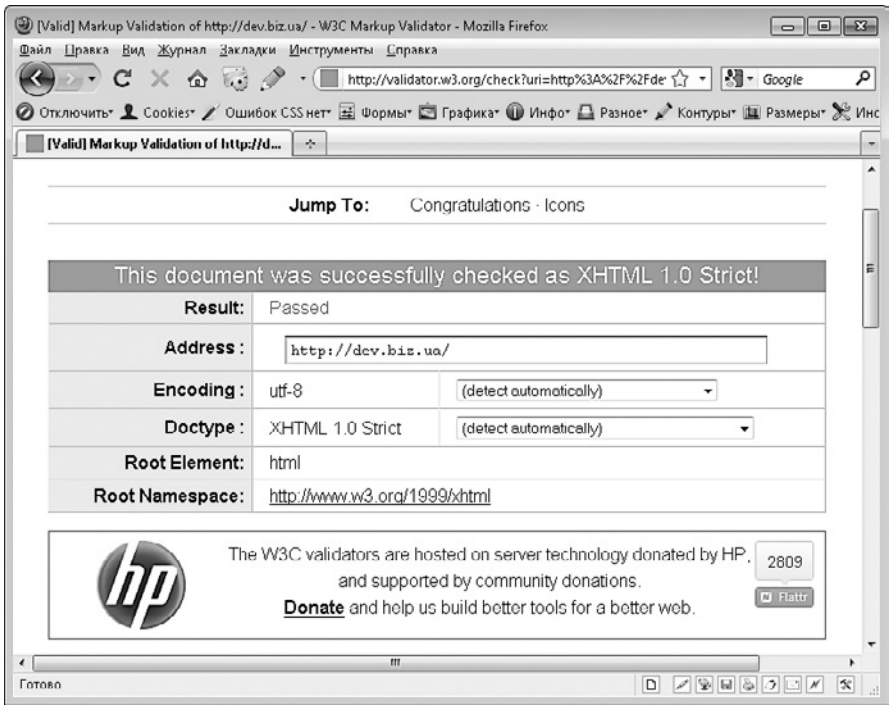


Рис. 1.10. Успешное прохождение валидации

Если у вас нет доступа к Интернету, проверить HTML-документ на валидность можно с помощью бесплатной программы Tidy. Ее можно скачать с сайта http://tidy.sourceforge.net.

А если вы используете браузер Mozilla Firefox, то в качестве валидатора, не требующего доступа к Интернету, может выступать расширение HTML Validator. Его скачать можно по адресу http://users.skynet.be/mgueury/mozilla/.

ПРИМЕЧАНИЕ

В приложении 1 приведено описание основных ошибок и предупреждений, которые отображают валидаторы, работающие на основе HTML Tidy; а также способы решения обнаруженных проблем.

Графические редакторы

Верстальщик сайтов должен знать не только HTML, CSS, JavaScript. Он работает с макетами дизайна, которые, как правило, сделаны в Photoshop. А значит, ему необходимы графический редактор Adobe Photoshop и базовые знания по работе с ним.

Перед тем как начать верстку сайта, необходимо выделить из полученного макета графические элементы и сохранить их в виде отдельных изображений. Ваших знаний должно хватать на выполнение этих операций.

Если у вас нет возможности приобрести редактор Adobe Photoshop, вы можете воспользоваться бесплатным онлайн-сервисом http://pixlr.com/editor (рис. 1.11).

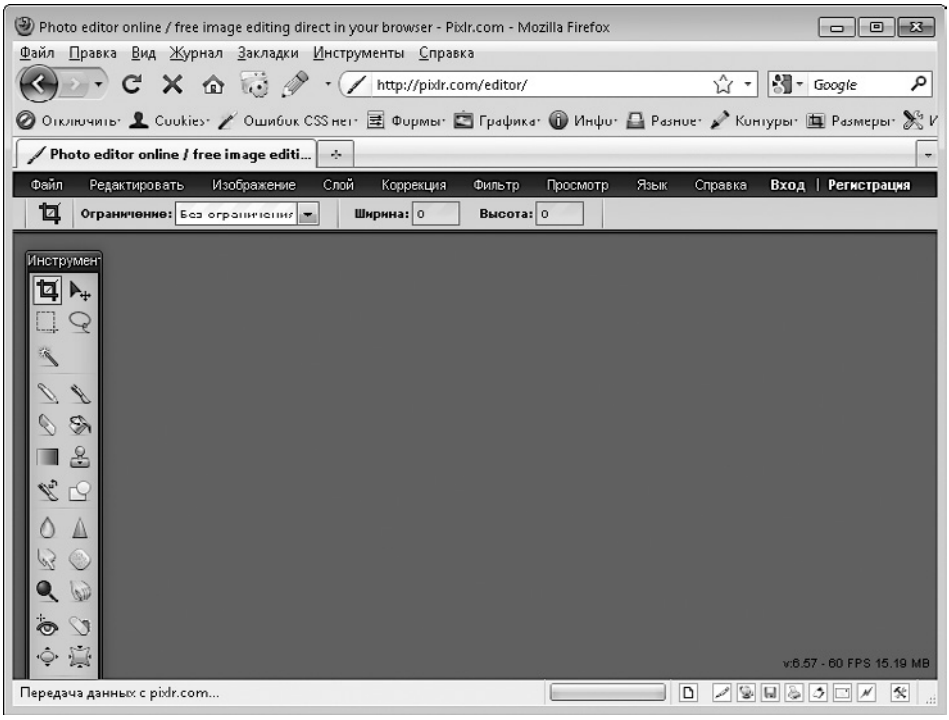


Рис. 1.11. Онлайн-версия графического редактора

Заменой Adobe Photoshop могут также служить бесплатные графические редакторы, поддерживающие работу с родным форматом Adobe Photoshop — PSD:

- Paint.Net (http://www.getpaint.net/download.html) с установленным плагином Paint.NET PSD Plugin (http://psdplugin.codeplex.com);
- PSD viewer (http://psdviewer.org) — приложение, предназначенное только для просмотра PSD-файлов;
- GIMP (http://www.gimp.org/downloads) — бесплатный аналог Adobe Photoshop, изначально предназначенный для ОС Linux, но сейчас работающий и в Windows.

В исключительных случаях макет могут принести не в формате PSD, а в других форматах:

- CDR — формат программы векторной графики CorelDRAW; в этом случае для просмотра макета можно использовать следующие бесплатные программы:
  - XnView (http://xnview.com) — дает возможность просматривать CDR-документы;
  - CDR Explorer (http://sk1project.org) — позволяет просматривать отдельные объекты, из которых состоит CDR-изображение;
  - SK1 (http://sk1project.org) — бесплатный векторный редактор для Linux;
  - Inkscape (http://inkscape.org/index.php?lang=ru) — бесплатный векторный редактор; не позволяет работать с CDR, но вы можете конвертировать CDR-файл в поддерживаемый редактором формат с помощью программы UniConvertor;
  - UniConvertor (http://sk1project.org) — дает возможность конвертировать изображения из формата CDR в форматы AI, PDF, SVG и др.;
- AI — формат программы векторной графики Adobe Illustrator; в качестве бесплатного просмотрщика можно использовать программу XnView с установленной программой Ghostscript (http://www.ghostscript.com);
- PDF — можно открыть любым бесплатным просмотрщиком файлов PDF, например Acrobat Reader.

Если макет предоставлен в одном из этих форматов, лучше открыть его в подходящей программе и сохранить либо в формате PSD, если такая возможность есть (например, это возможно для файлов в формате AI), либо в формате EPS (форматы CDR, AI), либо в формате PNG24.

Файлы формата EPS также являются файлами векторной графики, но их можно открыть в графическом редакторе Adobe Photoshop.

Файлы растровой графики PNG24 также можно открыть в Photoshop. При сохранении в формате PNG24 потери качества изображения не происходит.

После сохранения макета в одном из поддерживаемых Photoshop форматов дальнейшие работы над макетом будут происходить в Adobe Photoshop.

Другие программы

Помимо перечисленных программ, на помощь веб-мастеру могут прийти различные плагины для браузеров. В частности, незаменимым является расширение Web Developer для браузера Mozilla Firefox (https://addons.mozilla.org/ru/firefox/addon/web-developer). После установки данного расширения в браузере Mozilla Firefox появится дополнительная панель с огромным количеством команд, упрощающих разработку сайтов (рис. 1.12).

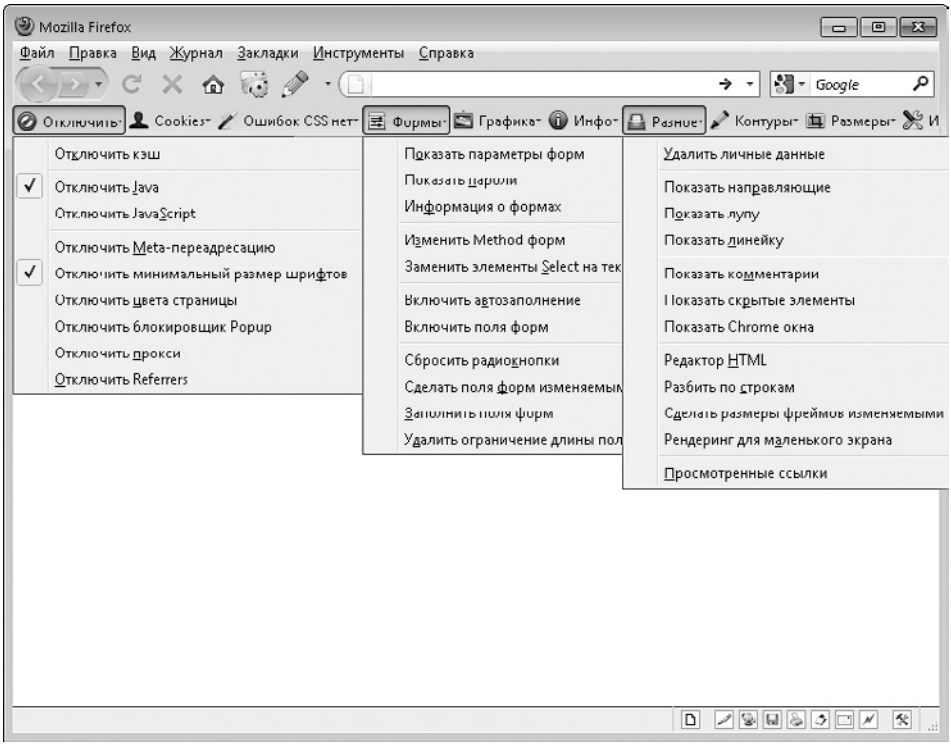


Рис. 1.12. Расширение Web Developer для браузера Mozilla Firefox

Наиболее полезные из них следующие:

- Отключить ▸ Отключить кэш — отключает использование кэша браузера Mozilla Firefox для хранения файлов веб-страниц;
- Отключить ▸ Отключить JavaScript ▸ Полностью — запрещает выполнение JavaScript на сайтах, загруженных в любой вкладке браузера Mozilla Firefox;
- Cookies ▸ Отключить Cookies ▸ Все Cookies — отключает использование Cookies для любого сайта, открытого в браузере Mozilla Firefox;

- Ошибок CSS нет ▸ Редактор CSS — позволяет просмотреть все CSS-файлы, используемые загруженной веб-страницей;
- Формы ▸ Показать параметры форм — отображает идентификаторы всех форм и их элементов на странице, а также значения таких атрибутов, как `action`, `method`, `name`, `size`, `maxlength`;
- Формы ▸ Показать пароли — если в поле для ввода пароля указаны звездочки, позволяет заменить звездочки на введенный пароль;
- Графика ▸ Отключить изображения ▸ Все изображения — запрещает браузеру загрузку и отображение любых изображений, которые есть на открываемых сайтах;
- Графика ▸ Показать атрибуты ALT — отображает рядом с каждым изображением на сайте содержимое атрибута `alt` тега `img` данного изображения;
- Графика ▸ Показать размеры изображений — показывает рядом с каждым изображением на сайте реальные размеры данного изображения;
- Графика ▸ Показать пути изображений — отображает рядом с каждым изображением на сайте адрес, по которому хранится данное изображение;
- Разное ▸ Редактор HTML — открывает встроенный редактор HTML расширения Web Developer;
- Инструменты ▸ Validate HTML — позволяет произвести валидацию открытой страницы с помощью сайта http://validator.w3.org;
- Инструменты ▸ Validate CSS — выполняет валидацию CSS для текущего сайта.

Кроме того, очень полезными являются значки справа, сигнализирующие об ошибках CSS и JavaScript.

Структура документа HTML

Вот мы и подошли к самому интересному. Рабочую среду — редактор HTML — мы подобрали. Основные сведения о HTML получили. Теперь пора переходить к верстке.

Следует сразу сказать, что мы с вами будем изучать только те теги, атрибуты и их значения, которые поддерживаются всеми браузерами. Дело в том, что на заре становления Интернета разработчики браузеров увлекались добавлением в свои продукты дополнительных возможностей и не заявленных ни в одной версии языка HTML тегов, атрибутов, значений. К счастью, с каждой новой версией браузеры стали иметь все меньше и меньше таких уникальных возможностей. И в дальнейшем, возможно, они совсем исчезнут. Соответственно, и изучать их нет никакого смысла.

Основные понятия

Перед тем как перейти к правилам HTML, следует объяснить основные понятия, с которыми вы столкнетесь в процессе обучения. В частности, загадочные понятия «тег», «атрибут» и «значение».

Теги

В языке HTML все, что вы вводите в HTML-документе, будет выводиться на экран браузера сплошным текстом, то есть:

- переводы строк игнорируются и не влияют на отображение текста браузером;
- несколько идущих подряд пробелов заменяются одним.

В результате получается такая неутешительная картина (рис. 1.13).

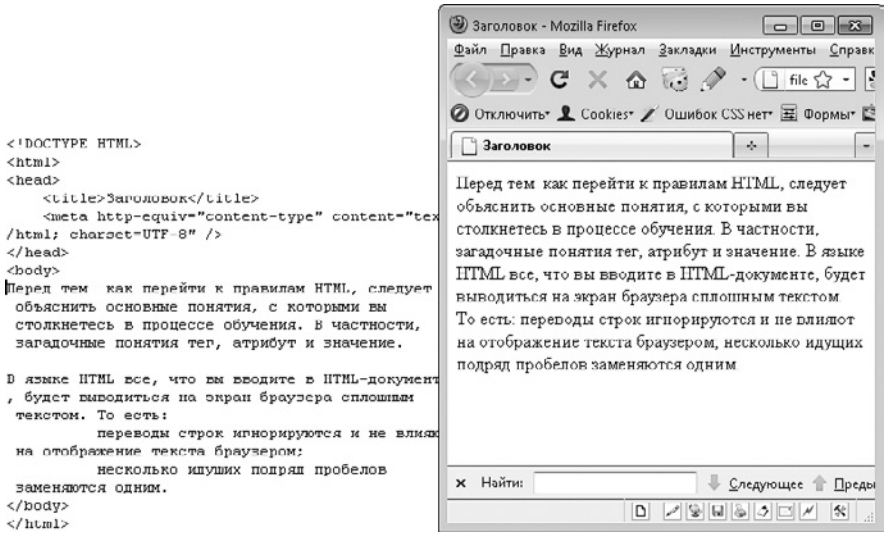


Рис. 1.13. Переводы строк и лишние пробелы в HTML-документе удаляются браузером

Теги были придуманы для того, чтобы отформатировать текст. Теги — это управляющие команды, говорящие браузеру о том, как именно нужно вывести на экран часть текста, заключенную в теге. Фактически тег — это определенная последовательность символов, перед которой идет символ < и после которой идет символ >, то есть выглядит это так: `<тег>`.

ПРИМЕЧАНИЕ

В дальнейшем при упоминании какого-либо тега в книге мы будем опускать открывающую и закрывающую скобки и писать просто название тега. Например, «тег div», вместо «тег ‹div›».

Если браузер встретит какой-либо неизвестный ему тег, то проигнорирует его. Валидатор же сообщит об ошибке, если в HTML-документе будут обнаружены теги, не входящие в выбранную спецификацию HTML.

Теги бывают одинарные и парные.

Одинарный тег. Состоит только из одной последовательности символов — `<тег>`, то есть в него не заключен никакой текст. Одинарному тегу текст не нужен. Он служит для других целей — перевода строки, вывода картинок, вывода элементов формы и т. д.

Парный тег. Парные теги еще называют контейнерами. Они имеют такой вид: `<тег></тег>`, то есть открывающий тег — `<тег>` и закрывающий тег — `</тег>`.

Следует внимательно относиться к правильному написанию парных тегов. Отсутствие или ошибки в написании закрывающего парного тега могут привести к неправильному отображению страницы одним (как правило, Internet Explorer) или несколькими браузерами. Верстка может поплыть.

ПРИМЕЧАНИЕ

Под фразой «верстка поплыла» понимаются любые неконтролируемые и не предусмотренные верстальщиком изменения в сверстанной веб-странице. Изменения могут проявляться во всех, но чаще в одном или нескольких браузерах. Причем в более сложных случаях верстка может плыть только при определенной ширине или высоте экрана браузера либо при каких-то других условиях.

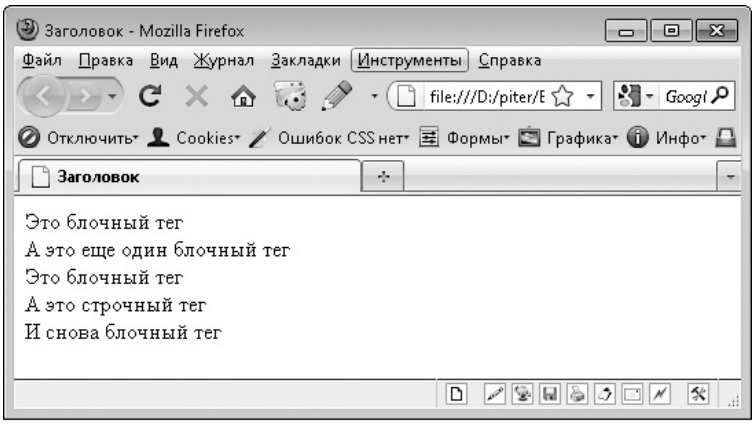
Между открывающим и закрывающим тегами находится какой-либо текст: `<тег>какой-либо текст</тег>`. При этом в тексте может содержаться любое количество других одинарных и/или парных тегов.

Парный тег влияет только на тот текст и те теги, которые располагаются внутри него (между его открывающим и закрывающим тегами).

Типы тегов. Теги принято разделять на отдельные группы. Делается это на основе назначения тега либо его поведения. Один и тот же тег может входить в несколько групп. Итак, теги могут входить в следующие группы (то есть иметь следующий тип):

- теги верхнего уровня — теги, из которых состоит базовая структура документа — его каркас, то есть определяющие раздел заголовка и тела документа: `html`, `head`, `body`;
- теги заголовка документа — к данной группе относятся все теги, которые указываются внутри тега `head`: `title`, `meta`;
- блочные элементы — в данный тип входят все теги, которые занимают всю доступную ширину экрана, независимо от того, сколько текста и других элементов

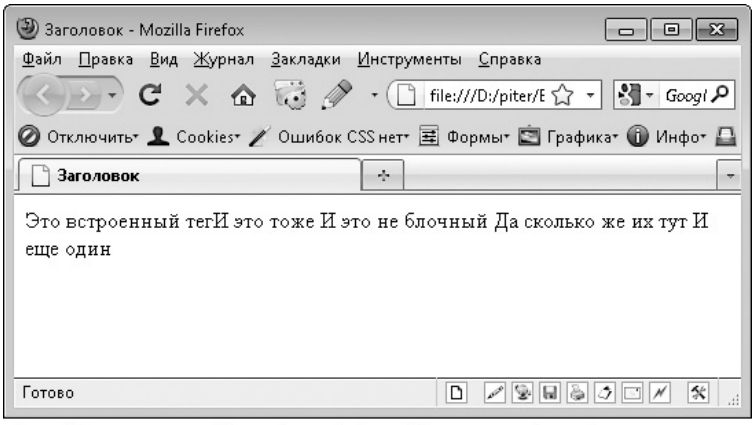
находится внутри тега: `blockquote`, `div`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `hr`, `p`, `pre`; любой тег, идущий за блочным тегом, будет отображаться с новой строки (рис. 1.14);




```
<div>Это блочный тег</div><div>А это еще один блочный тег</div>
<div>Это блочный тег</div><span>А это строчный тег</span>
<div>И снова блочный тег</div>
```

Рис. 1.14. Отображение блочных тегов

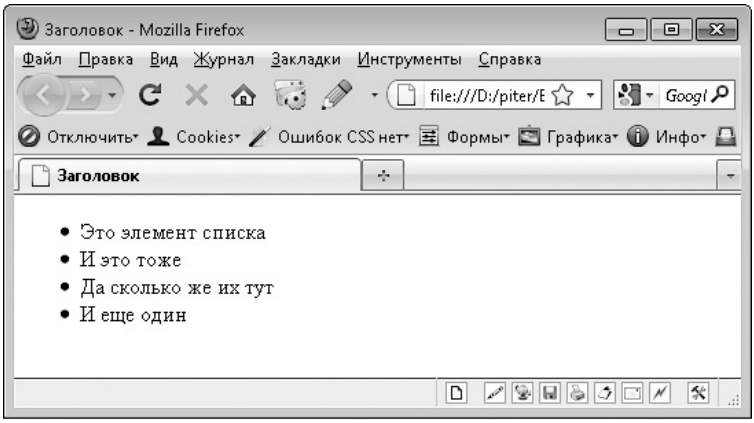
- встроенные (строчные) элементы — в отличие от блочных, теги данного типа занимают лишь ту ширину экрана, которая им реально нужна; при этом следующий за ними встроенный элемент на новую строку не переносится (конечно, если ему хватает места рядом с предыдущим строчным элементом): `a`, `b`, `big`, `em`, `i`, `img`, `small`, `span`, `strong`, `sub`, `sup` (рис. 1.15);




```
<span>Это встроенный тег</span><span>И это тоже</span>
<span>И это не блочный</span>
<span>Да сколько же их тут</span>
<span>И еще один</span>
```

Рис. 1.15. Отображение строчных тегов

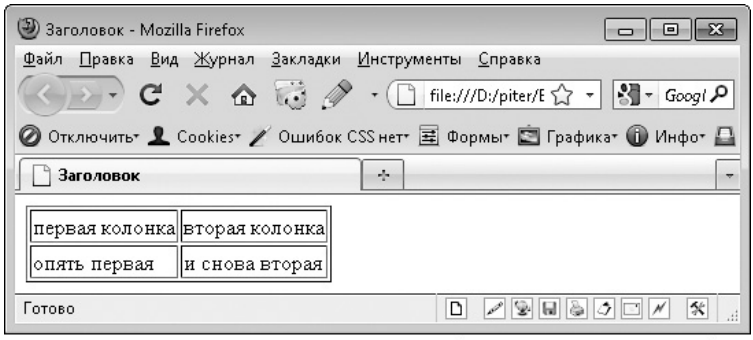
- универсальные элементы — теги данного типа могут быть как блочными, так и строчными: `del`, `ins`;
- списки — в данную группу тегов входят все теги, которые применяются для создания нумерованных либо ненумерованных списков: `ul`, `ol`, `li`, `dd`, `dt`, `dl` (рис. 1.16);




```
<ul>
<li>Это элемент списка</li><li>И это тоже</li>
<li>Да сколько же их тут</li>
<li>И еще один</li>
</ul>
```

Рис. 1.16. Отображение списков

- таблицы — как можно понять из названия, в данную группу входят теги, предназначенные для создания таблиц: `table`, `thead`, `tbody`, `td`, `th`, `tr` (рис. 1.17);




```
<table border="1"><tr><td>первая колонка</td><td>вторая колонка</td>
<tr>
<td>опять первая</td>
<td>и снова вторая</td>
</tr>
</table>
```

Рис. 1.17. Отображение таблиц

- фреймы — все теги, предназначенные для создания фреймов (области внутри HTML-документа, в которые загружается другой HTML-документ): `frame`, `frameset`, `iframe`.

Наиболее часто в своей работе вы будете использовать блочные и строчные теги. Чтобы лучше понять их различия, рассмотрим небольшую таблицу (табл. 1.1).

Таблица 1.1. Разница между блочными и строчными тегами

Характеристики	Блочный тег	Строчный тег
Какие теги могут находиться внутри данных тегов?	Как блочные, так и строчные	Только строчные
Тег начинается с новой строки?	Да, всегда	Нет, переносятся на новую строку, только если на предыдущей нет места либо если предыдущий тег — блочный
Теги занимают всю доступную ширину окна браузера?	Да	Нет, их ширина равна ширине содержимого плюс размеры заданных отступов, полей и границ

Атрибуты и их значения

Внутри одинарного или открывающего тега могут находиться атрибуты и их значения:

```
<тег атрибут1="значение1" атрибут2="значение2">текст</тег>
```

Атрибуты и их значения необходимы для того, чтобы каким-либо образом изменить функциональность того или иного тега. Порядок расположения атрибутов в теге ни на что не влияет.

Несколько атрибутов внутри одного тега разделяются пробелами. Атрибуты можно также переносить на новую строку:

```
<тег

 атрибут1="значение1"

 атрибут2="значение2">текст</тег>
```

Мы изучаем синтаксис языка XHTML. В нем каждый атрибут обязан иметь значение. Даже если в более ранних версиях языка HTML для этого атрибута вообще не существовало возможных значений.

В качестве значения атрибута могут выступать либо определенные ключевые слова, либо любой пользовательский текст.

Кавычки. Значение атрибута указывается в кавычках после знака =. Кавычки могут быть двух типов:

- одинарные: ';
- двойные: ".

Другие кавычки — ‘, “, `, « — использовать нельзя.

Разницы между одинарными и двойными кавычками нет.

Одни и те же кавычки не могут быть вложены друг в друга. Такие варианты кода недопустимы:

- `атрибут="значе"ние";`
- `атрибут="значе"ни"е".`

Если внутри одних кавычек нужно указать другие, следует использовать кавычки разного типа: если внешние кавычки одинарные, то внутри нужно использовать двойные кавычки, и наоборот. Следующий синтаксис правилен и к ошибкам не приведет:

- `атрибут="значе'ние";`
- `атрибут="значе'ни'е";`
- `атрибут='значе"ние';`
- `атрибут='значе"ни"е'.`

Если же требуется использовать именно тот тип кавычек, который применялся для внешнего ограничения, то внутренние кавычки следует экранировать. Точно так же следует поступить и в том случае, если внутри одного типа кавычек должны находиться сразу два типа кавычек: одинарные и парные. Итак, следующий код правилен и к ошибкам не приведет:

- `атрибут="значе\"ние";`
- `атрибут="значе\"ни\"е";`
- `атрибут="значе'ни\"е";`
- `атрибут='зна\'че"ни"е'.`

Эти же правила действуют и в других языках программирования: JavaScript и PHP.

Значения. Все значения атрибутов можно разделить на несколько типов.

- Задающие цвет. В качестве значения может выступать:
  - специальное ключевое слово: `black` (черный), `blue` (синий) и т. д.;
  - шестнадцатеричное представление цвета: `#ff0000` (красный), `#000000` (черный) и т. д.

- Задающие размер. В качестве значения может выступать:
  - просто целое число без каких-либо единиц измерения, означающее, что размер задан в пикселах (самая маленькая точка на экране монитора);
  - целое число от 1 до 100 в процентах, определяющее размер элемента относительно размера родительского элемента либо окна браузера, если ни для одного из родительских элементов размер задан не был, например `70%`, `30%`.
- Задающие адрес в Интернете. В качестве значения может выступать:
  - абсолютный адрес — http://microsoft.com, http://mail.ru/content.html, то есть адрес в формате `протокол://сайт/страница`;
  - относительный адрес — `/content.html`, `../css/style.css`, то есть адрес относительно адреса загруженной в данный момент страницы; относительные адреса всегда указывают на тот же сайт, который открыт в браузере сейчас.
- Различные ключевые слова.

Атрибуты, для которых используются значения, задающие цвет и размер, в данный момент считаются устаревшими. Поэтому мы не будем их рассматривать. Более подробно мы поговорим о цвете и размере в главе 2, посвященной CSS.

Спецсимволы

Как уже говорилось, между парными тегами может находиться текст. Но что насчет самого этого текста? Все ли символы можно использовать в тексте? И как можно ввести символы, которых нет на клавиатуре?

Оказывается, в тексте HTML-документа разрешено применять не все символы, которые вы видите на клавиатуре. Впрочем, это логично. Если каждый тег начинается с символа < и заканчивается символом >, то вполне логично, что эти символы нельзя указывать внутри обычного текста. Иначе как браузер сможет определить, что это — тег или введенная вами фраза?

Чтобы решить проблему с добавлением символов, которые запрещено вводить в тексте документа, был разработан набор спецсимволов — последовательностей, которые заменяют собой какой-либо запрещенный для ввода символ. Каждая такая последовательность (ссылка-мнемоника) обязательно начинается с символа `&`, а заканчивается символом `;`. Между этими символами можно ввести:

- либо имя спецсимвола;
- либо его числовой код, который вводится после знака `#` (то есть при наборе спецсимвола путем указания его числового кода нужно использовать следующую запись: `&#код;`).

Как правило, спецсимволы вводятся путем указания их имени — имя запомнить намного проще, чем число. Поэтому в дальнейшем мы будем применять именно эту запись.

Из-за того что символ `&` используется для обозначения начала спецсимвола, он также считается запрещенным для применения в тексте документа.

Но как же ввести в тексте документа символы <, >, &? Для этого достаточно заменить их следующими спецсимволами (рис. 1.18):

- `>` — `>`;
- `<` — `<`;
- `&` — `&`.

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Заголовок</title>
    <meta http-equiv="content
-type" content="text/html;
 charset=UTF-8" />
</head>
<body>


Но как же нам ввести в тексте
 документа символы &lt;, &gt;,
&amp;? Для этого достаточно
 заменить их на следующие
 спецсимволы (см. рис.1.18):
&gt; — &amp;gt;
&lt; — &amp;lt;
&amp; — &amp;amp;


</body>
</html>
```

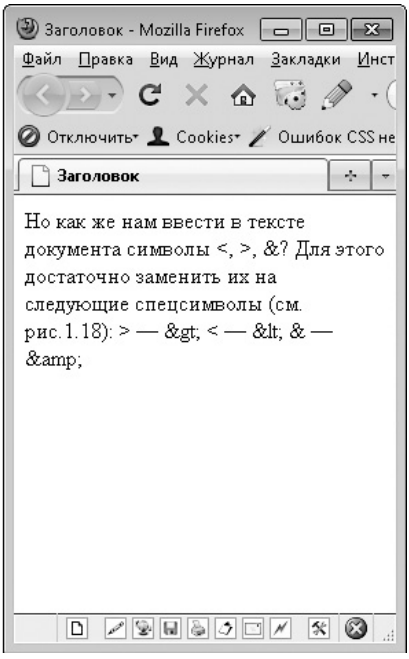


Рис. 1.18. Использование спецсимволов в HTML-документе

Вы уже знаете, что внутри двойных кавычек можно использовать еще одни двойные кавычки лишь в том случае, если экранировать их. Однако при вводе обычного текста (не кода на каком-либо языке сценариев) вместо экранирования можно применять спецсимвол, заменяющий двойные кавычки: `&quot;`.

Спецсимволы используются также для того, чтобы добавить в документ символы, которые нельзя ввести с клавиатуры. Например, часто с помощью спецсимвола `&copy;` верстальщики добавляют в HTML-документ символ копирайта (©).

ПРИМЕЧАНИЕ

Если вы создаете HTML-документ в формате UTF-8, то для вставки символов, отсутствующих на клавиатуре, можно использовать не только спецсимволы. В этом случае вы можете воспользоваться программой для просмотра символов, которая входит в стандартную поставку ОС Windows — charmap.exe. Запустите данное приложение (рис. 1.19). Далее с помощью кнопки Выбрать выберите в нем нужный символ. После чего воспользуйтесь кнопкой Копировать, чтобы поместить выбранный символ в буфер обмена. После этого останется лишь вставить выбранный символ в HTML-документ (сочетание клавиш Ctrl+V).

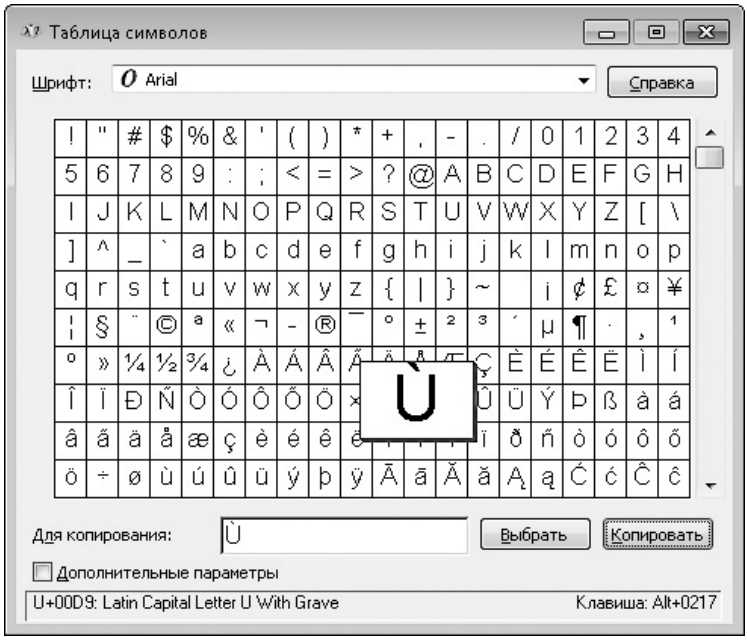


Рис. 1.19. Таблица символов (charmap.exe) — стандартное приложение ОС Windows

Помимо символа копирайта, довольно часто используются следующие спецсимволы:

- ` ` — крайне полезный спецсимвол, который позволяет добавить в документ неразрывный пробел; такой пробел будет выглядеть как обычный, но он гарантирует, что слова, между которыми он стоит, при любых обстоятельствах будут находиться на одной строке;
- `&euro;` — знак доллара с клавиатуры ввести легко, а вот как ввести знак евро (€) без данного спецсимвола?

Полный список спецсимволов представлен в приложении 2.

Основной синтаксис

Запомните основные правила синтаксиса XHTML:

- HTML-документ обязательно должен начинаться с `DOCTYPE`;
- все теги и атрибуты должны записываться строчными буквами (в нижнем регистре), так как язык XHTML регистрозависим;
- любые значения атрибутов необходимо заключать в кавычки;
- все парные теги должны иметь закрывающий тег;
- все одинарные теги должны завершаться слешем перед закрывающей скобкой: `<br />`, `<img />`, `<input />`, `<hr />`, `<meta />`;
- каждый тег должен быть корректно вложен в другой, ошибки неправильного вложения недопустимы (например, `<strong>пример <em>ошибочного</strong> вложения</em>`);
- сокращенные атрибуты (без значения) запрещены; для них в качестве значения обязательно нужно указывать название атрибута: `checked="checked"`, `compact="compact"`, `disabled="disabled"`, `ismap="ismap"`, `multiple="multiple"`, `nohref="nohref"`, `noresize="noresize"`, `noshade="noshade"`, `nowrap="nowrap"`, `readonly="readonly"`, `selected="selected"`;
- для обозначения цели в тегах `a`, `frame`, `iframe`, `img`, `map` вместо атрибута `name` необходимо использовать атрибут `id`.

Большинство из перечисленных правил необязательны для языка HTML 4 и более ранних. Но лучше сразу приучать себя к синтаксису, который одинаково правильно будет работать во всех версиях HTML. Таким синтаксисом и является синтаксис языка XHTML.

Следование данным правилам позволяет минимизировать количество ошибок в коде и, соответственно, добиться одинакового отображения веб-страницы во всех браузерах. Ведь, как показывает практика, наиболее частыми причинами неправильного отображения веб-страницы в браузере являются следующие:

- отсутствие `DOCTYPE`;
- неверно написанный тег или атрибут;
- отсутствие обязательного для тега атрибута;
- использование тегов и атрибутов, являющихся уникальными для одного из браузеров и не поддерживающихся другими браузерами;
- неверное вложение тегов.

Большинство таких ошибок можно обнаружить при валидации HTML-документа. В приложении 1 приведен список основных ошибок, которые находят валидаторы на основе HTML Tidy, а также способы их решения.

Семантическая верстка

Помимо определенных правил, существуют условия, которые нужно соблюдать, если вы хотите верстать правильно, то есть использовать так называемую семантическую верстку.

Семантическая верстка — это способ верстки, при котором каждый тег, атрибут и его значение применяются по своему истинному назначению. При этом следует стремиться к максимальной наглядности и понятности верстки как для посетителей, так и для разработчиков, читающих созданный вами код. Возможно, это определение не совсем вам понятно. Тогда хотя бы запомните, что нужно просто следовать условиям, которые предъявляет семантическая верстка.

Под условиями в семантической верстке понимаются случаи, в которых нужно применять тот или иной тег. Например, для создания любых меню в семантической верстке должен использоваться тег `ul` или `ol`.

В последнее время клиенты начали требовать, чтобы верстка была не только валидной[1] и кросс-браузерной, но и семантической. И пусть таких клиентов пока немного, лучше сразу учиться верстать правильно. Поэтому далее в книге мы будем рассматривать не только теги, но и случаи, когда их стоит применять в контексте семантической верстки.

Содержимое тега head

Как вы уже знаете, любой HTML-документ начинается с объявления `DOCTYPE`.

На следующей строке идет парный тег `html`, в который вложены два других парных тега — `head` и `body`:

```
<html>

<head></head>

<body></body>

</html>
```

[1] То есть верстка, успешно прошедшая валидацию на одном из сайтов валидации HTML.

Сам HTML-документ, отображаемый на экране браузера, расположен внутри тега `body`, а вот содержимое тега `head` на экран вообще не выводится. Однако от этого тег `head`, называемый заголовком документа, не становится менее важным.

Внутри тега `head` записывается информация, крайне важная для правильного отображения веб-страницы: ссылки на CSS и JS-файлы, ссылки на RSS-ленты, используемая страницей кодировка и другие сведения.

Но давайте по порядку. Хотя эта фраза в данном случае неприменима — порядок указания тегов внутри тега `head` не имеет значения.

Заголовок веб-страницы

Каждая веб-страница должна иметь заголовок. Как правило, он отображается в заголовке окна браузера (рис. 1.20) и служит для идентификации сайта как для посетителя, так и для роботов поисковых систем.

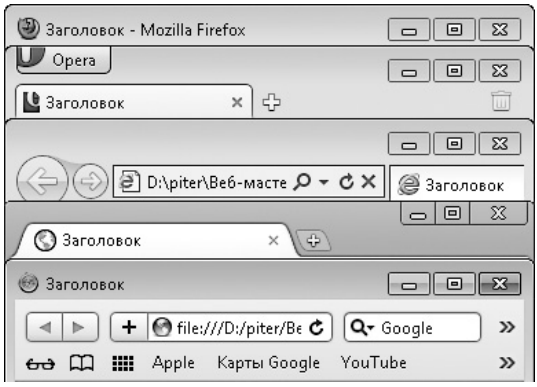


Рис. 1.20. Отображение заголовка веб-страницы в различных браузерах

Для указания заголовка служит парный тег `title` — заголовок вводится между его открывающим и закрывающим тегами.

Тег meta

Внутри тега `head` может также находиться любое количество тегов `meta` (метатегов). Они предназначены для описания информации, необходимой браузеру для корректного отображения веб-страницы, а также поисковым машинам.

Сам по себе тег `meta` никакого функционала не несет. Назначение тега `meta` зависит от атрибутов, которые вы в нем указали. Именно из-за этой его особенности внутри тега `head` может находиться любое количество тегов `meta` — с разным значением одних и тех же атрибутов.

Определение кодировки. Наиболее важным вариантом тега `meta` является вариант, задающий кодировку HTML-документа:

```
<meta http-equiv="Content-Type" content="text/html; charset=
кодировка">
```

Его указание обязательно для каждого HTML-документа.

Чаще всего для HTML-документа используют одну из следующих кодировок:

- `Windows-1251` — позволяет применять символы европейского и латинского алфавитов и писать тексты на русском, украинском, белорусском, сербском, болгарском, английском языках;
- `UTF-8` — дает возможность использовать символы любого языка мира.

Описание сайта. Еще одним вариантом, который часто применяется при создании сайта, является следующий:

```
<meta name="description" content="описание вашего сайта">
```

Он позволяет задать краткое описание создаваемого вами сайта либо текущей его страницы, то есть указать, о чем данный сайт или отдельная его страница.

Данное описание используется поисковыми системами. В частности, оно отображается в результатах поиска для описания вашего сайта, если более подходящего текста на странице найдено не было (рис. 1.21).

Задание ключевых слов. Вместе с вариантом метатега, задающим описание сайта, часто указывают вариант, который определяет ключевые слова, характеризующие содержимое сайта:

```
<meta name="keywords" content="HTML, META, метатег, тег">
```

Таким образом, в атрибуте `content` через запятую указываются слова и фразы, характеризующие сам сайт и/или содержимое текущей страницы сайта.

Содержимое данного варианта метатега может использоваться поисковыми системами для определения позиции вашего сайта в результатах поиска по определенной фразе. Хотя руководство очень многих поисковых систем отрицает использование содержимого данного варианта метатега, большинство веб-мастеров не забывают его указывать. Так, на всякий случай.

Другие теги

Внутри тега `head` практически всегда находятся другие теги. Но их назначение и синтаксис мы рассмотрим в следующих главах.

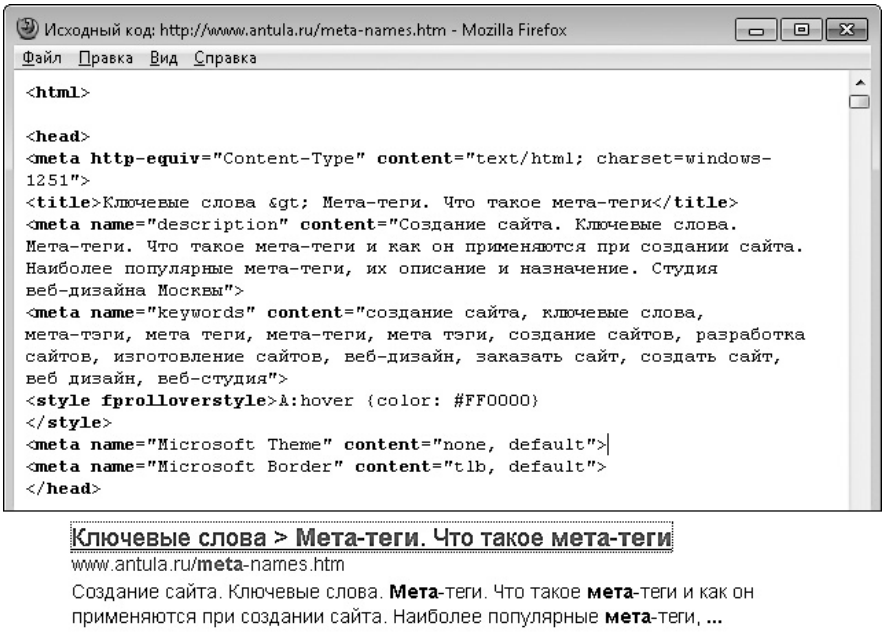


Рис. 1.21. Использование описания страницы поисковыми системами

Работаем с макетом

Итак, с чего начинается верстка сайта? Предполагается, что шаблон HTML-документа с базовой структурой у вас уже есть. Поэтому точно не с его создания.

Предположим также, что у вас уже есть макет будущей веб-страницы в формате PSD, сделанный дизайнером. Именно с изучения такого макета любая верстка и начинается. Мы попробуем сверстать макет, представленный на рис. 1.22.

ПРИМЕЧАНИЕ

Выражаю благодарность за предоставленный макет дизайнеру Анжеле Романовской (http://angedesign.ru).

Откройте макет в программе Adobe Photoshop и изучите его. Подумайте, как лучше сверстать отдельные элементы макета. После чего разбейте макет на отдельные картинки, из которых в дальнейшем будете верстать веб-страницу.

Первым делом стоит сохранить в виде отдельных картинок такие элементы, как логотип, значки, корзину, фотографии. То есть цельные изображения, с которыми в дальнейшем ничего не нужно будет делать.

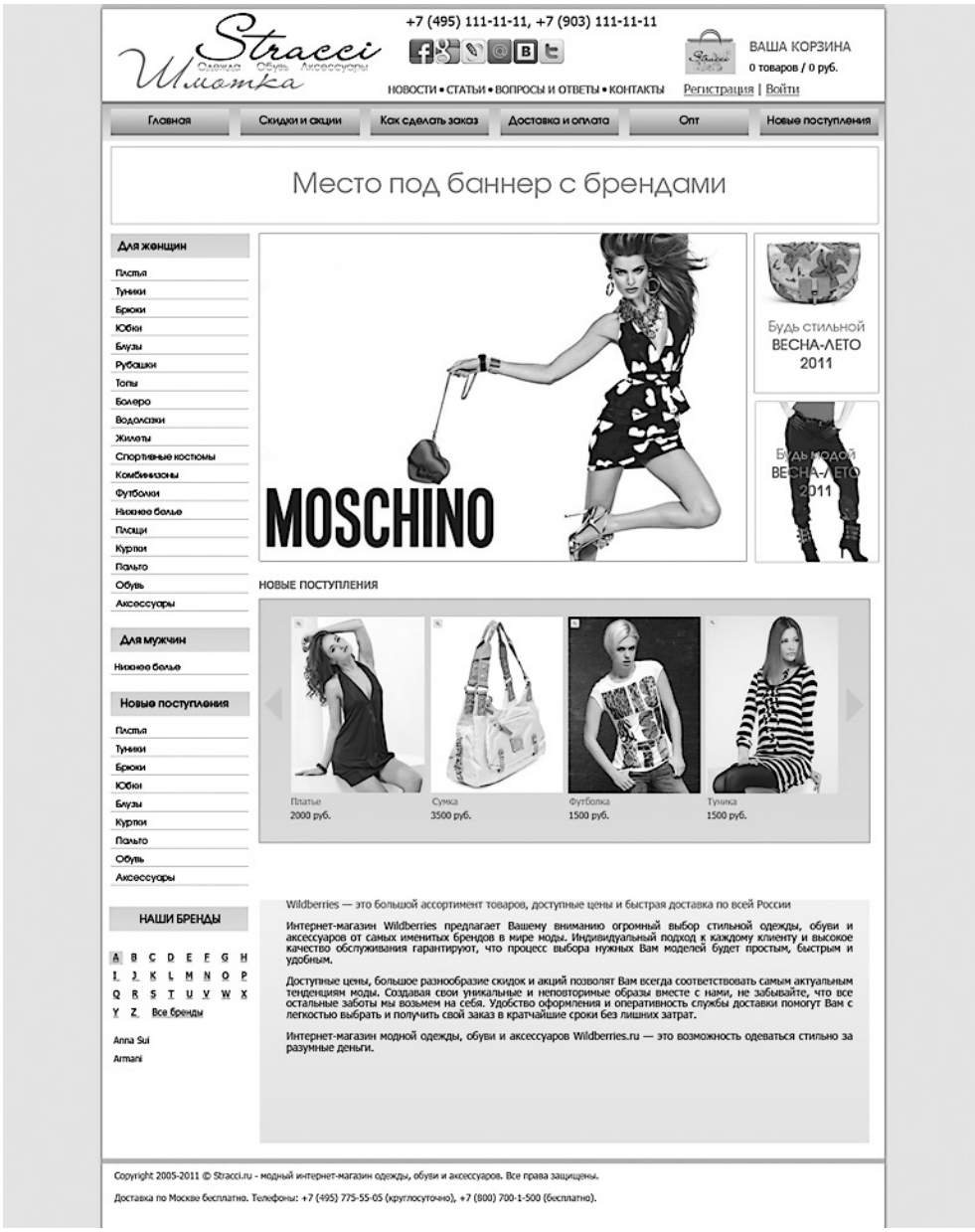


Рис. 1.22. Макет веб-страницы

Выделяем изображение из макета. Если вы плохо знаете Adobe Photoshop, у вас может возникнуть вопрос: как можно выделить из макета отдельное изображение? На самом деле все просто.

Скорее всего, нужное вам изображение находится на отдельном слое — найдите этот слой на вкладке Слои. После этого выполните следующее.

1. Нажмите клавишу Ctrl и щелкните кнопкой мыши на названии слоя на вкладке Слои. В результате выделятся все пикселы, которые есть на выбранном слое.
2. Нажмите сочетание клавиш Ctrl+C, чтобы скопировать все пикселы изображения в буфер обмена.
3. Нажмите сочетание клавиш Ctrl+N, чтобы создать новый документ, ширина и высота которого будут автоматически установлены таким образом, чтобы вместить изображение, находящееся в буфере обмена.
4. Воспользуйтесь сочетаением клавиш Ctrl+V, чтобы поместить изображение из буфера обмена в новый документ.
5. Если изображение содержит прозрачные пикселы, убедитесь, что в созданном вами документе нет фонового слоя, который перекрывал бы прозрачные пикселы изображения. При наличии такого слоя просто скройте либо удалите его.

Данный способ подойдет лишь в том случае, если изображению в макете не присвоены какие-либо стили, размывающие границу изображения (например, стиль, имитирующий тень). Если рисунку присвоены стили, то сделайте следующее.

1. Создайте в Photoshop второй документ, ширины и высоты которого будет достаточно, чтобы с избытком вместить в себя нужное изображение.
2. Перетащите слой с изображением и стилями на рабочую поверхность нового документа.
3. Нажмите клавишу C и кадрируйте новый документ так, чтобы удалить с него пустое пространство.

Если на слое с нужным вам изображением находятся другие изображения, то просто выделите нужное изображение и воспользуйтесь сочетанием клавиш Ctrl+C, чтобы поместить выделенные пикселы в буфер обмена. После этого создайте новый документ (Ctrl+N) и вставьте в него изображение (Ctrl+V).

Сохранение изображения. После того как изображение было помещено в отдельный документ, его следует сохранить. Но не просто сохранить, а сохранить специальным образом — сохранить для Веб. Это позволит максимально уменьшить размер изображения без существенной потери качества. А значит, ускорить открытие сайта.

Сохранить для Веб можно с помощью сочетания клавиш Ctrl+Shift+Alt+S либо команды Файл ▸ Сохранить для веб и устройств (рис. 1.23).

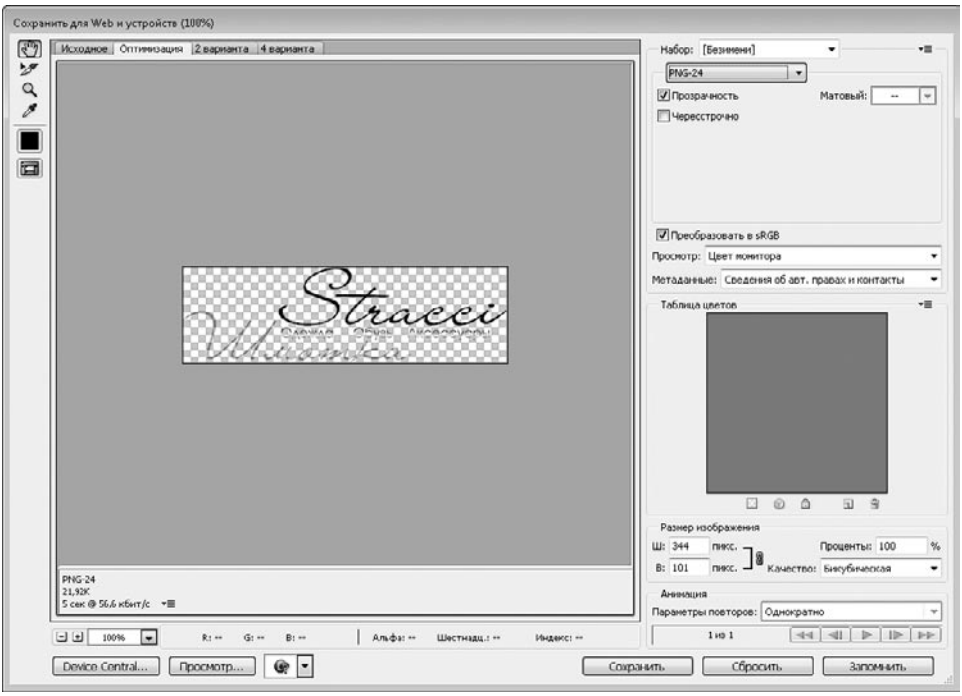


Рис. 1.23. Сохранение изображения для Интернета

Диалог Сохранить для Web и устройств позволяет сохранять изображение в нескольких форматах. Рассмотрим три из них.

- JPG — идеально подходит для сохранения изображений фотографического качества, размер которых не менее 200 пикселов. После выбора данного формата отобразится несколько настроек.

  Самая важная настройка — ползунок Качество, который может принимать значения от 0 до 100. Чем больше его значение, тем лучше будет выглядеть изображение, но тем больше оно будет весить. При сохранении элемента интерфейса веб-страницы использовать качество более 50–60 не имеет смысла. Можно ограничиться и качеством в 40–50. Никто не будет присматриваться к такому изображению.

  Обратите также внимание, что JPG-формат не поддерживает прозрачность. Поэтому, если на изображении есть прозрачные участки, прозрачность будет заменена цветом, указанным в настройке Матовый.

- PNG8 — отлично подходит для сохранения небольших элементов интерфейса с прозрачностью и без нее. А также идеально подходит для сохранения скриншотов с изображением окон, меню и других стандартных элементов интерфейса ОС Windows.

После выбора данного формата отобразится много интересных настроек (рис. 1.24).

Рис. 1.24. Настройки сохранения изображения в формате PNG-8

Чтобы немного уменьшить размер будущего изображения, выберите из первого раскрывающегося списка пункт Адаптивная.

Далее уменьшите количество цветов, которые нужно использовать в изображении. Формат PNG-8 позволяет применять максимум 256 цветов. Но чем меньше будет цветов, тем меньше будет размер изображения. Уменьшая количество используемых цветов, смотрите, как изменится изображение в окне предварительного просмотра. Не забывайте также следить за изменением размера изображения — размер изображения с выбранными в данный момент настройками отображается под окном предварительного просмотра. Если изменения несущественные, можно уменьшить количество цветов еще больше.

Далее обратите внимание на настройку Матовый. Как уже говорилось, формат PNG-8 может работать только с одним видом прозрачности. Точнее, для него прозрачность либо есть, либо ее нет. А как же быть в том случае, если в изображении присутствуют полупрозрачные пикселы? Для работы с такими пикселами предназначен раскрывающийся список Матовый:

- пункт Не показывать данного списка позволяет полностью удалить из изображения полупрозрачные пикселы; в результате часть изображения может быть утеряна, либо края изображения станут неровными (рис. 1.25, *вверху*);
- остальные пункты позволяют указать фоновый цвет, который будет помещен под полупрозрачные пикселы для сохранения целостности и плавности изображения (см. рис. 1.25, *внизу*).

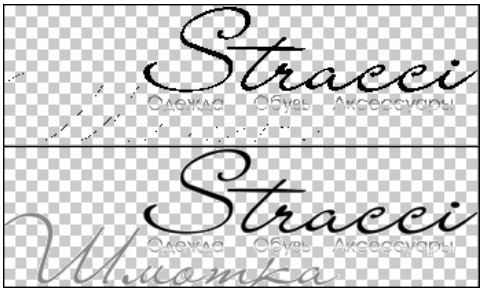


Рис. 1.25. Использование раскрывающегося списка Матовый

Остальные настройки формата PNG-8, как правило, не нужны.

- PNG24 — единственное решение для сохранения изображений с частичной прозрачностью либо для сохранения изображения без потери качества.

  Данный вариант не имеет настроек. Так что просто сохраните изображение.

Остальные форматы при сохранении изображений для Интернета использовать не рекомендуется. Это же касается и формата GIF, на смену которому пришел формат PNG. На данный момент формат GIF следует применять только в одном случае — при создании баннеров.

Рассмотрим также варианты, при которых чаще всего применяется тот или иной формат изображения.

Формат JPG:

- при сохранении больших изображений фотографического качества, с большим количеством цветов и наличием градиентов;
- в случае если размер JPG-изображения меньше PNG-изображения и потеря качества вас не смущает.

Формат PNG-8:

- при сохранении любых мелких изображений, у которых нет прозрачности;
- при сохранении окон, меню и других элементов интерфейса ОС Windows, любого размера; при этом количество цветов стоит выбирать либо в районе 128,

либо в районе 16 (как правило, в таком случае размер изображения будет наименьшим, а качество — наилучшим);

- при сохранении любых мелких градиентов, которые в HTML-документе предполагается повторять по горизонтали и/или вертикали;
- при сохранении любых изображений, у которых есть прозрачность только одного вида — 100 %;
- при сохранении любых изображений с полупрозрачными пикселами, если эти изображения расположены на однотонном фоне, который можно выбрать в раскрывающемся списке Матовый.

Формат PNG-24:

- при сохранении изображений с полупрозрачными пикселами, если эти изображения расположены не на однотонном фоне либо фон предполагается часто изменять;
- по желанию данный формат можно использовать вместо PNG-24 при сохранении любых мелких изображений — размер изображения не сильно изменится.

Нарезаем макет. Далее нас ждет более сложная задача — сохранить в виде отдельных изображений кнопки, тени, фоновые градиенты и подобные элементы интерфейса. Казалось бы, что может быть сложного — просто выделить кнопку и сохранить ее точно так же, как мы сохраняли логотип и значки. Но так ли это на самом деле?

Посмотрим на дизайн основного меню нашего макета (рис. 1.26).

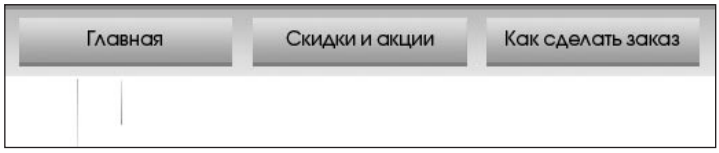


Рис. 1.26. Нарезаем макет

Если вы считаете, что достаточно просто вырезать из макета всю кнопку, то вы не правы. Конечно, если сравнить кнопки, то окажется, что на макете все они имеют одинаковую ширину. И вполне логичным кажется, что вырезать нужно именно всю кнопку. Но перед началом работы верстальщик должен подумать обо всех вариантах использования сверстанной веб-страницы. В том числе что будет, если администратор сайта добавит в меню пункт с настолько длинным названием, что оно не поместится под отведенное пространство для пункта меню? В таком случае:

- часть названия будет перенесена на новую строку и верстка поплывет;
- часть названия будет скрыта, что нехорошо;

- в идеальном случае ширина кнопки увеличится, чтобы вместить в себя весь пункт меню.

Задача верстальщика и состоит в том, чтобы предусмотреть все возможности и выбрать идеальные решения для каждой проблемной ситуации. В нашем случае идеальным решением будет изображение шириной 1 пиксел, которое при верстке будет повторено на всю ширину пункта меню, какой бы эта ширина ни была.

Обратите также внимание на низ кнопки в меню — после градиента идет толстая однотонная линия. Что бы вы сделали с этой линией? Включили бы ее в состав изображения, из которого при верстке будет формироваться кнопка? Хороший верстальщик сказал бы: «Нет». Зачем увеличивать размер изображения, если точно такую же линию можно сделать средствами CSS — просто задав нижнюю границу для пункта меню. Это не только уменьшит размер изображения. Веб-страница будет более точно совпадать с макетом, если посетитель откроет веб-страницу без картинок (например, отключив загрузку изображений для экономии трафика).

Точно то же касается и фона под пунктами меню. Для верстки этого фона достаточно картинки шириной 1 пиксел.

Но об этом мы еще поговорим, когда подробнее узнаем о работе с изображениями, а также о CSS.

Контейнеры

После того как макет нарезан и у вас есть отдельные изображения, можно начинать верстку. Прежде всего следует подумать, где будут отображаться текст и графические элементы. Ведь непосредственно в тег `body` помещать текст и изображения запрещено правилами последних версий HTML.

ВНИМАНИЕ

Непосредственно внутри тега body не должно быть ни текста, ни изображений. По правилам, любой текст и изображения должны находиться внутри парных тегов span, div, списков или таблиц.

Тег div (контейнер)

Первым тегом, который мы рассмотрим, будет тег `div`. Это блочный тег, который служит для единственной цели — быть контейнером, в котором могут находиться другие теги, изображения или текст.

Вам нужно установить фон или границу для отдельного участка веб-страницы? Поместите нужный участок в тег `div`, после чего установите фон и/или границу для этого `div`.

Необходимо сместить участок веб-страницы относительно самой страницы? Поместите нужный участок в тег `div`, после чего задайте ему относительное или абсолютное позиционирование с нужными координатами.

Следует отформатировать какой-либо блок текста? Поместите этот текст в `div`, и вы легко сможете произвести форматирование.

Итак, тег `div` служит для того, чтобы:

- применять какие-либо настройки к отдельному информационному блоку на сайте;
- быть контейнером для текста, изображений и других элементов HTML-документа.

Тег span (контейнер)

Как уже говорилось, тег `div` — блочный, то есть он занимает всю ширину веб-страницы, от ее начала и до конца. Поэтому ни слева, ни справа от содержимого тега `div` ничего отображаться не может. Элементы слева будут перенесены на строку выше тега `div`, а элементы справа от тега `div` — на строку ниже тега `div`.

Помимо тега `div`, существует еще один тег-контейнер — `span`. Он применяется для тех же целей, что и тег `div`, но имеет одно существенное отличие. А именно, тег `span` — строчный, то есть не требует для себя всей ширины окна браузера. Ему достаточно лишь той ширины, которую физически занимают все элементы, находящиеся в нем.

Благодаря этому тег `span` может использоваться для форматирования отдельных слов внутри абзаца (рис. 1.27).

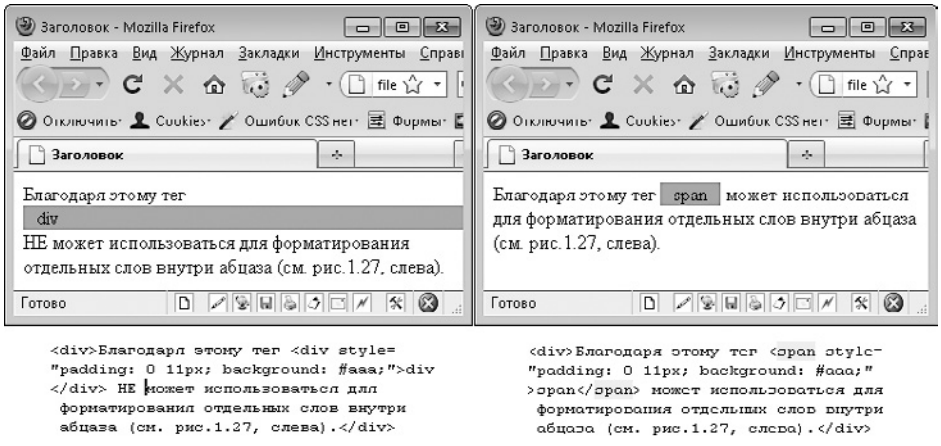


Рис. 1.27. Разница между тегами span и div

Верстаем макет

Но давайте ближе к делу. Нам ведь предстоит сверстать макет, о котором мы говорили выше. Еще раз посмотрим на него, но с некоторыми пояснениями (рис. 1.28).

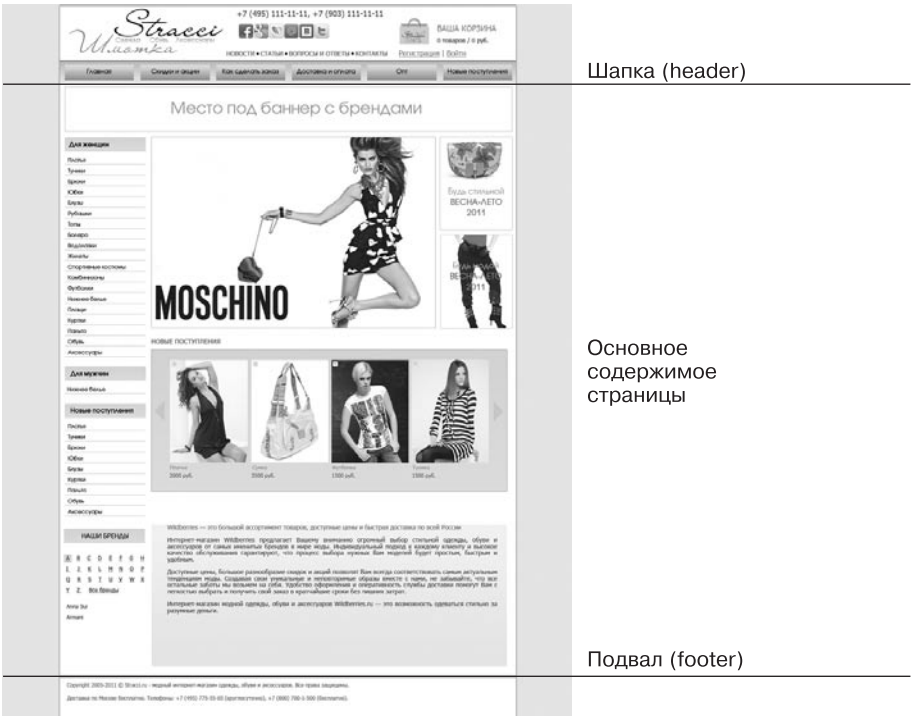


Рис. 1.28. Макет страницы состоит из шапки, футера и основного содержимого

Как правило, каждая веб-страница состоит из стандартного набора областей, для которых существуют специальные названия.

Верхняя часть страницы называется шапкой или хедером (header). Чаще всего на ней размещаются пункты главного меню, контактные данные, логотип, какая-нибудь картинка или слайд-шоу.

Нижняя часть страницы называется подвалом или футером (footer). В нее помещают различные счетчики, рейтинги, а также копирайт, ссылки на разработчиков и контактные данные.

Между футером и шапкой находится основная часть страницы.

Иногда слева от основной части страницы располагается отдельная информационная полоска — левая боковая панель (как на нашем макете).

Точно так же справа от основной части страницы может размещаться правая боковая панель.

Сейчас для нас самое главное — понять основную структуру нашего HTML-документа. После чего разбить его на отдельные части и создать для этих частей отдельные теги `div`. В результате у нас получится следующий HTML-код (листинг 1.3).

Листинг 1.3. Подготавливаем контейнеры для текста

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
</head>
<body>

<div>
<div>Шапка</div>
<div>Основная часть</div>
<div>Футер</div>
</div>

</body>
</html>
```

Мы создали отдельные контейнеры для шапки, футера и основного контента. Но у вас может возникнуть вопрос: почему перечисленные контейнеры находятся внутри родительского контейнера `div`?

В первую очередь, это сделано для удобства. Но главное — родительский контейнер нужен для того, чтобы центрировать содержимое страницы относительно самой страницы. Ведь посмотрите, на макете все содержимое страницы находится в центре, а по бокам от него размещается пустое пространство с фоном. Такая верстка называется фиксированной. Подробнее о фиксированной и резиновой верстках мы поговорим в главе 2, посвященной CSS.

Работаем с текстом

После того как мы подготовили контейнеры, можно приступать к наполнению веб-страницы текстом. Отдельные слова и фразы можно просто поместить в теги `div`

и span. Но если вам предстоит ввести большой объем текста, разбитого на абзацы, вам пригодится тег абзацев p.

Тег p (абзац)

Парный тег p предназначен для того, чтобы создать абзац текста:

```
<p>Это первый абзац</p><p>Это второй абзац</p>
```

Фактически, это означает, что весь текст, который заключен в тег p, будет начинаться с новой строки. И элементы, которые будут идти после закрывающего тега p, также будут начинаться с новой строки (рис. 1.29).

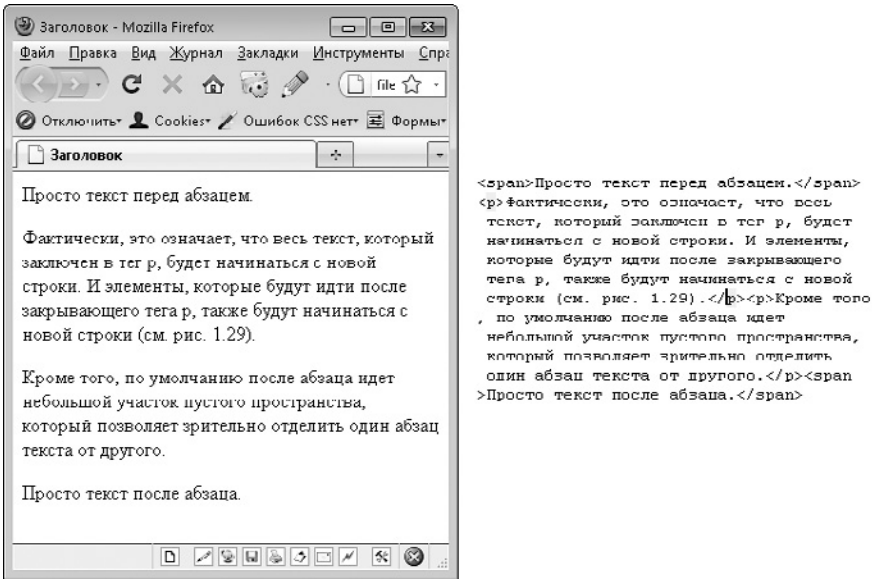


Рис. 1.29. Отображение абзацев на веб-странице

Кроме того, по умолчанию перед и после абзаца идет небольшой участок пустого пространства, который позволяет зрительно отделить один абзац текста от другого.

Тег br (перенос строки)

Тег p позволяет логически и физически отделить один абзац текста от другого. Но что делать, если внутри абзаца нужно перенести текст на новую строку? Специально для этих целей предназначен одинарный тег br. Как и все одинарные теги, он не имеет закрывающего тега и записывается следующим образом: <br />.

Одним из видимых с первого взгляда отличий данного тега от p является отсутствие отступов после и перед тегом br (рис. 1.30).

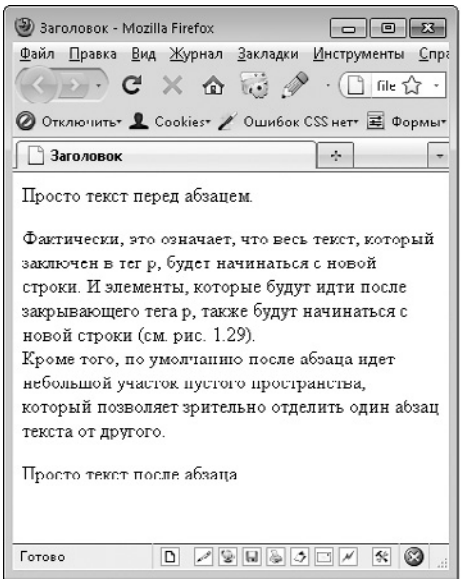


Рис. 1.30. После тега br и перед ним нет никаких отступов

Как вы уже поняли, тег br применяется к тексту, расположенному внутри тега p, тегов-контейнеров, списков, таблиц и т. д. Применение тега br не избавляет вас от необходимости помещать текст в какой-либо контейнер.

Теги h1–h6 (заголовки)

Какой текст может обойтись без заголовка? Понимали это и разработчики языка HTML. Для создания заголовков они выделили целых шесть тегов: h1, h2, h3, h4, h5 и h6. Все перечисленные теги подобны друг другу (являются парными блочными тегами), но позволяют создавать заголовки разных уровней.

Тег h1 — это заголовок первого уровня, то есть самый большой и заметный заголовок из всех.

Тег h2 — заголовок второго уровня. Он меньше заголовка первого уровня, но больше заголовков третьего и других уровней. И так далее (рис. 1.31).

При верстке документов не рекомендуется перескакивать между заголовками разных уровней. То есть если у вас в тексте есть несколько вложенных друг в друга заголовков, то верхний заголовок должен формироваться тегом h1, подзаголовок —

тегом h2, а подзаголовок подзаголовка — тегом h3. А не, например, h1, потом h3, а потом h4 — в этой цепочке явно пропущен заголовок второго уровня.

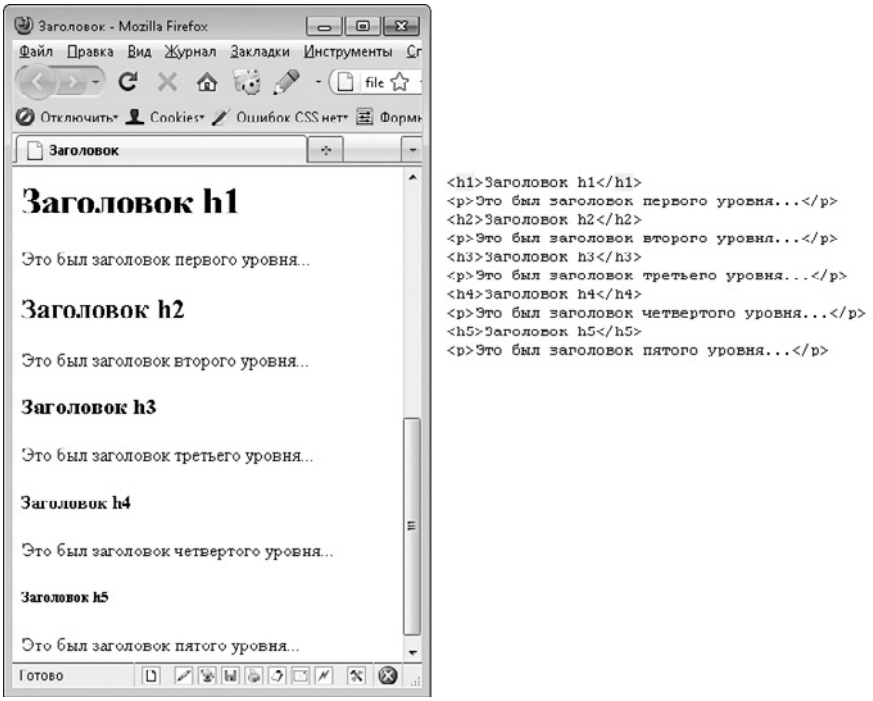


Рис. 1.31. Разные уровни заголовков

Тег center (центрирование)

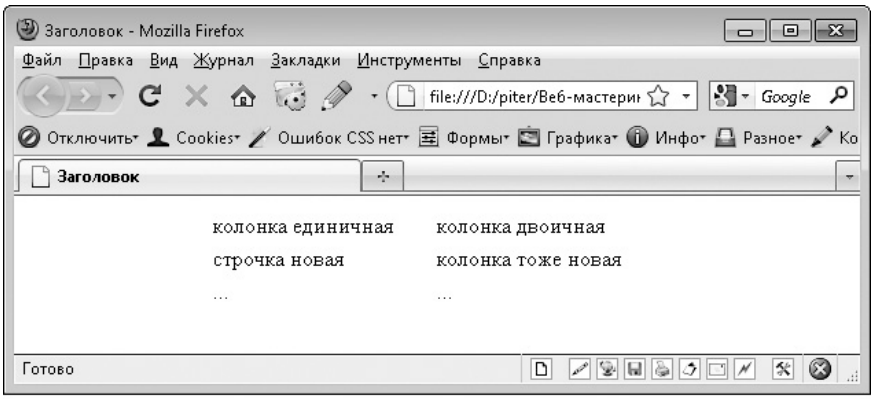
Сразу хочется предупредить, что тег center вне закона. Стандартами последних версий HTML он запрещен (вместо него используется свойство CSS). Но знать его все-таки необходимо, так как в редких случаях center — это единственный способ центрировать что бы то ни было.

Парный тег center применяется не к тексту, а к контейнеру и всему его содержимому. Будь то контейнер div, абзац p, таблица или просто рисунок. Однако с центрированием контейнеров div и абзацев p проблем никогда не возникает, а вот с таблицами другое дело (рис. 1.32).

Теги strong и em (полужирное и курсивное начертания)

Каждый, кто работал с текстом в программе Microsoft Word, знает, что, помимо обычного текста, существует также полужирный и курсивный текст. Эти варианты

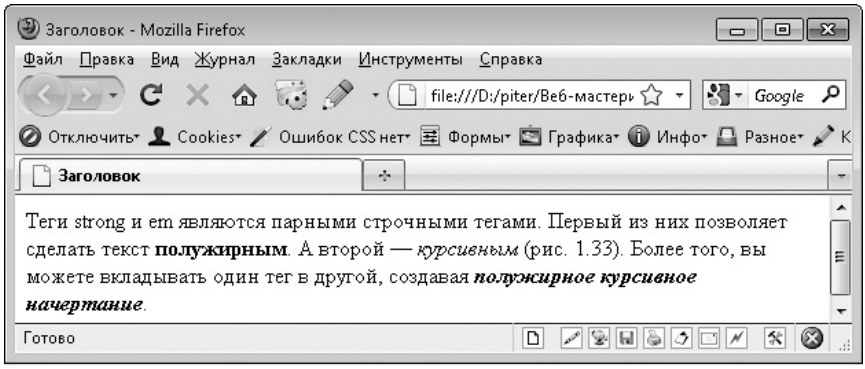
начертания настолько вошли в нашу жизнь, что разработчики языка HTML решили создать для них два отдельных тега: `strong` и `em`.




```
<center><table style="width: 333px;">
<tr><td>колонка единичная</td><td>колонка двоичная</td></tr>
<tr><td>строчка новая</td><td>колонка тоже новая</td></tr>
<tr><td>...</td><td>...</td></tr>
</table></center>
```

Рис. 1.32. Центрирование таблицы

Теги `strong` и `em` являются парными строчными тегами. Первый из них позволяет сделать текст полужирным. А второй — курсивным (рис. 1.33). Более того, вы можете вкладывать один тег в другой, создавая полужирное курсивное начертание. Однако будьте осторожны — не перепутайте вложенность тегов (то есть не перепутайте местами расположение закрывающих тегов).




```
Теги strong и em являются парными строчными тегами. Первый из них
 позволяет сделать текст <strong>полужирным</strong>. А второй — <em
>курсивным</em> (рис. 1.33). Более того, вы можете вкладывать один тег в
 другой, создавая <strong><em>полужирное курсивное начертание</em></
/strong>.
```

Рис. 1.33. Полужирное и курсивное начертания текста

Теги sub и sup (подстрочный и надстрочный)

Мягко говоря, это не самые часто употребляемые теги. Но если уж они вам понадобятся, вы перероете весь Интернет, но не остановитесь, пока не найдете их описание. Ведь они делают такое, чего нельзя сделать ни одним другим способом, ни с помощью HTML, ни с помощью CSS или JScript.

Эти парные строчные теги позволяют создать подстрочный или надстрочный текст. Как самый простой пример — обозначение квадратных метров. Как вы собираетесь указывать двоечку после буквы «м» без помощи тега надстрочного текста `sup`? А с его помощью это очень даже просто: `м<sup>2</sup>` (рис. 1.34).

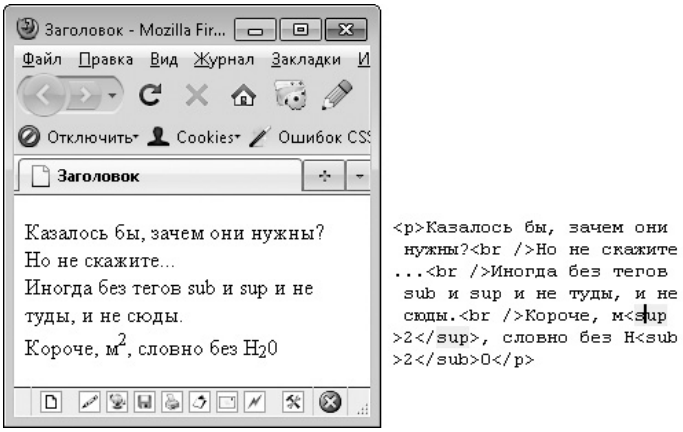


Рис. 1.34. Надстрочный и подстрочный текст

Тег pre (код)

Надеюсь, вы еще не забыли, что в языке HTML переносы строк, идущие подряд пробелы и табуляции игнорируются? Это правило, но у каждого правила есть исключения. И исключением для данного правила выступает парный блочный тег `pre`.

Все, что записано внутри тега `pre`, будет выводиться на экран браузера как есть: со всеми пробелами, табуляциями и переносами строк (рис. 1.35).

Обратите также внимание, что внутри тега `pre` не перестают действовать другие правила языка HTML. В частности, не стоит забывать о запрете использования символов <, >, `&` в тексте HTML-документа. Даже внутри тега `pre` их нужно заменять соответствующими спецсимволами (`<`, `>`, `&`).

Тег `pre` удобно использовать для вставки в HTML-документ строк кода на любом языке программирования.

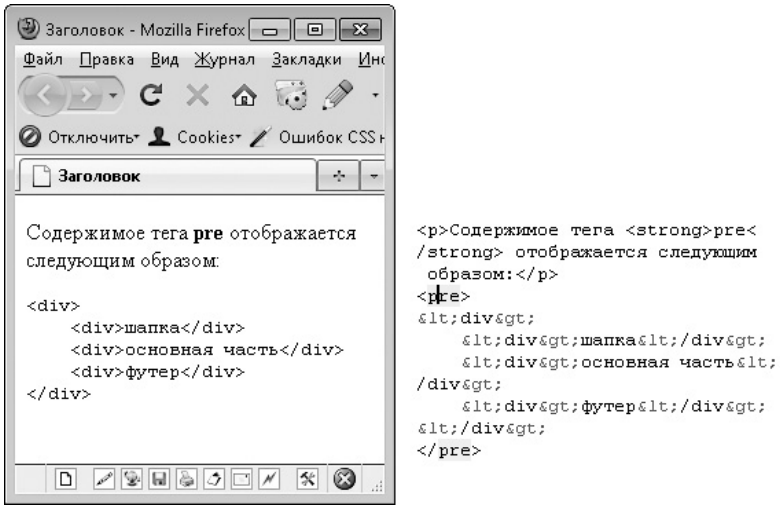


Рис. 1.35. Использование тега pre

Комментарии

Мы с вами изучили уже столько тегов. И между делом совершенно забыли о самом важном теге — теге комментариев. Например, что делать, если нужно добавить в HTML-документ комментарий, поясняющий определенный участок кода?

Комментирование весьма поощряется разработчиками HTML, поэтому обязательно выучите следующий весьма необычный тег: `<!-- -->`. Комментарий пишется между `<!--` и `-->`. Все, что находится между этими стрелочками, не будет отображаться браузером.

Например:

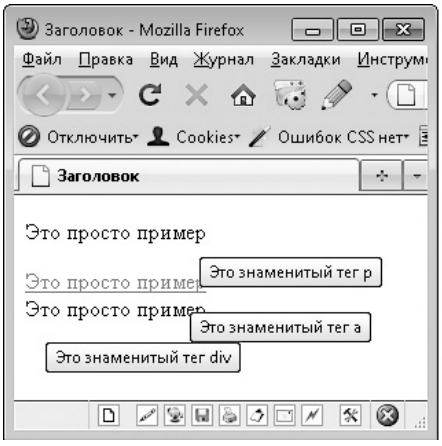
```
<!-- это наш комментарий -->
```

Атрибут title

Говоря о тексте, не следует также забывать об атрибуте `title`, который позволяет добавить описание к любому элементу веб-страницы, то есть всплывающую подсказку, которая будет отображаться при наведении указателя мыши на данный элемент (рис. 1.36).

Это необязательный атрибут, то есть вы можете либо указывать его, либо нет. Атрибут `title` можно использовать практически для любого тега, который может быть вложен внутрь тега `body`. Если вам интересен полный список тегов, для которых

можно указывать данный атрибут, то вот он: a, abbr, acronym, address, applet, area, b, basefont, bdo bgsound, big, blockquote, body, br, button, caption, center, cite, code, col, colgroup, dd, del, dfn, dir, div, dl, dt, em, embed, fieldset, font, form, frame, h1, h2, h3, h4, h5, h6, hr, i, iframe, img, input, ins, isindex, kbd, label, legend, li, link, map, marquee, menu, nobr, object, ol, option, p, plaintext, pre, q, s, samp, select, small, span, strike, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, u, ul, var, wbr, xmp. Думаю, многие из этих тегов вам уже знакомы. Хотя это только официальный список. На самом деле атрибут `title` поддерживают и многие другие теги.




```
<p title="Это знаменитый тег p"
>Это просто пример</p>
<a title="Это знаменитый тег a"
 href="#">Это просто пример</a>
<div title="Это знаменитый тег
 div">Это просто пример</div>
```

Рис. 1.36. Отображение атрибута title

Верстаем макет — добавляем текст

Мы с вами узнали что-то новое. А значит, пора применить новые знания на практике. Так что достанем наш макет и верстку страницы и добавим в нее текст (листинг 1.4).

Листинг 1.4. Верстаем макет — добавляем текст

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
</head>
<body>

<div>
```

```
<div><!-- Шапка начинается -->
    <div>+7 (495) 111-11-11, +7 (903) 111-11-11</div>
    <h3>Ваша корзина</h3>
    <div>0 товаров / 0 руб.</div>
</div><!-- Шапка заканчивается -->

<div><!-- Основная часть начинается -->
    <div>
    <h3>Новые поступления</h3>
    </div>
    <div>
    <h3>Wildberries — это большой ассортимент товаров, доступные цены и быстрая доставка по всей России</h3>
    <p>Интернет-магазин Wildberries предлагает вашему вниманию огромный выбор стильной одежды, обуви и аксессуаров от самых именитых брендов в мире моды. Индивидуальный подход к каждому клиенту и высокое качество обслуживания гарантируют, что процесс выбора нужных вам моделей будет простым, быстрым и удобным.</p>
    <p>Доступные цены, большое разнообразие скидок и акций позволят вам всегда соответствовать самым актуальным тенденциям моды. Создавая свои уникальные и неповторимые образы вместе с нами, не забывайте, что все остальные заботы мы возьмем на себя. Удобство оформления и оперативность службы доставки помогут вам с легкостью выбрать и получить свой заказ в кратчайшие сроки без лишних затрат.</p>
    <p>Интернет-магазин модной одежды, обуви и аксессуаров Wildberries.ru — это возможность одеваться стильно за разумные деньги.</p>
    </div>
</div><!-- Основная часть заканчивается -->

<div><!-- Футер начинается -->
    <p>Copyright 2005-2011 © Stracci.ru — модный интернет-магазин одежды, обуви и аксессуаров. Все права защищены.</p>
    <p>Доставка по Москве бесплатно. Телефоны: +7 (495) 775-55-05 (круглосуточно), +7 (800) 700-1-500 (бесплатно).</p>
</div><!-- Футер заканчивается -->
</div>

</body>
</html>
```

Итак, что у нас получилось (рис. 1.37)?

Что-то не очень похоже на макет, правда? Я предупреждал, что верстка — это сложный, трудоемкий и высокоинтеллектуальный процесс. Но не огорчайтесь,

возможно, после добавления изображений в следующем разделе будет больше сходства.

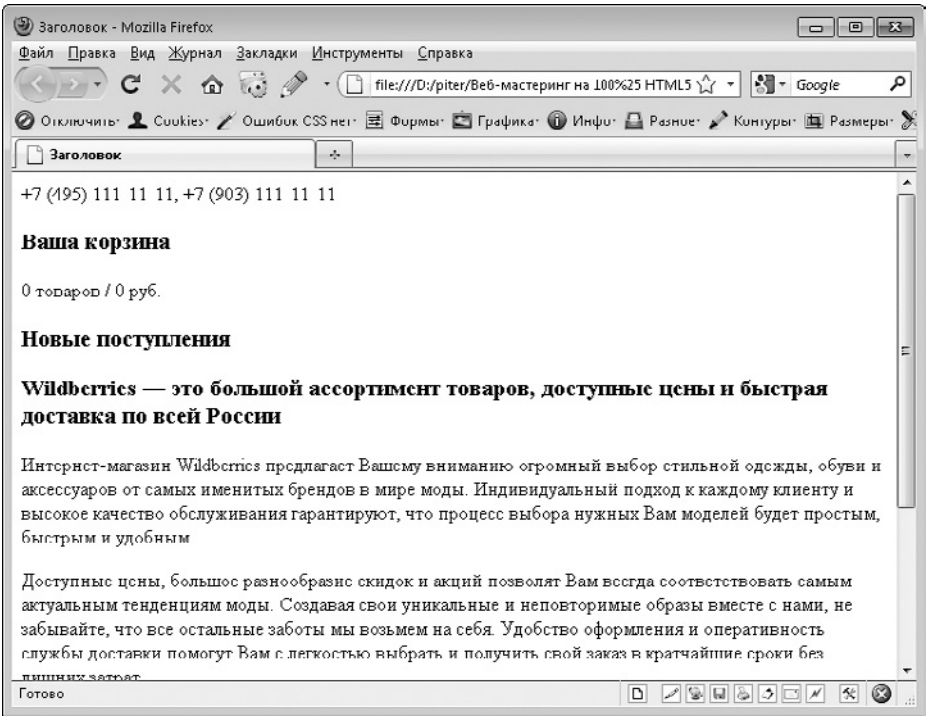


Рис. 1.37. Добавляем текст в макет

Изображения и ссылки

Перефразируя классика, можно с уверенностью сказать, что «изображения правят миром». Ни одна статья ни в одном популярном журнале не обходится без изображений. Что уж тут говорить о рекламе.

Изображения используются везде. Они дают возможность быстро представить то, что описывается в тексте. Они позволяют сконцентрировать внимание на нужной фразе и донести нужную мысль. Они просто делают окружающий текст более привлекательным для чтения.

Что уж говорить об Интернете, если даже редакторы журналов требуют от своих авторов, чтобы те вставляли в статьи как можно больше изображений.

И сейчас мы с вами постигнем тайные знания. Научимся добавлять изображения на веб-страницу.

Тег img (изображение)

На самом деле ничего сложного в добавлении изображения на веб-страницу нет. Делается это одним-единственным тегом — `img`. Это одинарный строчный тег, то есть записывается он следующим образом: `<img />`. Располагаться тег `img` должен в каком-либо контейнере — непосредственно в теге `body` размещать его нельзя. Вообще, это же можно сказать обо всех строчных тегах.

У тега `img` — несколько атрибутов, среди которых есть обязательные.

Атрибут src. Собственно, по-настоящему обязательный атрибут только один — это атрибут `src`. Именно он позволяет указать путь к изображению, которое должно быть отображено в HTML-документе. В качестве изображения могут выступать GIF-, PNG-, JPG- и JPEG-файлы. Как их создавать, мы уже рассматривали выше.

Обратите внимание, что в качестве значения атрибута `src` могут выступать лишь те изображения, которые находятся в Интернете: на вашем хостинге либо любом стороннем сайте (который не запрещает вывод своих изображений на других сайтах Интернета). Изображения, которые хранятся на вашем компьютере, вывести на сайте не удастся. Сначала их нужно выложить в Интернете.

Атрибут alt. Еще один атрибут, который нужно указывать в теге `img`, атрибут `alt`. Он позволяет добавить текст, который будет отображаться вместо картинки, если по каким-либо причинам изображение загрузить не удалось (рис. 1.38). Кроме того, атрибут `alt` тега `img` используется поисковыми системами для поиска по содержимому картинок в Интернете.

Рис. 1.38. Вставляем изображения

Обратите внимание: даже если вам нечего сказать об изображении, атрибут `alt` все равно нужно указывать, присваивая ему пустое значение: `alt=""`. Иначе валидатор обнаружит изображение без тега `alt` и сообщит об ошибке.

Другие атрибуты. Тегу `img` могут быть присвоены и другие атрибуты. Однако все их можно заменить с помощью CSS. Поэтому сейчас рассматривать их мы не будем. Или почти не будем...

Атрибуты width и height. Атрибуты `width` и `height` можно задавать многим тегам, включая `img`. Однако для всех этих тегов данные атрибуты считаются устаревшими. Для всех, кроме тега `img`.

Когда браузер открывает веб-страницу, он сначала скачивает сам HTML-документ, потом прикрепленные к нему CSS и JS-файлы и, конечно, изображения. Все это занимает определенное время. И чтобы все это время страница браузера не была пустой, браузер начинает потихоньку выводить на экран веб-страницу. Казалось бы, все хорошо. Но вот беда, когда браузер встречает в коде тег `img`, перед ним встает вопрос, каких же размеров встреченное изображение? Ведь само изображение еще не скачано. Чтобы точно указать ширину и высоту изображения, верстальщики часто задают размеры изображения с помощью атрибутов `width` и `height`. Это считается хорошим тоном.

Указание атрибутов `width` и `height` позволяет также ограничить пространство, выделяемое под изображение и альтернативный текст, который задан в атрибуте `alt`. Ведь если изображение небольшое (например, значок), а альтернативный текст для него не состоит из нескольких букв, то при отображении альтернативного текста он может сдвинуть близлежащие к изображению элементы. И верстка может поплыть.

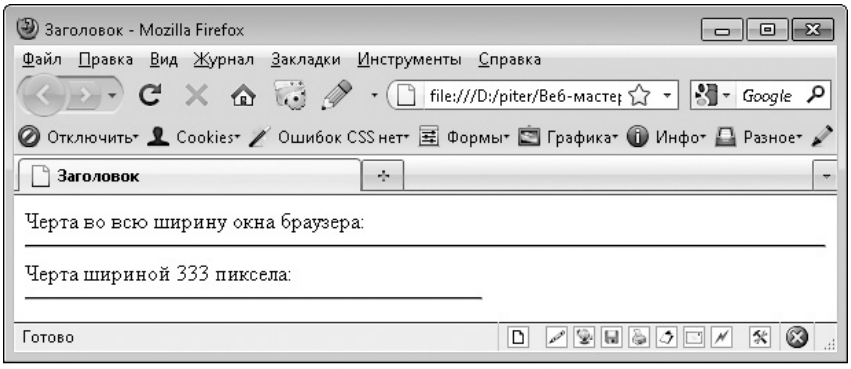
Общий формат тега. Рассмотрим общий синтаксис тега `img`:

```
<img alt="описание" src="путь к файлу" />

<img alt="" src="путь к файлу" />
```

Тег hr (горизонтальная линия)

Тег `hr` сложно отнести к изображениям. Но куда-то ведь его нужно вставить, так как пусть и редко, но он все-таки используется.

Данный блочный одинарный тег позволяет вставить горизонтальную черту, ширина которой равна либо всей ширине контейнера, в который вложен тег, либо ширине окна браузера (рис. 1.39).




```
<div>Черта во всю ширину окна браузера:</div><hr />
<div>Черта шириной 333 пиксела:</div><div style="width: 333px;"><hr /></div>
```

Рис. 1.39. Создание горизонтальной черты

Тег a (ссылка)

Ссылки являются одним из основополагающих моментов Интернета. Без них Интернет никогда не стал бы настолько популярным. Если бы не было ссылок, вам бы все время приходилось вводить адрес новой страницы в адресной строке браузера.

Ссылки предназначены для того, чтобы связывать друг с другом отдельные страницы как одного сайта, так и самых разных сайтов в Интернете. Фактически ссылка представляет собой текст, при щелчке на котором открывается заданная в атрибутах ссылки веб-страница.

Чтобы вставить в HTML-документ ссылку, достаточно воспользоваться парным строчным тегом `a`. При этом главное — не забыть указать один обязательный атрибут данного тега — `href`. В качестве значения данного атрибута выступает адрес веб-страницы, которая должна открываться при щелчке на данной ссылке. Если же вы хотите, чтобы при щелчке на ссылке ничего не происходило (в некоторых случаях это необходимо), то в качестве значения атрибута `href` можно указать знак диез — `#`.

Варианты открытия ссылки. Помимо атрибута `href`, тегу `a` можно присваивать еще один интересный атрибут — `target`. От значения данного атрибута зависит, в каком окне откроется веб-страница, указанная в ссылке.

По умолчанию страница открывается в том же окне браузера, в котором находилась ссылка, то есть заменяет собой открытую веб-страницу. Однако если вы присвоите атрибуту `target` значение `_blank`, страница будет открываться в новом окне (на новой вкладке браузера, если браузер поддерживает вкладки).

Помимо значения _blank, атрибуту target можно присваивать следующие значения:

- _self — открыть страницу в том же окне (то есть как и без указания атрибута target);
- _parent — используется при работе с фреймами и позволяет открыть страницу в родительском фрейме; если на текущей странице нет фреймов, выполняется действие, аналогичное _self;
- _top — применяется при работе с фреймами и позволяет открыть страницу в окне браузера, заменяя собой все фреймы, которые были на странице; если на текущей странице нет фреймов, то действие аналогично значению _self.

Синтаксис. Основной синтаксис тега a следующий:

```
<a href="адрес">текст</a>

<a target="_blank" href="адрес">текст</a>
```

Создание якорей (закладок). Создание ссылок на внешние страницы в Интернете — это основное назначение тега a. Но, помимо данной функциональности, у тега a есть еще одна очень полезная функция. Он позволяет создавать закладки на самой странице, то есть сделать так, чтобы при щелчке на ссылке страница в окне браузера перелистывалась вниз/вверх до тех пор, пока не появится элемент, на который ссылалась ссылка. Если вы когда-нибудь пользовались закладками, например, в программе Acrobat Reader, то должны ясно представлять себе, что такое якоря (закладки) в HTML.

Итак, чтобы создать закладку, достаточно в качестве значения атрибута href тега a указать следующее: #идентификатор_элемента, где в качестве идентификатора элемента выступает значение атрибута id какого-либо тега на веб-странице (листинг 1.5).

Листинг 1.5. Создание якорей

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
</head>
<body>
<div id="goto_top"><a href="#goto_bottom">Вниз</a></div>
<div>
<p>Много-много...</p>
```

```
<p>Много-много...</p>
<p>Много-много...</p>
<p>Много-много...</p>
<p>Много-много...</p>
<p>Много-много...</p>
<p>...нашего текста</p>
</div>
<div id="goto_bottom"><a href="#goto_top">Вверх</a></div>
</body>
</html>
```

На рис. 1.40 можно видеть одно из основных применений якорей — создание закладок для быстрого перехода в начало и конец веб-страницы. Закладки также часто используются для создания оглавления статьи.

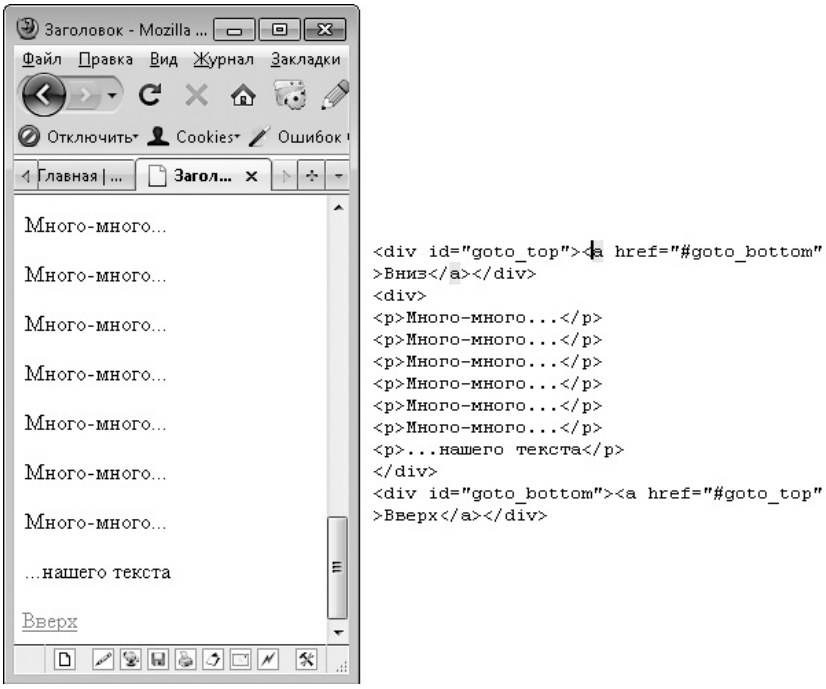


Рис. 1.40. Создание якорей (закладок) в HTML-документе

Помимо создания закладок на элементы той же страницы, вам ничто не мешает создать закладку на элемент любой другой веб-страницы в Интернете. Для этого достаточно в качестве значения атрибута `href` указать адрес страницы и идентификатор закладки: `<a href="http://microsoft.com#onfooter">контактные`

данные. При щелчке на такой закладке сначала откроется указанная страница. А после полного открытия страницы браузер попытается найти на странице и отобразить элемент с указанным идентификатором.

Ссылки на файлы. В качестве значения атрибута `href` тега `a` может выступать не только адрес какой-либо веб-страницы или якоря. Вы также можете ссылаться на картинки, программы и любые другие файлы в Интернете. В этом случае только от вашего браузера и установленных для него плагинов зависит, что именно будет происходить при щелчке на ссылке. Например, попробуем сослаться на файлы со следующими расширениями:

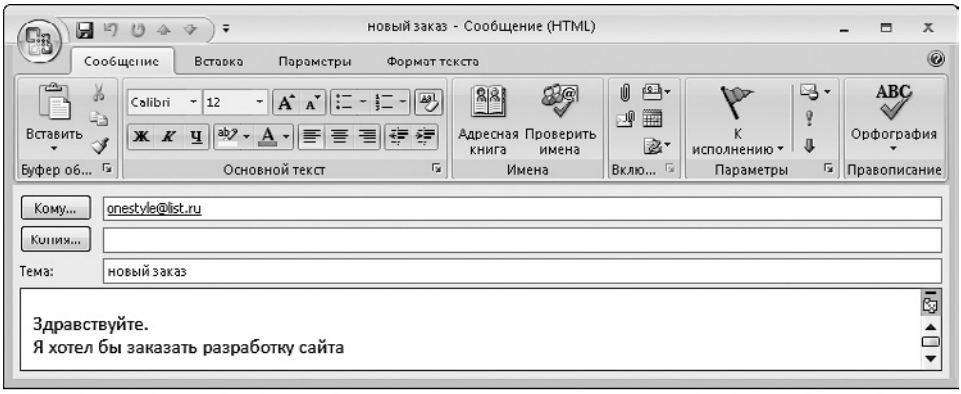
- HTM, HTML, PHP, CGI — будет открыта данная веб-страница (как вариант, выполнен данный сценарий);
- TXT — будет открыт данный текстовый файл (конечно, с потерей переносов на новую строку, табуляций и идущих подряд пробелов);
- JPG, JPEG, GIF, PNG — в окне браузера откроется указанное изображение; нажав сочетание клавиш Ctrl+S, вы сможете сохранить изображение на компьютере;
- SWF — если в браузере установлен плагин Adobe Flash, будет воспроизведен данный флеш-ролик, в противном случае браузер предложит сохранить SWF-файл на вашем компьютере;
- PDF — если в браузере установлен соответствующий плагин, то в окне браузера откроется содержимое PDF-документа с панелями инструментов программы для чтения PDF-файлов;
- EXE, DOC, DOCX, XLS, XSLX, ZIP, RAR — браузер предложит открыть данный файл в нужной программе либо сохранить его на компьютере;
- другие расширения — если браузер не знает, что делать с файлами данного формата, он предложит сохранить файл на диске.

Ссылки на электронные адреса. Одной из полезных возможностей тега `a` является возможность ссылаться на электронные адреса. В этом случае при щелчке на такой ссылке у посетителя будет открываться почтовый клиент по умолчанию с уже открытым окном для создания и отправки письма по электронному адресу, который был указан в значении атрибута `href` (рис. 1.41).

Самый простой формат ссылки на электронный адрес следующий:

```
<a href="mailto:onestyle@list.ru">Нажми меня</a>
```

При щелчке на такой ссылке откроется окно создания электронного письма, в котором в качестве получателя e-mail будет указан адрес onestyle@list.ru.




```
<div><a href="mailto:onestyle@list.ru?subject=новый%20заказ&body=Здравствуйте.%0AЯ
%20хотел%20бы%20заказать%20разработку%20сайта">заказать разработку сайта</a></div>
```

Рис. 1.41. Результат нажатия ссылки, код которой приведен под окном программы Outlook

Указанным выше форматом пользуются чаще всего. Но на этом возможности тега a по отправке электронных писем не заканчиваются. Ведь после электронного адреса можно указывать и другие данные. Основной формат записи в этом случае:

```
<a href="mailto:onestyle@list.ru?ключ=значение">Нажми меня</a>
```

```
<a href="mailto:onestyle@list.ru?ключ=значение&ключ2=значение2">
Нажми меня</a>
```

В качестве ключа могут выступать следующие ключевые слова:

- `subject` — тема письма;
- `body` — текст письма;
- `cc` — электронный адрес, на который будет отправлена копия письма;
- `bcc` — скрытый электронный адрес, на который будет отправлена копия письма.

Если в значении какого-либо из ключевых слов есть пробелы, их нужно заменить символами `%20`. В значении можно также указывать переводы строк — для этого используется последовательность `%0A`.

Пример (см. рис. 1.41):

```
<a href="mailto:onestyle@list.ru?subject=новый%20заказ&body=
Здравствуйте.%0AЯ%20хотел%20бы%20заказать%20разработку%20сайта">
заказать разработку сайта</a>.
```

Ссылки на другие протоколы. Продвинутый пользователь ОС Windows и Интернета, возможно, уже догадался, что `http` и `mailto`, которые указываются в качестве части значения атрибута `href`, являются протоколами или обработчиками, определяющими, какой именно программе браузер должен передать ссылку для открытия. А значит, возможности тега `a` действительно безграничны. Ведь каждая установленная на компьютере пользователя программа может создать свой протокол, после чего у тега `a` появится новый функционал.

И такие программы уже есть. К ним относятся довольно популярные приложения:

- Skype — при щелчке на теге `a` с атрибутом `href`, для которого установлено значение `skype:логин`, программа Skype позвонит указанному логину;
- ICQ — при щелчке на теге `a` с атрибутом `href`, для которого установлено значение `icq:номер`, клиент ICQ добавит указанный контакт;
- Webmoney Keeper Classic — для работы с данным электронным кошельком используется протокол `wmk`. Он позволяет выполнять следующие операции:
  - `wmk:refresh` — обновить данные в запущенной в данный момент программе Webmoney Keeper Classic;
  - `wmk:exit` — выйти из программы;
  - `wmk:payto?purse=кошелек&amount=сумма&desc=описание` — открыть окно Передать WM с заполненными данными;
  - `wmk:msgto?to=WMID&subject=тема_письма&msg=текст_письма` — отправить сообщение указанному WMID;
  - `wmk:explore?url=страница&title=заголовок` — открыть в окне с указанным заголовком веб-страницу;
  - `wmk:display?window=окно&page=страница_диалога&param=параметры_для_работы+окна` — открыть окно, приведенное в значении ключевого слова `window` на указанной странице. В качестве значения ключевого слова `window` могут выступать следующие слова: `main` (окно программы), `options` (диалог Параметры программы), `props` (диалог О себе или Информация о корреспонденте, если в параметрах указан WMID), `about` (диалог О программе), `messages` (диалог Список всех сообщений), `invoices_out` (диалог Список выписанных вами счетов), `invoices_in` (диалог Список счетов, предъявленных вам для оплаты), `operations` (окно История операций).

Обратите внимание: как правило, чтобы одна из вышеперечисленных ссылок работала, у пользователя должна быть запущена программа, обрабатывающая команды, которые поступают от ссылки.

Адреса в Интернете

Мы изучили уже два тега, в атрибутах которых нужно указывать какой-либо адрес в Интернете (`img` и `a`). Пришла пора поговорить о самих адресах.

Существует два типа адресов в Интернете: абсолютные и относительные. рассмотрим их подробнее.

Абсолютные адреса. Абсолютным называется адрес, который состоит из протокола (http:// или https://), имени сайта (например, microsoft.com), пути к веб-странице и имени самой веб-страницы (например, index.htm), которую нужно открыть.

Между протоколом и именем сайта может идти префикс www. Сравните:

- http://microsoft.com/index.htm;
- http://www.microsoft.com/index.htm.

Как правило, ссылка с www и ссылка без www ведут на одну и ту же страницу. Но так бывает не всегда. Могут также встречаться следующие ситуации:

- ссылка без www открывается, а страницу с www в адресе браузер вообще не находит и не может отобразить; данная ситуация может возникать при намеренной или неправильной настройке веб-сервера;
- обратная ситуация также встречается: ссылка с www открывается, а адрес без www браузер найти не может;
- по адресам с www и без www открывается одна и та же страница, но для каждой из них используется своя сессия; то есть, например, по адресу с www вы можете быть зарегистрированным на сайте пользователем, а по адресу без www окажетесь анонимным.

Между адресом сайта и именем веб-страницы может располагаться путь к веб-странице, то есть путь к папке, в которой находится данная веб-страница. Ведь не всегда HTML-документы и другие файлы размещаются в корне сайта. Сравните:

- http://dev.biz.ua/index.php — ссылка на страницу в корне сайта;
- http://dev.biz.ua/files/index.php — ссылка на страницу, которая находится в папке files корня сайта;
- http://dev.biz.ua/sites/files/index.php — ссылка на страницу, находящуюся в папке files, которая, в свою очередь, располагается в папке sites, и только папка sites находится в корне сайта.

Абсолютные адреса используются как для открытия страниц текущего сайта, так и для открытия страниц любого другого сайта в Интернете.

Относительные адреса. Относительные адреса позволяют обратиться к любому файлу, который находится внутри корневой папки сайта и расположенных в ней подпапках. Но с его помощью нельзя открыть страницы и файлы, размещенные на других сайтах Интернета.

Относительный адрес состоит из пути к файлу и имени файла. Причем от того, как начинается относительный адрес, зависит способ определения местоположения указанной в адресе страницы. Существуют следующие варианты относительных адресов.

- `/sites/index.php` — файл `index.php` находится в папке sites, которая расположена в корне открытого в данный момент сайта. Этот относительный адрес аналогичен абсолютному адресу http://текущий_сайт/sites/index.php. Следует обратить внимание на то, что данный вариант относительного адреса работает лишь для сайтов, расположенных в Интернете или на локальном веб-сервере. Так что он не будет работать в HTML-документе, который вы просто создали у себя на компьютере.
- `sites/index.php` — файл `index.php` находится в папке sites, которая расположена там же, где и открытая в данный момент страница. То есть если указанная ссылка находится на странице http://dev.biz.ua/files/about.php, то после щелчка на ней откроется страница http://dev.biz.ua/files/sites/index.php. Если же ссылка располагается на странице http://dev.biz.ua/about.php, то откроется страница http://dev.biz.ua/sites/index.php.
- `../sites/index.php` — файл `index.php` находится в папке sites, которая, в свою очередь, размещается в папке, являющейся родительской к папке, в которой располагается страница, загруженная в данный момент в браузер. То есть если указанная ссылка находится на странице http://dev.biz.ua/files/about.php, то после щелчка на ней откроется страница http://dev.biz.ua/sites/index.php.

  Конструкция `..` говорит о необходимости подняться на один уровень выше, чем текущее расположение веб-страницы. При этом в адресе может находиться любое количество конструкций `...` Например, адрес `../../sites/index.php` указывает на необходимость подняться на два уровня выше текущего расположения веб-страницы.

Верстаем макет — добавляем изображения

Вернемся к нашему макету. Возможно, добавление в HTML-документ картинок сделает нашу страницу более похожей на макет (листинг 1.6).

Листинг 1.6. Верстаем макет — добавляем в HTML-документ изображения

```
<div>
<div><!-- Шапка начинается -->
```

```
    <a href="/"><img alt="" src="img/logo.png" /></a>
    <div>+7 (495) 111-11-11, +7 (903) 111-11-11</div>
    <div>
        <a href="#"><img alt="facebook" src="img/btn_facebook.png"
           width="32px" height="32px" /></a>
        <a href="#"><img alt="google" src="img/btn_google.png" width="32px"
           height="32px" /></a>
        <a href="#"><img alt="livejournal" src="img/btn_livejournal.png"
           width="32px" height="32px" /></a>
        <a href="#"><img alt="mailru" src="img/btn_mailru.png" width="32px"
           height="32px" /></a>
        <a href="#"><img alt="vkontakte" src="img/btn_vkontakte.png"
           width="32px" height="32px" /></a>
        <a href="#"><img alt="twitter" src="img/btn_twitter.png"
           width="32px" height="32px" /></a>
    </div>
    <div>
        <a href="#"><img alt="" src="img/cart.png" width="66px"
           height="59px" /></a>
        <h3>Ваша корзина</h3>
        <div>0 товаров / 0 руб.</div>
    </div>
</div><!-- Шапка заканчивается -->

<div><!-- Основная часть начинается -->
    <div>
        <a href="#"><img alt="" src="img/banner3.jpg" width="637px"
           height="428px" /></a>
        <a href="#"><img alt="" src="img/banner2.jpg" width="162px"
           height="208px" /></a>
        <a href="#"><img alt="" src="img/banner1.jpg" width="162px"
           height="208px" /></a>
    </div>
    <div>
    <h3>Новые поступления</h3>
    <div>
        <img alt="назад" src="img/rotator_left.png" width="23px"
```

```
            height="43px" />
        <a href="#"><img alt="" src="img/product1.jpg" width="173px"
            height="230px" /></a>
        <a href="#"><img alt="" src="img/product2.jpg" width="173px"
            height="230px" /></a>
        <a href="#"><img alt="" src="img/product3.jpg" width="173px"
            height="230px" /></a>
        <a href="#"><img alt="" src="img/product4.jpg" width="173px"
            height="230px" /></a>
        <img alt="вперед" src="img/rotator_right.png" width="23px"
            height="43px" />
    </div>
    </div>
    <div>
    <h3>Wildberries — это большой ассортимент товаров, доступные цены и быстрая
доставка по всей России</h3>
    <p>Интернет-магазин Wildberries предлагает вашему вниманию огромный выбор
стильной одежды, обуви и аксессуаров от самых именитых брендов в мире моды.
Индивидуальный подход к каждому клиенту и высокое качество обслуживания гаранти-
руют, что процесс выбора нужных вам моделей будет простым, быстрым и удобным.</p>
    <p>Доступные цены, большое разнообразие скидок и акций позволят вам всегда
соответствовать самым актуальным тенденциям моды. Создавая свои уникальные
и неповторимые образы вместе с нами, не забывайте, что все остальные заботы мы
возьмем на себя. Удобство оформления и оперативность службы доставки помогут вам
с легкостью выбрать и получить свой заказ в кратчайшие сроки без лишних за-
трат.</p>
    <p>Интернет-магазин модной одежды, обуви и аксессуаров Wildberries.ru — это
возможность одеваться стильно за разумные деньги.</p>
    </div>
</div><!-- Основная часть заканчивается -->


<div><!-- Футер начинается -->
    <p>Copyright 2005-2011 © Stracci.ru — модный интернет-магазин одежды, обуви
и аксессуаров. Все права защищены.</p>
    <p>Доставка по Москве бесплатно. Телефоны: +7 (495) 775-55-05 (круглосуточно),
+7 (800) 700-1-500 (бесплатно).</p>
</div><!-- Футер заканчивается -->
</div>
```

Если посмотреть на рис. 1.42, то можно заметить, что сходства с макетом стало еще меньше. Но ведь впереди еще много страниц книги. Возможно, в следующих главах все изменится к лучшему.

Рис. 1.42. Верстаем макет: добавляем изображения

Кроме того, на рисунке можно заметить границы непонятного цвета вокруг изображений. Их автоматически добавляет изображениям браузер Mozilla Firefox, если изображения находятся внутри ссылки. После изучения CSS мы научимся избавляться от этой границы.

Списки

Данный небольшой раздел посвящен парным блочным тегам `ul` и `ol`, позволяющим создавать списки, то есть маркированную или нумерованную последовательность данных.

Теги ul и ol (списки)

Не стоит недооценивать пользу списков. Ведь списки — это не только удобный способ подачи информации. Это еще и любые меню на сайте, так как по правилам семантической верстки любое меню и любой другой организованный список значений должны создаваться с помощью списков. И хотя следование семантической верстке не является пока обязательным правилом для верстальщиков, почему бы не привыкать к правильной верстке с самого начала.

Итак, для создания списков применяются два тега:

- `ul` — маркированный список, в котором одно значение отделяется от другого маркером или выбранным рисунком;
- `ol` — нумерованный список, в котором одно значение отделяется от другого уникальной буквой или числом в одном из алфавитов.

Тег `ul` имеет следующий синтаксис:

```
<ul>
<li>элемент 1</li>
<li>элемент 2</li>
</ul>
```

А тег `ol` следующий:

```
<ol>
<li>элемент 1</li>
<li>элемент 2</li>
</ol>
```

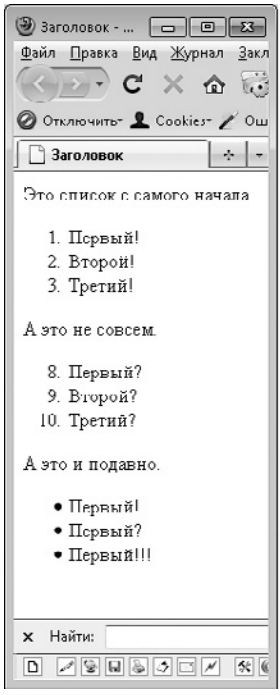
Таким образом, их общий синтаксис ничем не отличается. Но у тега `ol` есть один полезный атрибут, позволяющий указать, с какого номера будет начинаться список. Это атрибут `start` (рис. 1.43).

Тег dl (список определений)

Существует еще один тег для создания списков — `dl`. Как `ol` и `ul`, это парный блочный тег, который используется не сам по себе, а в связке с другими:

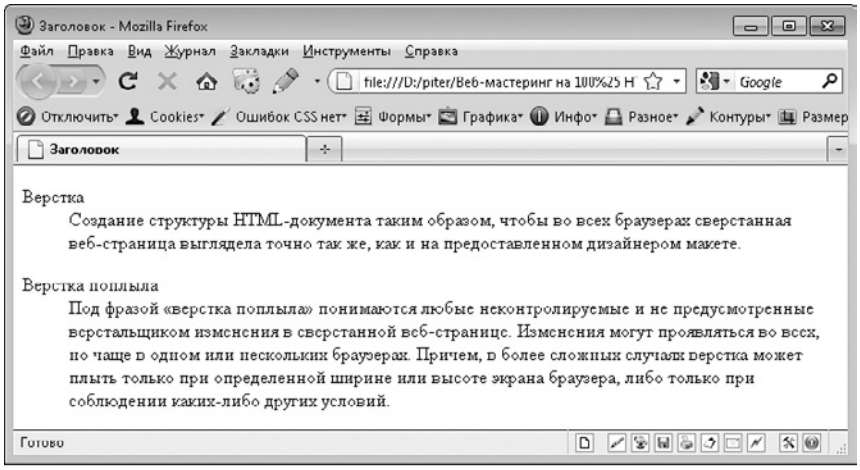
```
<dl>
<dt>Термин</dt>
<dd>Определение термина</dd>
</dl>
```

Как видно из представленного синтаксиса, данный тег можно использовать для создания словарей терминов, разделов FAQ и других текстовых групп, которые состоят из термина и пояснения к нему (рис. 1.44).




```
<div>Это список с самого начала.</div>
<ol>
<li>Первый!</li>
<li>Второй!</li>
<li>Третий!</li>
</ol>
<div>А это не совсем.</div>
<ol start="8">
<li>Первый?</li>
<li>Второй?</li>
<li>Третий?</li>
</ol>
<div>А это и подавно.</div>
<ul>
<li>Первый!</li>
<li>Первый?</li>
<li>Первый!!!</li>
</ul>
```

Рис. 1.43. Примеры списков




```
<dl>
<dt>Верстка</dt>
<dd>Создание структуры HTML-документа таким образом, чтобы во всех браузерах сверстанная
</dl>
<dl>
<dt>Верстка поплыла</dt>
<dd>Под фразой «верстка поплыла» понимаются любые неконтролируемые и не предусмотренные
</dl>
```

Рис. 1.44. Использование тегов dl, dt, dd

Верстаем макет — создаем пункты меню

Вернемся к нашему макету. Теперь мы знаем, как создать пункты меню в нем (листинг 1.7).

Листинг 1.7. Верстаем макет — создаем пункты меню

```
<div>
<div><!-- Шапка начинается -->
    <a href="/"><img alt="" src="img/logo.png" /></a>
    <div>+7 (495) 111-11-11, +7 (903) 111-11-11</div>
    <div>
        <a href="#"><img alt="facebook" src="img/btn_facebook.png"
           width="32px" height="32px" /></a>
        <a href="#"><img alt="google" src="img/btn_google.png" width="32px"
           height="32px" /></a>
        <a href="#"><img alt="livejournal" src="img/btn_livejournal.png"
           width="32px" height="32px" /></a>
        <a href="#"><img alt="mailru" src="img/btn_mailru.png" width="32px"
           height="32px" /></a>
        <a href="#"><img alt="vkontakte" src="img/btn_vkontakte.png"
           width="32px" height="32px" /></a>
        <a href="#"><img alt="twitter" src="img/btn_twitter.png"
           width="32px" height="32px" /></a>
    </div>
    <div>
        <ul>
        <li><a href="#">Новости</a></li>
        <li><a href="#">Статьи</a></li>
        <li><a href="#">Вопросы и ответы</a></li>
        <li><a href="#">Контакты</a></li>
        </ul>
    </div>
    <div>
        <a href="#"><img alt="" src="img/cart.png" width="66px"
           height="59px" /></a>
        <h3>Ваша корзина</h3>
        <div>0 товаров / 0 руб.</div>
    </div>
    <div>
```

```
        <ul>
        <li><a href="#">Регистрация</a></li>
        <li><a href="#">Войти</a></li>
        </ul>
    </div>
    <div>
        <ul>
        <li><a href="#">Главная</a></li>
        <li><a href="#">Скидки и акции</a></li>
        <li><a href="#">Как сделать заказ</a></li>
        <li><a href="#">Доставка и оплата</a></li>
        <li><a href="#">Опт</a></li>
        <li><a href="#">Новые поступления</a></li>
        </ul>
    </div>
</div><!-- Шапка заканчивается -->

<div><!-- Основная часть начинается -->
    <div>
        <ul>
        <li><a href="#">Для женщин</a>
            <ul>
            <li><a href="#">Платья</a></li>
            <li><a href="#">Туники</a></li>
            <li><a href="#">Брюки</a></li>
            <li><a href="#">Юбки</a></li>
            </ul>
        </li>
        <li><a href="#">Для мужчин</a>
            <ul>
            <li><a href="#">Нижнее белье</a></li>
            </ul>
        </li>
        <li><a href="#">Новые поступления</a>
            <ul>
            <li><a href="#">Платья</a></li>
            <li><a href="#">Туники</a></li>
```

```
            <li><a href="#">Брюки</a></li>
            <li><a href="#">Юбки</a></li>
            </ul>
        </li>
        </ul>
    </div>
    <div>
        <a href="#"><img alt="" src="img/banner3.jpg" width="637px"
           height="428px" /></a>
        <a href="#"><img alt="" src="img/banner2.jpg" width="162px"
           height="208px" /></a>
        <a href="#"><img alt="" src="img/banner1.jpg" width="162px"
           height="208px" /></a>
    </div>
    <div>
    <h3>Новые поступления</h3>
    <div>
        <img alt="назад" src="img/rotator_left.png" width="23px"
           height="43px" />
        <a href="#"><img alt="" src="img/product1.jpg" width="173px"
           height="230px" /></a>
        <a href="#"><img alt="" src="img/product2.jpg" width="173px"
           height="230px" /></a>
        <a href="#"><img alt="" src="img/product3.jpg" width="173px"
           height="230px" /></a>
        <a href="#"><img alt="" src="img/product4.jpg" width="173px"
           height="230px" /></a>
        <img alt="вперед" src="img/rotator_right.png" width="23px"
           height="43px" />
    </div>
    </div>
    <div>
    <h3>Wildberries — это большой ассортимент товаров, доступные цены и быстрая
доставка по всей России</h3>
    <p>Интернет-магазин Wildberries предлагает вашему вниманию огромный выбор
стильной одежды, обуви и аксессуаров от самых именитых брендов в мире моды.
Индивидуальный подход к каждому клиенту и высокое качество обслуживания гаранти-
руют, что процесс выбора нужных вам моделей будет простым, быстрым и удобным.</p>
```

```
    <p>Доступные цены, большое разнообразие скидок и акций позволят вам всегда
соответствовать самым актуальным тенденциям моды. Создавая свои уникальные
и неповторимые образы вместе с нами, не забывайте, что все остальные заботы мы
возьмем на себя. Удобство оформления и оперативность службы доставки помогут вам
с легкостью выбрать и получить свой заказ в кратчайшие сроки без лишних затрат.</p>

    <p>Интернет-магазин модной одежды, обуви и аксессуаров Wildberries.ru — это
возможность одеваться стильно за разумные деньги.</p>

    </div>

</div><!-- Основная часть заканчивается -->


<div><!-- Футер начинается -->

    <p>Copyright 2005-2011 © Stracci.ru - модный интернет-магазин одежды, обуви
и аксессуаров. Все права защищены.</p>

    <p>Доставка по Москве бесплатно. Телефоны: +7 (495) 775-55-05 (круглосуточно),
+7 (800) 700-1-500 (бесплатно).</p>

</div><!-- Футер заканчивается -->

</div>
```

Думаю, вы не удивитесь, узнав, что верстка стала еще более непохожей на макет, чем была раньше (рис. 1.45).

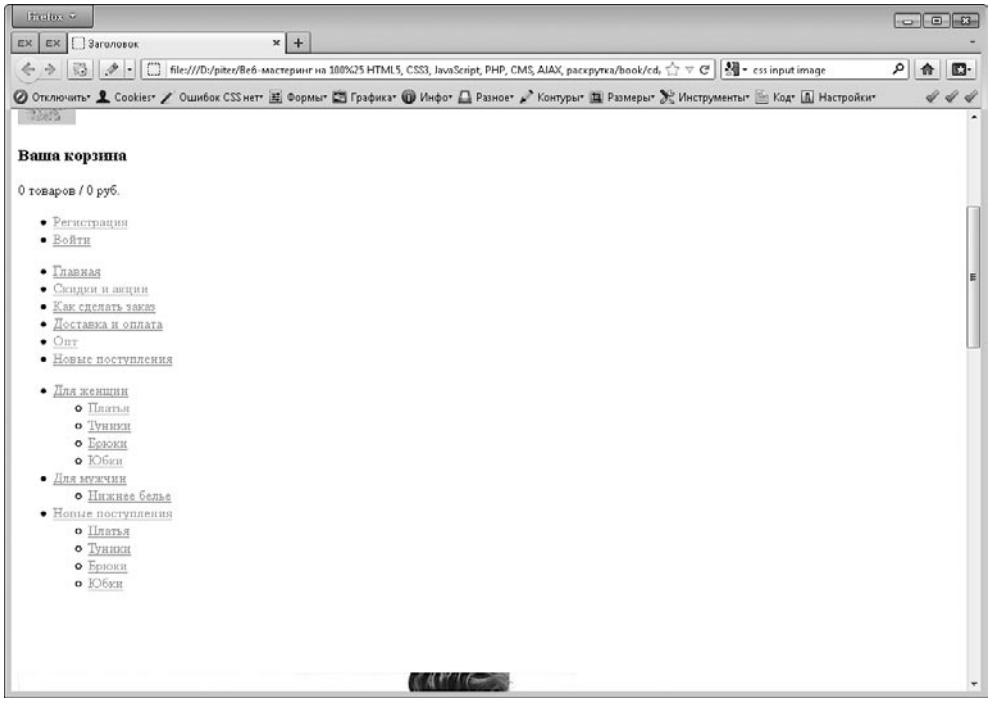


Рис. 1.45. Верстаем макет — добавляем меню

Таблицы

Еще несколько лет назад таблицы использовались повсеместно. И не столько по своему прямому назначению, сколько для верстки макета. Верстальщики использовали преимущественно табличную верстку, то есть располагали элементы на сайте с помощью таблиц. Например, футер и хедер делали в отдельных строках таблицы, а основное содержимое страницы, правую и левую панели — в отдельных столбцах отдельной строки таблицы.

Но все меняется. И теперь на место табличной верстки пришла так называемая дивная верстка. Элементы на сайте позиционируются с помощью тега `div` и свойств CSS. Но таблицы от этого не исчезли. Их по-прежнему нужно использовать, но только по своему прямому назначению.

Итак, для создания таблицы используется парный блочный тег `table`. Он просто сообщает браузеру о том, что его содержимое представляет собой таблицу. А вот как выглядит эта таблица, зависит от тегов, которые содержатся внутри тега `table`. а также количества этих тегов.

Тег tr (строки таблицы)

Как мы знаем, каждая таблица состоит из строк и столбцов. Это же правило действует и в HTML. Чтобы создать строку таблицы, нужно добавить внутрь тега `table` парный блочный тег `tr`. Сколько тегов `tr` вы добавите, столько строк в таблице и будет:

```
<table>

    <tr>…</tr>

    <tr>…</tr>

    …

</table>
```

Тег td (столбцы таблицы)

Каждая строка таблицы состоит из столбцов. Причем количество столбцов должно совпадать для каждой строки отдельной таблицы. Если в первой строке вы указали три столбца, то во второй, третьей и других строках таблицы должно быть именно три столбца. Не больше и не меньше.

Итак, чтобы создать столбец таблицы, нужно внутрь тега `tr` добавить парный блочный тег `td`. Если же вы хотите создать несколько столбцов таблицы, нужно добавить несколько тегов `td`:

```
<table>
   <tr>
      <td>…</td>
      <td>…</td>
      …
   </tr>
   <tr>
      <td>…</td>
      <td>…</td>
      …
   </tr>
   …
</table>
```

Обратите внимание: тег `td` должен находиться только в теге `tr`. Нельзя добавлять столбцы таблицы, не создав перед этим строк.

Общий синтаксис

Общий синтаксис создания таблицы следующий:

```
<table><tr>
   <td>столбец 1 строки 1</td>
   <td>столбец 2 строки 1</td>
</tr><tr>
   <td>столбец 1 строки 2</td>
   <td>столбец 2 строки 2</td>
</tr></table>
```

Это простой синтаксис создания таблицы — без заголовков и подвала для столбцов таблицы. В более сложном варианте все основные строки таблицы находятся внутри тега `tbody`:

```
<table>
<tbody><tr>
   <td>столбец 1 строки 1</td>
   <td>столбец 2 строки 1</td>
</tr><tr>
   <td>столбец 1 строки 2</td>
   <td>столбец 2 строки 2</td>
</tr></tbody>
</table>
```

Данный вариант синтаксиса не отличается от первого варианта. Но теперь вы можете также определить заголовки и/или подвал для столбцов таблицы.

Теги thead и th (заголовки для столбцов таблицы)

Итак, используя второй вариант синтаксиса создания таблицы, можно также добавлять таблице заголовки. При этом все строки заголовка таблицы (как правило, одна строка) должны находиться в теге `thead`. Сама же строка заголовка таблицы создается знакомым вам тегом `tr`. А вот внутри тега `tr` вместо `td` должен находиться тег `th`:

```
<table>
<thead><tr>
    <th>заголовок для столбца 1</th>
    <th>заголовок для столбца 2</th>
</tr></thead>
<tbody><tr>
   <td>столбец 1 строки 1</td>
   <td>столбец 2 строки 1</td>
</tr><tr>
   <td>столбец 1 строки 2</td>
```

```
   <td>столбец 2 строки 2</td>
</tr></tbody>
</table>
```

Вполне естественно, что количество заголовков для столбцов таблицы должно быть точно таким же, как количество столбцов таблицы. То есть количество тегов `th` должно совпадать с количеством тегов `td` в каждой строке таблицы.

Тег tfoot (подвал таблицы)

Помимо заголовка, каждый столбец таблицы может содержать подвал, например, с результатом подсчета всех строк таблицы для данного столбца. Чтобы создать подвал, достаточно воспользоваться парным блочным тегом `tfoot`, содержимое которого аналогично тегу `tbody`. Внутри его тегом `tr` определяются строки подвала таблицы, а внутри строк с помощью тегов `td` задается подвал для каждого столбца таблицы (рис. 1.46):

```
<table>
<thead><tr>
    <th>заголовок для столбца 1</th>
    <th>заголовок для столбца 2</th>
</tr></thead>
<tfoot><tr>
    <th>подвал для столбца 1</th>
    <th>подвал для столбца 2</th>
</tr></tfoot>
<tbody><tr>
   <td>столбец 1 строки 1</td>
   <td>столбец 2 строки 1</td>
</tr><tr>
   <td>столбец 1 строки 2</td>
   <td>столбец 2 строки 2</td>
```

```
</tr></tbody>

</table>
```

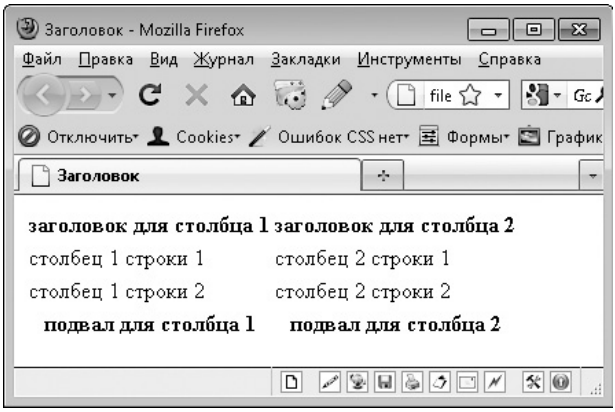


Рис. 1.46. Таблица с заголовками и подвалом для каждого столбца

Обратите внимание, подвал таблицы определяется выше самого содержимого таблицы, то есть сначала идет тег `thead`, затем `tfoot` и только потом `tbody`.

Как можно видеть на рис. 1.46, содержимое заголовков и подвала выводится полужирным шрифтом. Кроме того, хоть это и незаметно на рисунке, содержимое заголовков и подвала по умолчанию центрируется.

Тег caption (заголовок таблицы)

Помимо заголовков для каждого столбца таблицы, вы можете указать заголовок для всей таблицы. Для этого предназначен парный блочный тег `caption`. Если вы решили его задать, то делать это нужно сразу после тега `table`:

```
<table>

<caption>Тестовая таблица</caption>

<thead><tr>

    <th>заголовок для столбца 1</th>

    <th>заголовок для столбца 2</th>

</tr></thead>

<tfoot><tr>

    <th>подвал для столбца 1</th>
```

```
    <th>подвал для столбца 2</th>
</tr></tfoot>
<tbody><tr>
   <td>столбец 1 строки 1</td>
   <td>столбец 2 строки 1</td>
</tr><tr>
   <td>столбец 1 строки 2</td>
   <td>столбец 2 строки 2</td>
</tr></tbody>
</table>
```

Как и заголовки столбцов таблицы, заголовок таблицы выравнивается по центру. Однако выводится он обычным начертанием, а не полужирным.

Объединение строк и столбцов таблицы

До сих пор мы изучали простые таблицы, в которых количество столбцов в каждой строке и строк для каждого столбца совпадает. Однако возможности HTML на простых столбцах не заканчиваются.

Атрибут colspan. Довольно часто возникают ситуации, когда в отдельной строке таблицы необходимо объединить несколько столбцов в один. Сделать это довольно легко — достаточно в теге `td` указать атрибут `colspan`. Его значением является количество столбцов, из которого он состоит.

Например, запись `colspan="3"` говорит о том, что данный столбец замещает собой сразу три столбца таблицы: себя и еще два следующих. Естественно, столбцы, которые были объединены в один, не нужно создавать в данной строке таблицы:

```
<table><tr>
   <td>столбец 1 строки 1</td>
   <td>столбец 2 строки 1</td>
</tr><tr>
   <td colspan="2">столбец 1 и столбец 2 строки 2</td>
</tr></table>
```

Атрибут rowspan. Помимо `colspan`, существует атрибут `rowspan`, позволяющий объединять строки таблицы для данного столбца:

```
<table><tr>
   <td rowspan="2">столбец 1 строки 1 и строки 2</td>
   <td>столбец 2 строки 1</td>
</tr><tr>
   <td>столбец 2 строки 2</td>
</tr></table>
```

На рис. 1.47 показано, как выглядят таблицы, создаваемые приведенными выше записями: слева отображается таблица для заданного атрибута `colspan`, а справа — для атрибута `rowspan`.

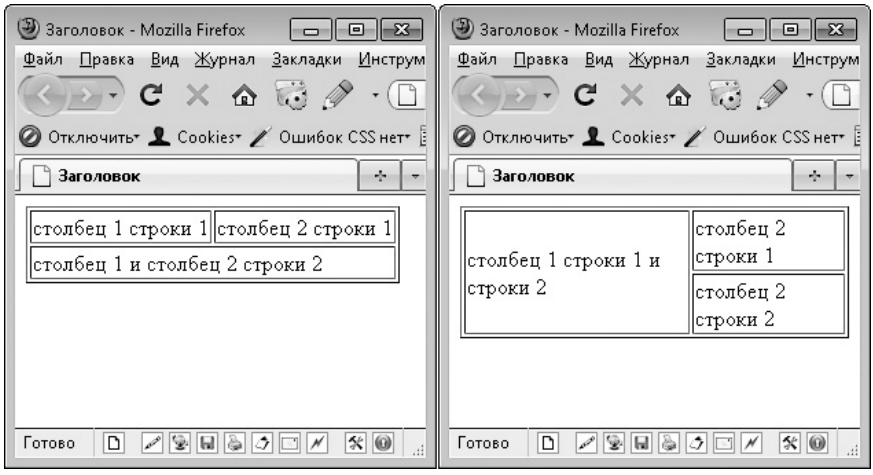


Рис. 1.47. Таблицы с объединенными столбцами и строками

Атрибуты `colspan` и `rowspan` можно указывать не только по отдельности, но и вместе. Но такая необходимость возникает нечасто.

Верстаем макет — табличная верстка

Несмотря на то что табличная верстка не рекомендуется для применения, в качестве примера попробуем сверстать наш макет с ее помощью. Для этого достаточно внести уже сверстанные нами элементы макета в столбцы таблицы (листинг 1.8).

Листинг 1.8. Верстаем макет — табличная верстка

```
<div>
<table><tr><!-- Шапка начинается -->
    <td>
        <a href="/"><img alt="" src="img/logo.png" /></a>
    </td><td>
        <div>+7 (495) 111-11-11, +7 (903) 111-11-11</div>
        <div>
            <a href="#"><img alt="facebook" src="img/btn_facebook.png"
               width="32px" height="32px" /></a>
            <a href="#"><img alt="google" src="img/btn_google.png"
               width="32px" height="32px" /></a>
            <a href="#"><img alt="livejournal" src="img/btn_livejournal.png"
               width="32px" height="32px" /></a>
            <a href="#"><img alt="mailru" src="img/btn_mailru.png"
               width="32px" height="32px" /></a>
            <a href="#"><img alt="vkontakte" src="img/btn_vkontakte.png"
               width="32px" height="32px" /></a>
            <a href="#"><img alt="twitter" src="img/btn_twitter.png"
               width="32px" height="32px" /></a>
        </div>
        <div>
            <ul>
            <li><a href="#">Новости</a></li>
            <li><a href="#">Статьи</a></li>
            <li><a href="#">Вопросы и ответы</a></li>
            <li><a href="#">Контакты</a></li>
            </ul>
        </div>
    </td><td>
        <div>
            <a href="#"><img alt="" src="img/cart.png" width="66px"
               height="59px" /></a>
            <h3>Ваша корзина</h3>
            <div>0 товаров / 0 руб.</div>
        </div>
        <div>
```

```
            <ul>
            <li><a href="#">Регистрация</a></li>
            <li><a href="#">Войти</a></li>
            </ul>
        </div>
    </td>
</tr><tr>
    <td colspan="3">
        <ul>
        <li><a href="#">Главная</a></li>
        <li><a href="#">Скидки и акции</a></li>
        <li><a href="#">Как сделать заказ</a></li>
        <li><a href="#">Доставка и оплата</a></li>
        <li><a href="#">Опт</a></li>
        <li><a href="#">Новые поступления</a></li>
        </ul>
    </td>
</tr></table><!-- Шапка заканчивается -->
<table><tr><!-- Основная часть начинается -->
    <td colspan="3">Место под баннер с брендами</td>
</tr><tr>
    <td rowspan="2">
        <div>
            <ul>
            <li><a href="#">Для женщин</a>
                <ul>
                <li><a href="#">Платья</a></li>
                <li><a href="#">Туники</a></li>
                <li><a href="#">Брюки</a></li>
                <li><a href="#">Юбки</a></li>
                </ul>
            </li>
            <li><a href="#">Для мужчин</a>
                <ul>
                <li><a href="#">Нижнее белье</a></li>
                </ul>
            </li>
            <li><a href="#">Новые поступления</a>
```

```
                <ul>
                <li><a href="#">Платья</a></li>
                <li><a href="#">Туники</a></li>
                <li><a href="#">Брюки</a></li>
                <li><a href="#">Юбки</a></li>
                </ul>
            </li>
            </ul>
        </div>
    </td><td>
        <a href="#"><img alt="" src="img/banner3.jpg" width="637px"
            height="428px" /></a>
    </td><td>
        <a href="#"><img alt="" src="img/banner2.jpg" width="162px"
            height="208px" /></a>
        <a href="#"><img alt="" src="img/banner1.jpg" width="162px"
            height="208px" /></a>
    </td>
</tr><tr>
    <td colspan="2">
        <div>
        <h3>Новые поступления</h3>
        <div>
            <img alt="назад" src="img/rotator_left.png" width="23px"
               height="43px" />
            <a href="#"><img alt="" src="img/product1.jpg" width="173px"
               height="230px" /></a>
            <a href="#"><img alt="" src="img/product2.jpg" width="173px"
               height="230px" /></a>
            <a href="#"><img alt="" src="img/product3.jpg" width="173px"
               height="230px" /></a>
            <a href="#"><img alt="" src="img/product4.jpg" width="173px"
               height="230px" /></a>
            <img alt="вперед" src="img/rotator_right.png" width="23px"
               height="43px" />
        </div>
        </div>
        <div>
```

```
        <h3>Wildberries — это большой ассортимент товаров, доступные цены и быстрая
доставка по всей России</h3>

        <p>Интернет-магазин Wildberries предлагает вашему вниманию огромный выбор
стильной одежды, обуви и аксессуаров от самых именитых брендов в мире моды.
Индивидуальный подход к каждому клиенту и высокое качество обслуживания гаранти-
руют, что процесс выбора нужных вам моделей будет простым, быстрым и удобным.</p>

        <p>Доступные цены, большое разнообразие скидок и акций позволят вам всегда
соответствовать самым актуальным тенденциям моды. Создавая свои уникальные
и неповторимые образы вместе с нами, не забывайте, что все остальные заботы мы
возьмем на себя. Удобство оформления и оперативность службы доставки помогут вам
с легкостью выбрать и получить свой заказ в кратчайшие сроки без лишних затрат.</p>

        <p>Интернет-магазин модной одежды, обуви и аксессуаров Wildberries.ru — это
возможность одеваться стильно за разумные деньги.</p>

        </div>

    </td>

</tr></table><!-- Основная часть заканчивается -->


<div><!-- Футер начинается -->

    <p>Copyright 2005-2011 © Stracci.ru — модный интернет-магазин одежды, обуви
и аксессуаров. Все права защищены.</p>

    <p>Доставка по Москве бесплатно. Телефоны: +7 (495) 775-55-05 (круглосуточно),
+7 (800) 700-1-500 (бесплатно).</p>

</div><!-- Футер заканчивается -->

</div>
```

И наконец свершилось чудо — наша веб-страница стала чуть более похожа на макет (рис. 1.48).

К сожалению, без подключения CSS большего сходства мы добиться не сможем. Так что на этом временно отложим наш макет. Но отложим не данную версию веб-страницы, а предыдущую. Ведь табличная верстка не для нас. Не сомневайтесь, что мы все-таки сможем сверстать данный макет с помощью дивной верстки.

Форма и ее элементы

Вот мы и подошли к самому сложному в HTML — к формам и элементам форм. Фактически это царство языков JavaScript и PHP, так как без использования данных языков формы являются лишь украшением и никакой функциональности не несут. Именно поэтому в этой главе мы коснемся форм лишь поверхностно, после чего продолжим их изучение в главах, посвященных JavaScript и PHP.

Что же такое форма? Это средство, которое позволяет посетителю взаимодействовать с вашим сайтом: отправлять сайту данные, определенные вами в форме.

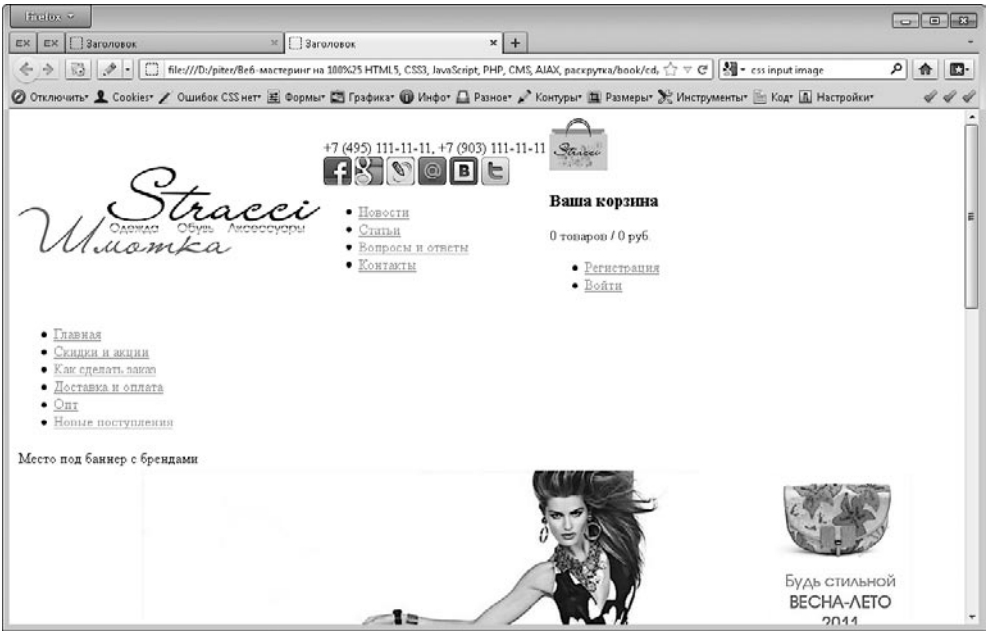


Рис. 1.48. Верстаем макет с помощью табличной верстки

Типичными примерами форм являются формы входа на сайт, поиска по сайту, отправки сообщения администратору сайта, регистрации на сайте, опросы. Все перечисленное и многое другое делается с помощью форм HTML.

Формы в HTML могут состоять из текстовых полей и текстовых областей, флажков и переключателей, а также раскрывающихся списков. Все это — элементы формы. Каждый элемент служит для того, чтобы передать какое-либо значение сайту.

Тег form (создаем форму)

Чтобы создать форму, следует воспользоваться парным блочным тегом `form`.

Все, что заключено в этом теге, будет считаться формой. Находиться в нем могут не только флажки, текстовые области и прочие элементы формы, но и любые другие теги языка HTML.

Конечно, значения всех тегов, не являющихся элементами формы, не будут передаваться сайту при отправке формы. В первую очередь потому, что тегам, не являющимся элементами формы, нельзя присваивать значения. Эти теги используются внутри формы лишь для того, чтобы придать форме нужный дизайн:

```
<form method="get" action="">

    <h3>Пример формы</h3>

    <div>Это уже наша форма</div>

</form>
```

Атрибут action. Как вы уже знаете, каждая форма служит для того, чтобы передавать значения на сайт. Но куда именно?

По умолчанию значения передаются на ту же страницу, где находится форма, то есть после отправки формы в браузере открывается та же страница.

Но с помощью атрибута `action` можно изменить поведение по умолчанию. Если вы укажете в значении данного атрибута абсолютный или относительный адрес, то после отправки формы будет открыта указанная в значении веб-страница.

Желательно всегда задавать значение атрибута `action`. Даже если форма должна передать значения на ту же страницу, где она расположена. В этом случае в качестве значения атрибута `action` укажите пустую строку: `action=""`.

Атрибут method. В теге `form` также часто задается атрибут `method`. С его помощью можно определить способ, которым будут переданы на сайт значения, выбранные пользователем в форме.

Наиболее популярными являются два способа передачи значений сайту:

- `get` — передача значений в адресе веб-страницы;
- `post` — в заголовке запроса к веб-странице.

Подробнее об этом мы поговорим в главе 4, посвященной языку PHP.

Элементы формы

Практически все элементы формы создаются с помощью тега `input`. И только от атрибута `type` этого одинарного строчного тега зависит, какой именно элемент формы будет создан:

```
<input type="значение" />
```

Помимо атрибута `type`, у тега `input` есть еще несколько обязательных атрибутов.

Тегу `input` следует добавить атрибут `name`. Значение этого атрибута определяет название переменной, в которой при отправке формы на страницу будет передано выбранное в данном элементе формы значение.

Тегу `input` следует также добавить атрибут `value`. Значение данного атрибута определяет само значение, которое при отправке формы будет передано в переменной, указанной атрибутом `name`.

При необходимости тегу `input` можно присвоить атрибут `readonly` со значением `readonly`. При наличии данного атрибута пользователю будет запрещено изменять значение по умолчанию, указанное вами в теге `input`. В этом случае значение элемента формы можно будет изменить только с помощью JavaScript.

Вместо атрибута `readonly` можно использовать атрибут `disabled` со значением `disabled`. В этом случае элемент формы не только становится недоступным для изменения, но и окрашивается в серый цвет. Кроме того, значение, указанное в этом элементе формы, не будет передано на сервер при отправке формы.

Наличие других атрибутов зависит от типа создаваемого элемента формы.

Кнопка отправки формы (submit)

Ранее мы уже несколько раз упоминали о возможности отправки формы. Но возникает вопрос, как это сделать. Ведь сама по себе форма не содержит никаких элементов. В том числе и кнопки, позволяющей отправить форму. Вот с создания этой кнопки мы и начнем.

Чтобы создать кнопку отправки формы, нужно присвоить атрибуту `type` значение `submit`:

```
<input type="submit" value="Отправить" name="form_submit" />
```

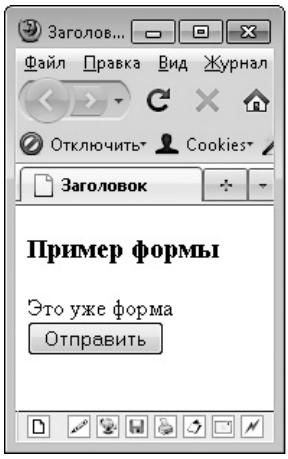
В данном случае значение атрибута `value` используется для задания текста, который будет отображаться на кнопке отправки формы (рис. 1.49). Конечно, значение этого атрибута будет также передано в форму, но, как правило, это значение никогда не используется.

Если в дальнейшем вы не собираетесь использовать значение, указанное в атрибуте `value` кнопки отправки формы, то атрибут `name` можно опустить.

Текстовое поле (text)

Как показывает практика, чаще всего применяется такой элемент формы, как текстовое поле. С его помощью посетитель может передать на сайт любую строку текста. Текстовое поле создается с помощью значения `text` атрибута `type`:

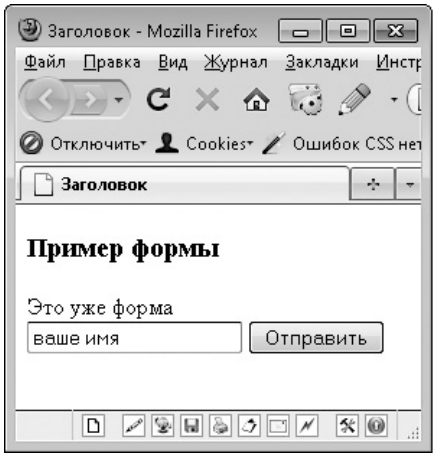
```
<input type="text" name="form_name" value="ваше имя" />
```




```
<form method="get" action="">
    <h3>Пример формы</h3>
    <div>Это уже форма</div>

    <input type="submit" value="Отправить" />
</form>
```

Рис. 1.49. Кнопка отправки формы

Атрибут `value` текстового поля предназначен для того, чтобы хранить значение, введенное пользователем в это поле. Но пока пользователь ничего не ввел, атрибут `value` может содержать в себе значение по умолчанию, то есть то значение, которое вы указали в атрибуте `value` при верстке. При этом значение по умолчанию будет отображаться в текстовом поле (рис. 1.50).




```
<form method="get" action="">
    <h3>Пример формы</h3>
    <div>Это уже форма</div>
    <input type="text" name="form_name"
 value="ваше имя" />
    <input type="submit" value=
"Отправить" />
</form>
```

Рис. 1.50. Создание текстового поля

При необходимости можно ограничить количество символов, которые посетитель может ввести в текстовом поле. Для этого предназначен атрибут `maxlength`, значение которого и является максимально разрешенным количеством символов.

Поле ввода пароля (password)

Разновидностью текстового поля является поле для ввода пароля. Оно поддерживает те же атрибуты, что и текстовое поле. Но весь текст, который посетитель введет в данном поле, будет отображаться в виде звездочек (рис. 1.51).

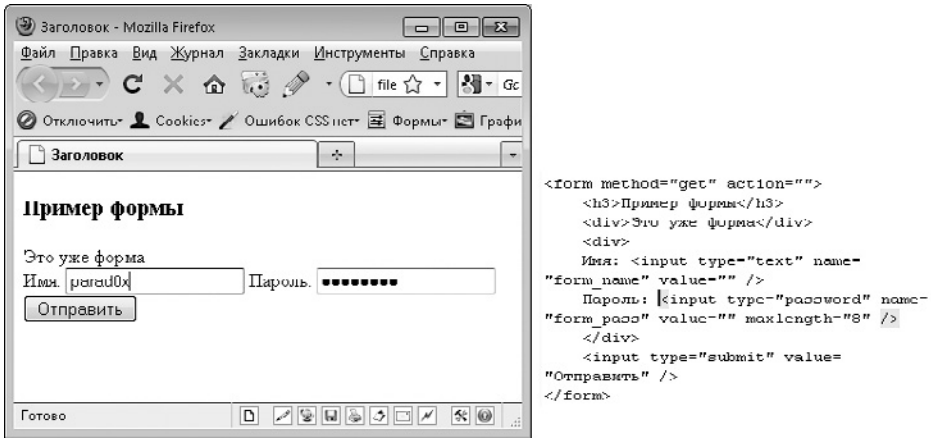


Рис. 1.51. Поля для ввода пароля

Итак, для создания поля ввода пароля предназначено значение `password` атрибута `type`:

```
<input type="password" name="form_pass" value=""
maxlength="8" />
```

Тег textarea (текстовая область)

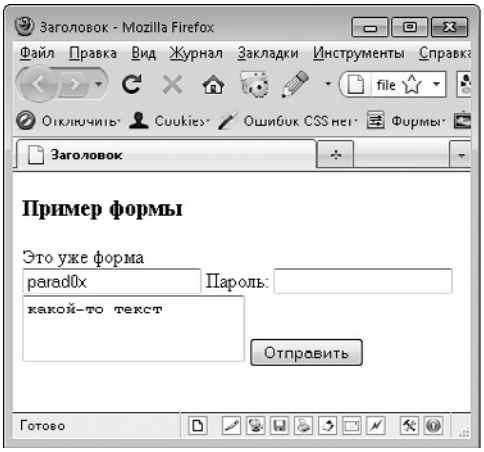
Раз уж речь зашла о тексте, рассмотрим один из элементов формы, который создается не с помощью тега `input`. Я имею в виду текстовую область, для создания которой следует использовать парный блочный тег `textarea`:

```
<textarea name="form_bigtext">какой-то текст</textarea>
```

Как видно из рис. 1.52, текстовая область представляет собой текстовое поле, состоящее из нескольких строк. Соответственно, в него можно вводить полноценный текст, включающий в себя переносы строк, тогда как в обычное текстовое поле можно ввести только одну строку.

Количество строк, из которых состоит текстовое поле, определяется значением атрибута `rows`. То есть значение данного атрибута определяет количество строк, которые текстовая область будет занимать на экране. Ничто не помешает вам ввести

в текстовую область больше строк текста. В этом случае в текстовой области появится вертикальная прокрутка.




```
<form method="get" action="">
    <h3>Пример формы</h3>
    <div>Это уже форма</div>
    <div>
    <input type="text" name="form_name" value
="Имя" />
    Пароль: <input type="password" name=
"form_pass" value="" maxlength="8" />
    </div>
    <textarea name="form_bigtext">какой-то
 текст</textarea>
    <input type="submit" value="Отправить" />
</form>
```

Рис. 1.52. Текстовая область

Как и тегу `input`, тегу `textarea` можно присваивать атрибуты `readonly` и `disabled`. Их действие аналогично действию для тега `input`.

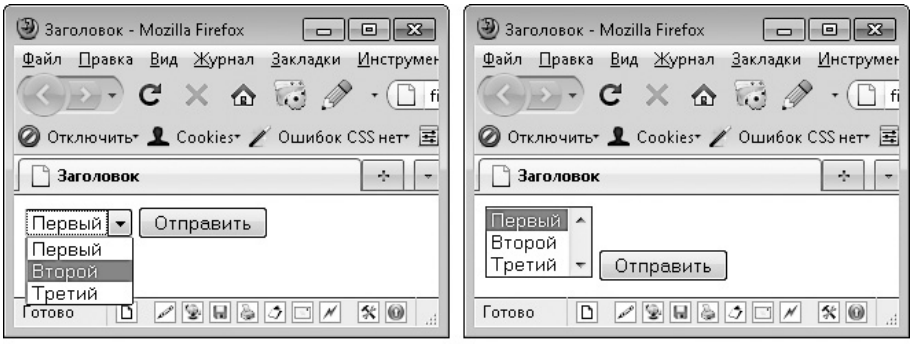
Тег select (раскрывающийся список)

Помимо тегов `input` и `textarea`, для создания элементов формы служит еще один тег — строчный парный тег `select`. Он позволяет создавать списки с одинарным или множественным выбором (рис. 1.53).

Основной синтаксис создания списка следующий:

```
<select name="form_select">

<option value="1">Первый</option>

<option value="2">Второй</option>

<option value="3">Третий</option>

</select>
```

Атрибут `name`, определяющий имя переменной, в которой значение элемента формы будет передано при отправке формы, указывается в теге `select`. А атрибут `value`, определяющий передаваемое значение, расположен в каждом из вложенных тегов `option`.




```
<form method="get" action="">
<select name="form_select">
<option value="1">Первый</option>
<option value="2">Второй</option>
<option value="3">Третий</option>
</select>
    <input type="submit" value=
"Отправить" />
</form>
```

```
<form method="get" action="">
<select size="3" name="form_select">
<option value="1">Первый</option>
<option value="2">Второй</option>
<option value="3">Третий</option>
</select>
    <input type="submit" value=
"Отправить" />
</form>
```

Рис. 1.53. Варианты списков, создаваемых тегом select

Парный тег `option` описывает один вариант, который посетитель может выбрать из данного списка.

При этом содержимое тега `option` используется в качестве названия данного пункта в списке (любые теги, размещенные внутри тега `option`, будут проигнорированы). А значение атрибута `value` выбранного посетителем тега `option` будет передано при отправке формы.

Атрибут size. По умолчанию с помощью тега `select` создается раскрывающийся список, то есть список, в котором изначально видно только одно значение. Остальные значения можно увидеть, щелкнув на стрелке вниз (см. рис. 1.53, *слева*).

Однако с помощью значения атрибута `size` тега `select` можно указать, сколько пунктов списка должно быть видно изначально (см. рис. 1.53, *справа*). Если в списке больше значений, чем количество, указанное атрибутом `size`, то другие значения можно увидеть, щелкая на стрелочках вверх и вниз.

Атрибут multiple. Атрибут `size` позволяет отобразить сразу несколько пунктов списка. Но это не означает, что посетитель может выбрать из списка несколько элементов. По-прежнему из списка можно выбрать только один пункт.

Чтобы посетитель мог выбрать из списка любое количество пунктов, нужно использовать атрибут `multiple` с одноименным значением: `multiple="multiple"`.

Атрибут selected. Помимо атрибута `value`, тегу `option` можно присвоить атрибут `selected` с одноименным значением. В этом случае данный пункт будет считаться выбранным по умолчанию. Если в пределах тега `select` нет ни одного тега `option` с атрибутом `selected`, то по умолчанию в списке будет отображаться первый пункт.

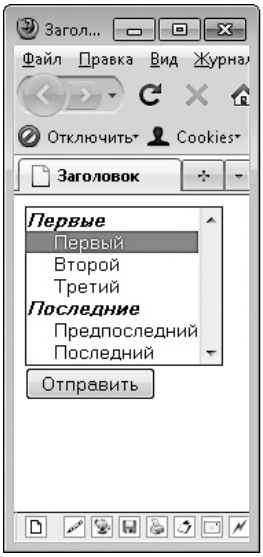
Естественно, в пределах тега `select` может быть только один тег `option` с атрибутом `selected`.

Атрибут disabled. Уже известный вам атрибут `disabled` со значением `disabled` можно устанавливать не только для тега `select`, но и для `option`. В первом случае становится отключенным весь список. А во втором — только отдельный пункт списка.

Группировка с помощью тега optgroup. Если в создаваемом списке находится большое количество пунктов, поиск нужного среди них становится довольно сложным занятием. В этом случае крайне полезной может оказаться возможность объединения отдельных пунктов списка в группы. Делается это с помощью парного тега `optgroup`, добавляемого внутрь тега `select`. В результате получается следующий синтаксис (рис. 1.54):

```
<select size="7" name="form_select">

<optgroup label="Первые">

<option value="1">Первый</option>

<option value="2">Второй</option>

<option value="3">Третий</option>

</optgroup>

<optgroup label="Последние">

<option value="777">Предпоследний</option>

<option value="778">Последний</option>

</optgroup>

</select>
```

Теги `option` помещаются внутрь тега `optgroup`, а метка группы указывается в атрибуте `label` тега `optgroup`.




```
<form method="get" action="">
<select size="7" name="form_select">
<optgroup label="Первые">
<option value="1">Первый</option>
<option value="2">Второй</option>
<option value="3">Третий</option>
</optgroup>
<optgroup label="Последние">
<option value="777">Предпоследний<
/option>
<option value="778">Последний</option>
</optgroup>
</select>
    <input type="submit" value=
"Отправить" />
</form>
```

Рис. 1.54. Группировка пунктов в списке

Флажки (checkbox)

Вернемся к уже знакомому вам тегу `input`. Значение `checkbox` атрибута `type` данного тега поможет нам создать флажок:

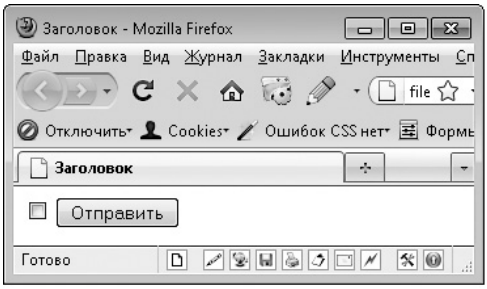
```
<input type="checkbox" name="form_checkbox" value="1" />
```

На рис. 1.55, *слева* мы создали флажок без всяких подписей и условных обозначений. Обычно же флажки создаются с подписью. Для этого можно как просто написать что-то рядом, так и воспользоваться специальным тегом `label` (рис. 1.55, *справа*). В последнем случае получится следующий синтаксис:

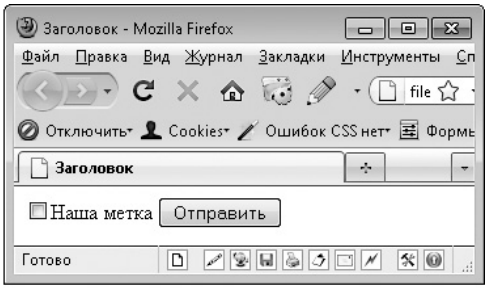
```
<input type="checkbox" name="form_checkbox" value="1" id="id_
checkbox" /><label for="id_checkbox">Наша метка</label>
```

Таким образом, при использовании тега `label` тегу `input` также нужно присвоить атрибут `id` со значением, уникальным для текущей страницы, после чего это же значение указать в качестве значения атрибута `for` тега `label`.

Использование тега `label` для создания меток предпочтительнее обычного текста. В первую очередь из-за удобства для конечного пользователя, ведь при применении тега `label` установить/снять флажок можно не только щелчком на поле флажка, но и на его метке.




```
<form method="get" action="">
<input type="checkbox" name="form_checkbox
" value="1" />
    <input type="submit" value="Отправить"
 />
</form>
```



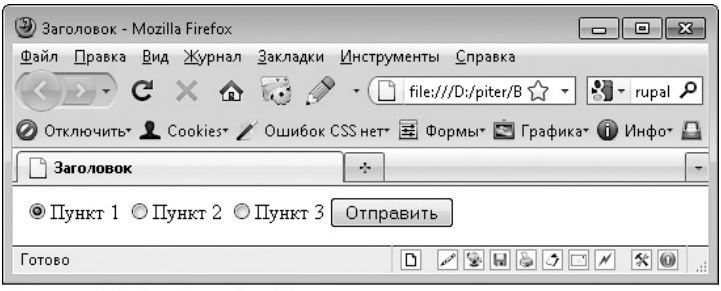

```
<form method="get" action="">
<input type="checkbox" name="form_checkbox
" value="1" id="id_checkbox" /><label for=
"id_checkbox">Наша метка</label>
    <input type="submit" value="Отправить"
 />
</form>
```

Рис. 1.55. Создаем флажки

Переключатели (radio)

Переключатели имеют некоторое сходство с флажками. Но если флажки работают по принципу логического «И» (позволяют посетителю выбрать И это значение, И это, И это), то переключатели используют принцип логического «ИЛИ», позволяя выбрать только одно из нескольких значений.

По этой причине переключатель никогда не создается как одиночный элемент — из чего выбирать, если переключатель имеет всего одно положение? Чаще всего создается группа переключателей, имеющих одно имя, но разные значения (рис. 1.56).




```
<form method="get" action="">
<input type="radio" name="form_radio" id="form_radio1" value="1" />
<label for="form_radio1">Пункт 1</label>
<input type="radio" name="form_radio" id="form_radio2" value="2" />
<label for="form_radio2">Пункт 2</label>
<input type="radio" name="form_radio" id="form_radio3" value="3" />
<label for="form_radio3">Пункт 3</label>
<input type="submit" value="Отправить" />
</form>
```

Рис. 1.56. Использование переключателей

Для создания переключателя служит значение `radio` атрибута `type` тега `input`:

```
<input type="radio" name="form_radio" value="1" />
```

Так мы создали только одно положение переключателя. Чтобы создать несколько положений переключателя, достаточно использовать тот же атрибут с различными значениями:

```
<input type="radio" name="form_radio" value="1" />

<input type="radio" name="form_radio" value="2" />

<input type="radio" name="form_radio" value="3" />
```

Как и флажки, переключатели представляют собой просто элемент формы, без всяких подписей и пояснений. Что довольно критично для переключателей, так как становится совершенно неясно, чем одно положение переключателя отличается от другого. Чтобы как-то озаглавить переключатель, нужно воспользоваться знакомым нам тегом `label`:

```
<input type="radio" name="form_radio" id="form_radio1" value="1"
/><label for="form_radio1">Пункт 1</label>

<input type="radio" name="form_radio" id="form_radio2" value="2"
/><label for="form_radio2">Пункт 2</label>

<input type="radio" name="form_radio" id="form_radio3" value="3"
/><label for="form_radio3">Пункт 3</label>
```

Скрытый элемент (hidden)

Иногда при отправке формы на сервер нужно передать некое значение, но видеть это значение посетителю не обязательно.

Например, так могут передаваться значения, которые посетитель указал на предыдущих шагах мастера. Или какой-то секретный ключ, который применяется формой для проверки легитимности полученных данных.

Для этих целей в HTML используется значение `hidden` атрибута `type` тега `input`:

```
<input type="hidden" value="1234567" name="userid" />
```

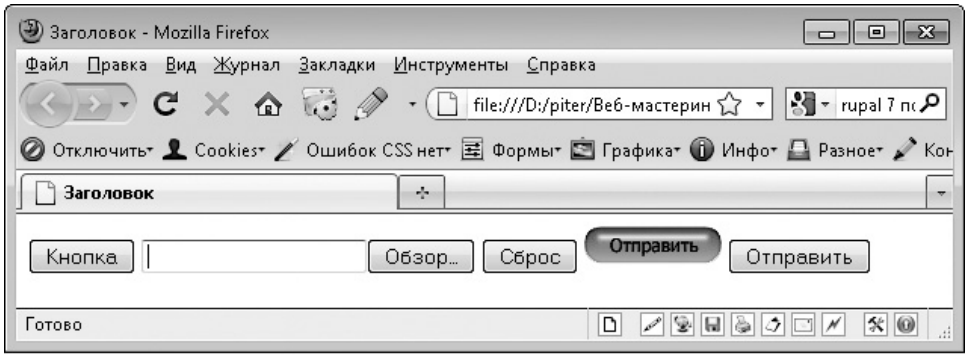
Данный пример позволяет при отправке формы передать на сервер переменную `userid` со значением `1234567`. Но при этом на веб-странице не будет видно никаких элементов формы, которые указывали бы на передачу этого значения.

Другие варианты тега input

Мы рассмотрели наиболее популярные варианты тега `input`. Но помимо них у атрибута `type` тега `input` существуют и другие значения:

- `button` — позволяет создать кнопку;
- `file` — используется для создания кнопки выбора файла на локальном компьютере, который будет загружен на сервер;
- `image` — аналог значения `submit` (то есть создает кнопку отправки формы), но также позволяет указать картинку, которая будет отображаться вместо стандартной кнопки;
- `reset` — создает кнопку для сброса всех значений, которые пользователь внес в форму, к значениям по умолчанию.

Возможно, в своих будущих проектах вы никогда не столкнетесь с данными вариантами тега `input`. А может, и будете их использовать, это дело привычки. На рис. 1.57 показано, как выглядит каждый перечисленный элемент формы.




```
<form method="get" action="">
<input name="form_button" value="Кнопка" type="button" />
<input name="form_file" type="file" />
<input name="form_reset" type="reset" />
<input type="image" src="imgbutton.gif" />
<input type="submit" value="Отправить" />
</form>
```

Рис. 1.57. Другие варианты тега input

Используем Flash

Думаю, ни для кого не секрет, что в HTML-документ можно встраивать картинки, баннеры, flash.

Как работать с картинками, вы уже знаете.

Как работать с баннерами, вы тоже уже в курсе. Ведь баннер — это обычная картинка в формате GIF. Точнее, не совсем обычная, а анимированная. Но от этого алгоритм работы с ней не меняется — баннеры также вставляются с помощью тега `img`.

А вот как вставить на страницу SWF-файл, то есть флеш-контент?

Код для размещения SWF-файла довольно объемен, поэтому запоминать его нет смысла. Проще всего создать шаблон, после чего вставлять его при необходимости в верстку. Итак, в листинге 1.9 представлен код размещения SWF-файла, а на рис. 1.58 показан результат публикации SWF-файла.

Листинг 1.9. Код для размещения Flash

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="460"
height="379" id="myFlashContent">
   <param name="movie" value="flash/pl.swf" />
   <param name="wmode" value="transparent" />
   <!--[if !IE]>-->
   <object type="application/x-shockwave-flash" data="flash/pl.swf"
    width="460" height="379">
      <param name="wmode" value="transparent" />
   <!--<![endif]-->
   <a href="http://www.adobe.com/go/getflashplayer">
   <img src="http://www.adobe.com/images/shared/download_buttons/get_flash_player.
gif" alt="Скачать Flash-плеер" />
   </a>
   <!--[if !IE]>-->
   </object>
   <!--<![endif]-->
</object>
```

Как можно заметить из кода, SWF-файл вставляется два раза. Дело в том, что алгоритм размещения Flash на сайте различается для разных браузеров. По этой причине применяется два разных способа подключения Flash — чтобы полученный код работал во всех браузерах.

При использовании представленного выше кода, если у посетителя имеется установленный Flash-плеер, у него будет воспроизведен SWF-файл (см. рис. 1.58, *слева*). Если же в браузере посетителя не установлен плагин Flash-плеера, то отобразится стандартная ссылка на скачивание Flash-плеера (см. рис. 1.58, *справа*).

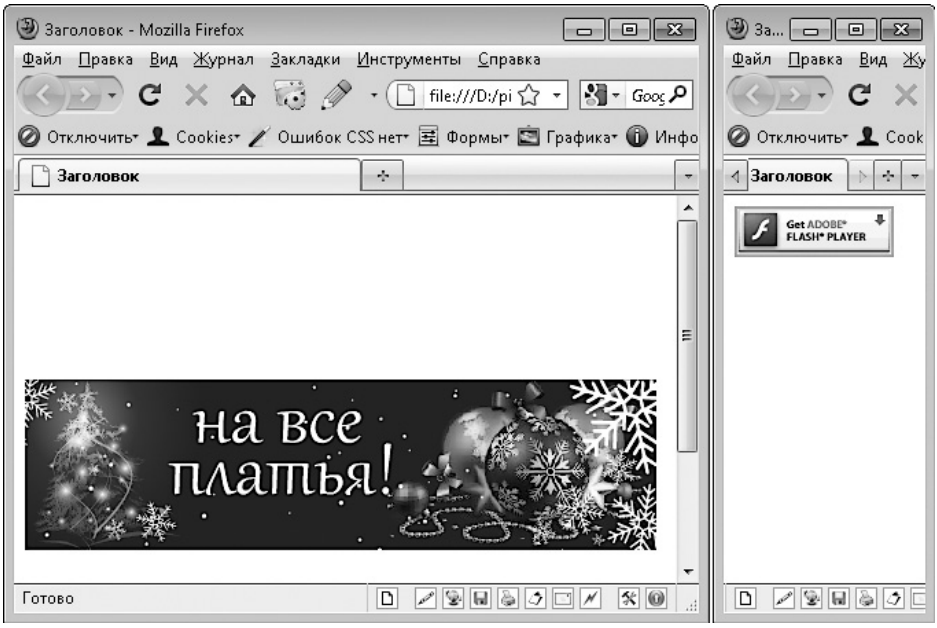


Рис. 1.58. Отображение Flash на сайте *(слева)* и ссылки на скачивание Flash-плеера при его отсутствии *(справа)*

Нововведения HTML5

Вот мы и подошли к завершающему разделу, посвященному HTML. В предыдущих разделах мы изучили функционал языка HTML, который чаще всего используется верстальщиками. Но мы намеренно обходили все нововведения версии HTML5.

Не переживайте по этому поводу — вы уже сейчас можете верстать веб-страницы как на языке XHTML и более ранних версиях HTML, так и на версии HTML5. Другое дело, что вы пока не сможете использовать новые возможности, которые принесла в мир версия HTML5. Не сможете до тех пор, пока не дочитаете эту главу до конца. Но и тогда не забывайте, что на данный момент подавляющее большинство браузеров у пользователей по всему миру не поддерживает новые возможности языка HTML5. По этой причине применять их, по крайней мере на момент написания книги, бессмысленно.

Кроме того, следует помнить, что пока спецификация HTML5 не утверждена окончательно. Поэтому в любой момент могут появиться новые теги либо быть запрещены уже существующие.

Итак, попробуем перейти на HTML5. Сделать это несложно. Просто смените в верстке `DOCTYPE` на `<!DOCTYPE html>`. Вот и все.

Что? У вас уже стоял такой DOCTYPE? Тогда в чем проблема, значит, мы давно уже используем HTML5.

Контейнеры в HTML5

Видимо, из-за пристрастия разработчиков к семантической верстке в HTML5 у тега div появилось множество братьев-близнецов, то есть точно таких же по функциональности тегов, имеющих разное название и предназначенных для немножко разных целей:

- header — все, что находится в данном теге, будет считаться шапкой сайта;
- footer — все, что располагается в этом теге, будет считаться футером всего сайта либо отдельного раздела сайта (например, для размещения имени автора статьи и даты создания статьи);
- article — разработчики языка HTML5 надеются, что в этот тег вы будете помещать отдельные статьи, новости, записи блога или другие целостные блоки текста;
- nav — контейнер для размещения ссылок главного меню сайта;
- section — контейнер для разбиения сайта на отдельные секции;
- aside — контейнер для создания левой или правой боковой панели на сайте.

Все перечисленные теги игнорируются браузером Internet Explorer версий 6, 7, 8. И только 9-я версия данного браузера стала распознавать перечисленные теги. Что касается браузера Mozilla Firefox, в нем поддержка данных тегов появилась только в 4-й версии.

В качестве примера попробуем переписать первую версию нашего макета с использованием всех возможностей HTML5 (листинг 1.10).

Листинг 1.10. Использование тегов-контейнеров HTML5

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
</head>
<body>
<section>
   <header>шапка</header>
   <section>основная часть</section>
```

```
   <footer>футер</footer>
</section>
</body>
</html>
```

Невооруженным глазом можно заметить, что данная версия верстки гораздо нагляднее, чем созданная нами ранее, в которой применялись только теги `div`.

На рис. 1.59, *слева* показана полученная веб-страница в браузере Internet Explorer 9. А на рис. 1.59, *справа* веб-страница открыта в браузере Mozilla Firefox 3.0, игнорирующем перечисленные теги.

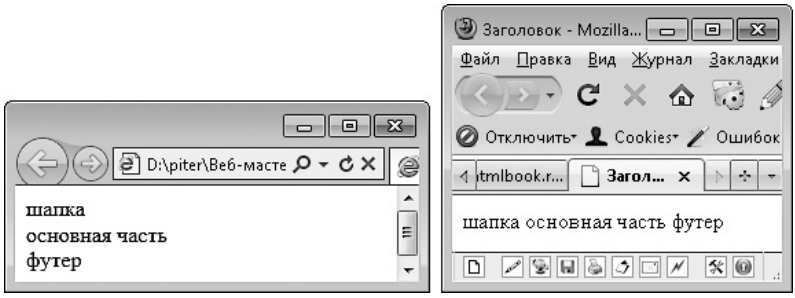


Рис. 1.59. Браузер, поддерживающий HTML5 *(слева)* и не поддерживающий его *(справа)*

Работа с текстом в HTML5

Склонность разработчиков HTML5 к семантической верстке выразилась также в добавлении следующих тегов.

Dialog. Данный парный блочный тег используется для создания диалога между собеседниками. Он выделяет сам диалог, а отдельные реплики создаются с помощью парных тегов `dt` и `dd`, расположенных внутри данного тега. Тег `dt` содержит имя собеседника, а тег `dd` — его реплику:

```
<dialog>
   <dt>Собеседник 1</dt>
   <dd>Доброго дня.</dd>
   <dt>Собеседник 2</dt>
   <dd>И вам доброго дня.</dd>
</dialog>
```

Данный тег был включен в первоначальную версию HTML5. Но, скорее всего, он будет удален в окончательной версии HTML5.

Mark. Парный строчный тег `mark` предназначен для выделения важной части текста: фразы, идеи или мысли. В браузерах Google Chrome и Mozilla Firefox текст, расположенный внутри данного тега, отображается на желтом фоне. В остальных браузерах текст не выделяется.

Time. Этот парный строчный тег используется в тексте для выделения даты и/или времени. Он может иметь несколько атрибутов:

- `datetime` — содержит дату и/или время для выделенного текста;
- `pubdate` —включает в себя дату публикации документа.

Meter. Данный парный строчный тег позволяет выделять в тексте числа вместе с единицами измерения. При этом атрибуты тега `meter` позволяют описать данное число:

- `value` — содержит само число;
- `min` — определяет возможный минимум для данного числа, если число может находиться только в определенном диапазоне;
- `low` — предельное значение, при достижении которого число считается минимальным;
- `hight` — предельное значение, при достижении которого число считается высоким;
- `max` — определяет максимум для данного показателя;
- `optium` — задает оптимальное значение для данного показателя.

Тег `meter` не поддерживается даже 9-й версией браузера Internet Explorer.

Progress. Данный парный строчный тег показывает прогресс выполнения какой-либо задачи, то есть отображает число или фразу, описывающую, на каком этапе находится выполнение той или иной задачи.

В дальнейшем прогресс можно изменять с помощью JavaScript.

Атрибуты данного тега позволяют лучше понять, насколько близко к поставленной цели вы подошли:

- `value` — число, определяющее текущий этап выполнения задачи;
- `max` — число, при достижении которого задача будет выполнена.

Тег `progress` не поддерживается даже 9-й версией браузера Internet Explorer.

Hgroup. Данный парный тег позволяет группировать теги заголовков (от `h1` до `h6`) веб-страницы или раздела.

Wbr. Этот одинарный строчный тег позволяет указать место, где разрешено делать перенос строки в тексте.

Изображения и ссылки в HTML5

Пятая версия языка HTML не обошла своим вниманием и изображения. Для работы с ними поддерживаются следующие дополнительные теги.

Figure. Данный парный блочный тег используется для создания изображения с подписью. Само изображение определяется с помощью тега `img`, расположенного внутри данного тега. А подпись задается посредством тега `legend`:

```
<figure>
   <legend>Изображение, которое изменит вашу жизнь</legend>
   <img alt="" src="/img/pict.png" />
</figure>
```

Этот тег может также использоваться для группировки любых данных, а не только изображения и подписи.

Figcaption. Данный парный тег позволяет задать описание для группы, созданной тегом `figure`. Соответственно, он должен находиться внутри тега `figure`. Причем он должен быть либо первым, либо последним тегом внутри тега `figure`:

```
<figure>
   <figcaption>Описание</figcaption>
   <legend>Изображение, которое изменит вашу жизнь</legend>
   <img alt="" src="/img/pict.png" />
</figure>
```

Возможности мультимедиа

Раньше для того, чтобы встроить в HTML-страницу проигрыватель аудио или видео, приходилось либо подключать Flash-проигрыватель, либо использовать какие-нибудь другие плагины, установленные в браузере пользователя. Но с приходом HTML5 все намного упростилось. Теперь для воспроизведения файлов мультимедиа предназначены специальные теги.

Работаем с аудио. Для воспроизведения аудиофайлов служит парный блочный тег `audio`:

```
<audio src="daleka_doroga.mp3" autoplay="autoplay" controls=
"controls" loop="3">Информация об аудиофайле</audio>
```

Внутри тега `audio` содержится описывающий аудиофайл текст. Этот текст будет выводиться в тех браузерах, которые не умеют работать с тегом `audio`.

Сам же аудиофайл и настройки его воспроизведения могут находиться в атрибутах тега `audio`:

- `src` — указывает путь к аудиофайлу, размещенному в Интернете;
- `autoplay` — наличие данного атрибута говорит о том, что аудиофайл будет воспроизведен автоматически после загрузки веб-страницы;
- `controls` — при наличии данного атрибута будет отображена панель управления воспроизведением файла (рис. 1.60);

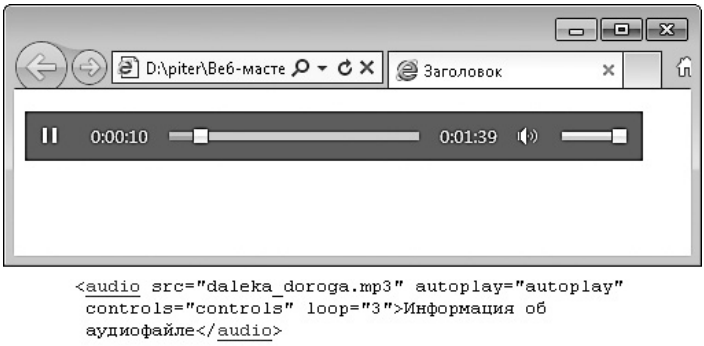


Рис. 1.60. Панель управления воспроизведением аудиофайла

- `loop` — содержит число, определяющее, сколько раз нужно повторить воспроизведение данного аудиофайла.

Настройки аудиофайла могут также определяться в атрибутах `src`, `type` и `codecs` одинарного тега `source`, который можно разместить внутри тега `audio`.

Внутри тега `audio` может находиться любое количество тегов `source`. Данная особенность часто используется, чтобы указать несколько вариантов аудиофайла, созданных с помощью различных кодеков.

Тег `audio`, как и тег `source`, может работать с любыми аудиофайлами, независимо от того, каким кодеком они созданы. Однако этого же нельзя сказать о браузерах. Они поддерживают ограниченный набор кодеков. Причем, по крайней мере на данный момент, список поддерживаемых кодеков отличается для каждого браузера:

- Internet Explorer 9 поддерживает кодеки MP3, AAC;
- Google Chrome 6 — ogg/vorbis, MP3, AAC;
- Opera 10.60 — ogg/vorbis, WAV;
- Safari 5 — MP3, AAC;
- Mozilla Firefox 3.6 — ogg/vorbis, WAV.

По этой причине, чтобы аудиофайл воспроизводился во всех последних версиях браузеров, приходится публиковать несколько вариантов данного аудиофайла, созданных с помощью разных кодеков.

Работаем с видео. Парный блочный тег `video` позволяет разместить на веб-странице видеофайл. При этом внутри тега `video` хранится описание видеофайла, которое будет отображено на странице, если браузер не поддерживает тег `video`.

Путь к видеофайлу, который нужно воспроизвести, может располагаться в атрибуте `src` данного тега:

```
<video src="/myvideo/video.avi">Описание видеофайла</video>
```

Путь к видеофайлу может также находиться в атрибуте `src` вложенного одинарного тега `source`.

Как и для тега `audio`, каждый браузер поддерживает свой набор видеокодеков, которыми может быть создан видеофайл:

- Internet Explorer 9 поддерживает кодеки H.264;
- Google Chrome 6 — ogg/theora, H.264, WebM;
- Opera 10.60 — ogg/theora, WebM;
- Safari 5 — H.264;
- Mozilla Firefox 3.6 — ogg/theora, WebM.

Интерактивные возможности

С появлением языка JavaScript и объектной модели DOM сайты стали динамичными (интерактивными). Появилась возможность изменять содержимое веб-страницы без ее перезагрузки. Но только с выходом HTML5 интерактивность стала частью языка HTML. В него были добавлены отдельные теги для создания интерактивных элементов.

Menu. Парный блочный тег `menu` предназначен для создания меню, пункты которого позволяют выполнять какой-либо код на языке JavaScript.

Тег menu лишь определяет само меню. А вот пункты этого меню создаются с помощью одинарного тега command:

```
<menu>
<command onclick="alert('команда 1')" label="Пункт первый"/>
<command onclick="alert('команда 2')" label="Пункт второй"/>
</menu>
```

Имя пункта меню содержится в атрибуте label тега command. Тогда как сама команда, которая будет выполнена при щелчке на данном пункте меню, находится в атрибуте onclick тега command.

Вместо тега command можно использовать теги li или button.

Datagrid. Парный блочный тег datagrid позволяет создавать интерактивные таблицы и списки. Такие таблицы смогут редактировать сами посетители сайта: удалять или создавать новые столбцы и строки таблицы, править содержимое ячеек, сортировать и сворачивать ячейки.

Собственно, тег datagrid лишь указывает, что таблица или список, расположенный внутри его, является интерактивным. Соответственно, внутри тега datagrid можно размещать теги:

- table, tr, td, th, caption, thead, tfoot, tbody — теги для создания таблиц;
- select, option, optgroup — теги для создания раскрывающихся списков.

Данный тег был включен в первоначальную версию HTML5. Но, возможно, он будет удален в окончательной версии HTML5.

Details. Данный парный блочный тег предназначен для создания дополнительной информации, которая будет отображаться только в том случае, если посетитель этого захочет.

По умолчанию дополнительная информация скрыта и будет отображена при щелчке на добавленной браузером сноске. Но вы можете присвоить тегу details атрибут open с одноименным значением, чтобы дополнительная информация отображалась сразу.

Данный тег не поддерживается даже 9-й версией браузера Internet Explorer.

Canvas. С помощью данного парного блочного тега можно средствами JavaScript создавать рисунки, анимацию, игры. Он позволяет рисовать различные изображения

или же выводить уже созданные изображения, при этом трансформируя их или меняя их свойства.

Атрибуты `width` и `height` данного тега дают возможность задать высоту и ширину создаваемого холста.

Но наиболее важен для тега `canvas` атрибут `id`. Он позволяет задать идентификатор, который в дальнейшем будет использоваться в JavaScript для работы с данным тегом.

Вот и подошла к концу глава про HTML. В процессе обучения вы узнали о многих тегах и атрибутах HTML5 и прежних версий. Но не обо всех. Если у вас по этому поводу возникло чувство сожаления, предлагаю продолжить изучение языка HTML самостоятельно. Для этого нет необходимости приобретать дополнительные книги. В Интернете можно найти множество электронных справочников, например:

- http://www.w3.org/TR/html4/index/elements.html — официальный английский справочник по всем тегам языка HTML;
- http://htmlbook.ru — отличный русскоязычный ресурс с огромным количеством информации о HTML.

Глава 2

CSS

Базовые понятия
Программное обеспечение
CSS-хаки
Изучаем свойства: отступ
Изучаем свойства: тип тега
Изучаем свойства: границы
Изучаем свойства: размер
Изучаем свойства: позиционирование
Верстаем макет страницы
Изучаем свойства: списки
Верстаем горизонтальное меню
Изучаем свойства: фон
Изучаем свойства: текст
Верстаем макет: меню
Изучаем свойства: таблицы
Изучаем свойства: печать
Изучаем свойства: другое
Нововведения CSS3

Язык HTML, который мы изучили в прошлой главе, предназначен для структурирования информации. Он позволяет расположить блоки информации в нужной последовательности.

За визуализацию информации отвечает другой язык — CSS. С его помощью можно:

- указать размер, начертание, цвет и другие параметры шрифта, который будет использоваться для вывода текста;
- расположить блок информации в нужном месте веб-страницы;
- оформить блок информации фоновой картинкой, границей и другими визуальными элементами.

Таким образом, с его помощью можно сделать все то, чего нам не хватало в прошлой главе, чтобы привести наш HTML-документ к виду, идентичному предоставленному дизайнером PSD-макету.

Базовые понятия

Язык CSS представляет собой набор правил, которые применяются к определенному объекту веб-страницы, чтобы задать особенности его вывода на экран. Типичный пример использования CSS:

```
body {
  font-family: Arial, Verdana,  sans-serif;
  font-size: 10pt;
  background-color: #f0f0f0;
  color: #ff0000;
}
h1 {
  color: #a52a2a;
  font-size: 24pt;
  font-family: Georgia, Times, serif;
  font-weight: bold;
}
```

В этом примере создано два набора правил: один используется для тега `body`, а другой указывает, как будет выводиться содержимое тега `h1`.

Набор правил имеет следующий формат:

```
селектор {
  правило CSS
  правило CSS
  ...
}
```

Каждый набор правил состоит из свойств CSS и их значений, то есть отдельных правил CSS, имеющих следующий формат записи: `свойство: значение;`. Обратите внимание, что значения в CSS не берутся в кавычки, как это было для значений атрибутов языка HTML. Точнее, в кавычки берутся только строчные значения, которые не являются ключевыми словами. А такие значения в CSS встречаются редко.

Итак, у нас получился следующий формат записи CSS-правила:

```
селектор {
   свойство: значение;
}
```

Возможен также такой вариант:

```
селектор { свойство: значение; }
```

Переносы строк в CSS не играют большой роли. Их добавляют для удобства верстальщика.

Внутри селектора может находиться любое количество свойств CSS (с их значениями), перечисленных через точку с запятой. Но за пределами селектора указывать свойства CSS нельзя. Каждое свойство CSS должно задаваться в каком-либо селекторе. Ведь именно селектор определяет, к какому элементу веб-страницы необходимо применить данное свойство CSS.

Селекторы

В качестве селектора могут выступать следующие данные:

- название тега — в самом простом случае в качестве селектора используется название тега: `body`, `p`, `input` и т. д.;

- присвоенный тегу класс (то есть значение присвоенного тегу атрибута `class`) — в данном случае перед селектором указывается символ «`.`»: `.myclass`, `.form_submit` и т. д.;
- идентификатор тега (то есть значение присвоенного тегу атрибута `id`) — в данном случае перед селектором указывается символ `#`: `#myid`, `#form_submit` и т. д;
- `*` — специальный селектор `*` позволяет присвоить свойства сразу всем элементам веб-страницы.

Атрибуты class и id

В HTML любому тегу можно присвоить атрибуты `class` и/или `id`. Оба этих атрибута служат схожей цели — идентификации какого-либо тега либо группы тегов.

Идентификация тегов позволяет обращаться к конкретному тегу через CSS или JavaScript. Именно поэтому мы не рассматривали данные атрибуты ранее. Для чистого HTML они просто не нужны.

Атрибут `class` предназначен для идентификации группы тегов. По этой причине значение данного атрибута может быть одинаковым для нескольких тегов.

Атрибут `id` предназначен для идентификации какого-либо конкретного тега. В пределах всей HTML-страницы должен существовать только один тег с определенным значением тега `id`. Наличие нескольких тегов с одинаковым значением тега `id` не допускается и может привести к ошибкам.

В качестве значения атрибутов `class` и `id` может выступать любая строка, состоящая из букв латинского алфавита, цифр, знака подчеркивания и дефиса. Причем начинаться значения данных атрибутов должны именно с буквы. Кроме того, значения атрибутов `class` и `id` чувствительны к регистру!

Обратите также внимание, что любому тегу может быть присвоено сразу несколько классов. В этом случае они указываются через пробел в значении атрибута `class`: `<div class="class1 class2 class3"></div>`.

Вес селекторов

Весь CSS-код выполняется браузером последовательно, в том порядке, в котором он указан. Сначала CSS-свойства присваиваются селекторам, объявленным первыми. Потом тем, которые идут за ними, и так до конца CSS.

Но такое поведение бывает в самом простом случае, когда селекторы не конфликтуют между собой. Например, рассмотрим такой HTML-код:

```
<div class="hello_class" id="hello_id">Привет, мир</div>
```

Для данного HTML-кода будет CSS-код:

```
div {
   font-size: 11pt;
   font-weight: bold;
}
#hello_id {
   font-size: 27pt;
   text-decoration: underline;
}
.hello_class {
   font-size: 14pt;
   font-style: italic;
}
```

В приведенном CSS-коде одному и тому же тегу свойства CSS присваиваются в трех разных селекторах. Причем в каждом селекторе указывается разное значение для свойства `font-size`. Какое же значение свойства `font-size` будет применяться браузером?

В этом случае ответ «последнее» является неправильным. Все дело в весе разных типов селекторов.

В CSS каждый тип селектора имеет свой вес:

- название тега имеет вес 1 балл;
- присвоенный тегу класс — 10 баллов;
- идентификатор тега — 100 баллов.

Если два селектора, идентифицирующих один и тот же тег, имеют одинаковый вес, для CSS-свойства будет взято значение из того селектора, который стоит ниже в коде.

Однако если вес одного из селекторов больше, чем вес других, то значение для CSS-свойства будет взято именно из него, независимо от того, в каком месте CSS-файла селектор объявлен.

В нашем примере самый большой вес имеет селектор на основе идентификатора. А значит, будет применяться именно он, независимо от расположения данного селектора в CSS-коде (то есть расположен ли он выше или ниже других селекторов, идентифицирующих тот же тег).

Другие CSS-свойства в нашем примере не конфликтуют между собой: `font-size` указан только для тега `div`; `text-decoration` — только для идентификатора `#hello_id`; `font-style` — только для класса `.hello_class`. Какие же из этих свойств будут применены к нашему тегу? Правильный ответ: все перечисленные. Ведь CSS-свойства применяются сразу из всех селекторов, а не только из того, который имеет наибольший вес.

Вес CSS-свойства

Чтобы еще больше вас запутать, разработчики CSS придумали не имеющее значения свойство `!important`, с помощью которого вы можете повысить вес конкретного CSS-свойства, чтобы независимо от веса селекторов применялось именно оно.

Данное свойство используется следующим образом:

```
свойство: значение !important;
```

Например:

```
div {
   font-size: 11pt !important;
   font-weight: bold;
}
#hello_id {
   font-size: 27pt;
   text-decoration: underline;
}
```

В представленном примере свойству `font-size` будет присвоено значение `11pt`, а не `27pt`, указанное в идентификаторе тега.

Каскадирование стилей

Если посмотреть отображение одной и той же HTML-страницы, созданной без использования CSS, в разных браузерах, то можно заметить различия в цвете ссы-

лок, разные отступы от заголовков и другие различия в выводе HTML-страницы. Эти вопросы непосредственно связаны с CSS. А точнее, со стилями, которые браузеры по умолчанию присваивают отдельным тегам HTML-документа.

Каждый браузер имеет свой набор правил CSS, которые по умолчанию присваиваются различным тегам. Совокупность этих правил называется стилем браузера.

Помимо стиля браузера, существует стиль пользователя. Практически каждый браузер позволяет пользователю указать файл с правилами CSS, которые должны присваиваться тегам любого открываемого HTML-документа.

И наконец, еще один стиль называется стилем автора. Стиль автора задаете вы как верстальщик документа.

Если какое-либо CSS-свойство не было указано для данного тега в стилях автора, его значение берется из стилей пользователя. Если стиль пользователя также не содержит значения для данного свойства, браузер попытается найти значение для него в стилях браузера. А если и там нет нужного значения, то будет применено значение по умолчанию.

А что будет в случае, если значение для одного и того же CSS-свойства указано в разных стилях? В данном случае в силу опять вступает система весов, но на этот раз весов для стилей, а не для селекторов, что можно описать следующим списком.

1. Стиль браузера.
2. Стиль пользователя.
3. Стиль автора.
4. Стиль автора с добавлением `!important`.
5. Стиль пользователя с добавлением `!important`.

В данном списке стили отсортированы в порядке возрастания весов. В самом начале указан стиль с наименьшим весом. А внизу — с наибольшим.

Составные селекторы

В качестве селектора может выступать не только отдельный тег, но и тег с определенным классом или идентификатором: `тег.класс` или `тег#идентификатор`.

В этом случае CSS-свойства применяются только к тегу с данным классом либо с данным идентификатором. При этом вес селектора определяется суммой весов его составляющих:

- `p` — вес 1 балл;
- `.intro` — 10 баллов;
- `p.intro` — 11 баллов;
- `#glava1` — 100 баллов;
- `p#glava1` — 101 балл.

Тегу может быть присвоен не только один класс, но и несколько: `p.intro.glava1`. В представленном примере вес селектора равен 21 баллу. А CSS-свойства, размещенные в данном селекторе, будут применяться только к тегу `p`, которому одновременно присвоено сразу два перечисленных класса (`<p class="intro glava1">test</p>`).

Однако обратите внимание: браузер Internet Explorer 6 не умеет работать с подобными селекторами, поэтому использовать в селекторе сразу два и более класса не рекомендуется.

Контекстные селекторы

Контекстными называются селекторы, в которых через пробел указано сразу несколько тегов, классов и/или идентификаторов.

- `div p` — CSS-свойства, указанные в селекторе, применяются только к тегам `p`, которые находятся внутри тега `div`:

```
<body>
   <div>
      <p>Селектор применяется</p>
   </div>
   <p>Селектор НЕ применяется</p>
</body>
```

- `.intro .outro` — CSS-свойства, указанные в селекторе, применяются только к тем тегам с классом `outro`, которые находятся внутри тегов с классом `intro`:

```
<body>
   <div class="intro">
      <p class="outro">Селектор применяется</p>
   </div>
   <p class="outro">Селектор НЕ применяется</p>
</body>
```

- `div.intro p#outro` — CSS-свойства, указанные в селекторе, применяются только к тому тегу `p` с идентификатором `outro`, который находится внутри тега `div` с классом `intro`:

```
<body>
   <div class="intro">
      <p id="outro">Селектор применяется</p>
   </div>
   <p id="outro">Селектор НЕ применяется. Данная запись во-
обще ошибочная — в пределах HTML-документа может быть только
один тег с идентификатором outro</p>
</body>
```

- `#footer div p.tel` — CSS-свойства, указанные в селекторе, применяются только к тем тегам `p` с классом `tel`, которые находятся внутри тегов `div`, располагающихся, в свою очередь, в теге с идентификатором `footer`:

```
<body>
   <div id="footer">
      <div>
         <p class="tel">Селектор применяется</p>
         <p class="tel">Селектор применяется</p>
      </div>
      <p class="tel">Селектор НЕ применяется</p>
   </div>
   <p class="tel">Селектор НЕ применяется</p>
</body>
```

Как вы уже поняли из примеров, контекстные селекторы позволяют точнее указать теги, которым будут присвоены перечисленные в селекторе CSS-свойства. Они дают возможность задавать цепочки из родительских тегов, в которых должен находиться дочерний тег.

Селекторы на соседние элементы

Под соседними элементами на веб-странице понимаются теги, которые следуют друг за другом. Причем ни один из этих тегов не является дочерним по отношению к другому. Рассмотрим несколько примеров:

- `<p>пример <strong>очень</strong> простого текста</p>` — теги `p` и `strong` НЕ являются соседними, так как тег `strong` расположен внутри тега `p` и является дочерним по отношению к нему;
- `<p>пример <strong>очень</strong> <em>простого</em> текста</p>` — теги `strong` и `em` являются соседними, так как имеют одинаковый уровень вложенности (ни один из них не является дочерним по отношению к другому) и между ними нет других тегов этого же уровня вложенности;
- `<p>пример <strong>очень</strong> простого <em>текста</em></p>` — теги `strong` и `em` являются соседними, так как они имеют одинаковый уровень вложенности (ни один из них не является дочерним по отношению к другому) и между ними нет других тегов этого же уровня вложенности.

Чтобы применить CSS-свойства к соседнему тегу, следует в селекторе между этими тегами поставить знак +: `strong + em { CSS-свойства }`. При этом CSS-свойства будут применяться к тегу `em`, который является соседним к тегу `strong`.

Использовать обращение к соседним тегам следует только в крайнем случае, так как браузер Internet Explorer 6 не умеет работать с ними.

Селекторы на дочерние элементы

Селекторы на дочерние элементы по своему назначению в чем-то похожи на контекстные селекторы (селекторы вида `тег тег {}`).

Как вы уже знаете, контекстный селектор позволяет указать тег, который находится внутри другого тега. Например, контекстный селектор `div strong { CSS-свойства }` применит указанные CSS-свойства к следующим тегам `strong`:

```
<body>
   <div>
      <strong>Селектор применяется</strong>
      <p>
         <strong>Селектор применяется</strong>
         <em><strong>Селектор применяется</strong></em>
      </p>
      <strong>Селектор применяется</strong>
```

```
    </div>

    <strong>Селектор НЕ применяется</strong>

</body>
```

Таким образом, контекстному селектору все равно, находится ли тег `strong` непосредственно в теге `div`, либо же он располагается в каком-либо другом теге или даже тегах, которые, в свою очередь, находятся в теге `div`. Главное, чтобы одним из родителей тега `strong` был тег `div`.

Селектор на дочерний элемент позволяет установить более строгое правило. При его использовании дочерний тег должен размещаться именно в родительском теге. Многоуровневые вложения тегов не допускаются.

Итак, чтобы воспользоваться селектором на дочерний элемент, достаточно поставить между двумя тегами селектора знак >.

Например, перепишем приведенный выше селектор так, чтобы он указывал на дочерний элемент: `div > strong { CSS-свойства }`. Теперь он будет применять указанные CSS-свойства к следующим тегам `strong`:

```
<body>

    <div>

        <strong>Селектор применяется</strong>

        <p>

            <strong>Селектор НЕ применяется</strong>

            <em><strong>Селектор НЕ применяется</strong></em>

        </p>

        <strong>Селектор применяется</strong>

    </div>

    <strong>Селектор НЕ применяется</strong>

</body>
```

Использовать обращение к дочерним элементам следует только в крайнем случае, поскольку браузер Internet Explorer 6 не умеет работать с ними.

Селекторы атрибутов

Существует возможность указывать в селекторе не только теги, классы и идентификаторы, но и атрибуты и их значения. В этом случае CSS-свойства будут применяться только к тем тегам, для которых указан атрибут:

- `тег[атрибут]` — с любым значением;
- `тег[атрибут="значение"]` — со значением `значение`;
- `тег[атрибут~="значение"]` — со значением `значение` либо если среди значений данного атрибута (перечисленных через пробел, как это возможно в атрибуте `class`) есть значение `значение`;
- `тег[атрибут^="значение"]` — со значением, которое начинается фразой `значение`;
- `тег[атрибут$="значение"]` — со значением, которое оканчивается фразой `значение`;
- `тег[атрибут*="значение"]` — со значением, которое содержит в себе фразу `значение` (в начале, в конце или в середине значения).

Сам тег из селектора можно исключить. В этом случае CSS-свойства будут применяться ко всем тегам, для которых указан данный атрибут.

Например:

- `img[alt]` — применить свойства ко всем картинкам, для которых указан атрибут `alt`;
- `a.link[target="_blank"]` — ко всем ссылкам с классом `link`, которые открываются в новом окне;
- `a[href^="http://"]` — ко всем ссылкам, указывающим на абсолютные адреса;
- `a[href$=".com"]` — ко всем ссылкам на домены в зоне `.com`.

Все разновидности селекторов атрибутов не поддерживаются браузером Internet Explorer 6, поэтому использовать их следует с осторожностью.

Псевдоклассы

В составе селектора могут также указываться псевдоклассы, позволяющие ограничить область применения CSS-свойств только теми тегами, которые в данный момент имеют определенное состояние.

Псевдокласс указывается в конце селектора после двоеточия:

- `:псевдокласс`;
- `тег:псевдокласс`;

- `.класс:псевдокласс`;
- `#идентификатор:псевдокласс`;
- `тег тег.класс:псевдокласс` и т. д.

Существуют следующие псевдоклассы.

- `:active` — указывает на ссылку, которая в данный момент активна (то есть на которую только что навели указатель и щелкнули кнопкой мыши).

  Преимущественно применяется только к тегу `a`.

- `:link` — указывает на ссылки (за исключением якорей), которые пользователь еще не посещал (которых нет в истории браузера). По большему счету, использование псевдокласса `.link` равнозначно применению CSS-свойств к тегу `a`. Единственное исключение: тег `a` действует как на ссылки, так и на якоря.

  Применяется только к тегу `a`.

- `:focus` — указывает на элемент страницы, для которого в данный момент установлен фокус.

  Применяется преимущественно к элементам формы.

- `:hover` — указывает на элемент страницы, над которым в данный момент находится указатель мыши.

  Обратите внимание: если в CSS для ссылок указан и селектор с псевдоклассом `:visited`, и селектор с псевдоклассом `:hover` и в этих селекторах есть одинаковые свойства CSS, для которых установлены разные значения, то селектор с псевдоклассом `:hover` должен указываться после селектора с псевдоклассом `:visited`. Иначе для посещенных ссылок при наведении на них указателя мыши значения конфликтующих свойств будут браться из селектора с псевдоклассом `:visited`, а не из селектора с псевдоклассом `:hover`.

  В браузере Internet Explorer 6 может применяться только к тегу `a`. В более современных браузерах данный псевдокласс разрешено применять ко всем элементам: `div`, `li`, `tr`, `td` и т. д.

- `:visited` — указывает на ссылки (за исключением якорей), которые пользователь уже посещал (которые есть в истории браузера).

- `:first-child` — указывает только на первый по счету элемент (среди дочерних элементов). Например, селектор `div strong:first-child` применит CSS-свойства только к следующим тегам:

```
<body>
   <div>
      <strong>Селектор применяется</strong>
      <strong>Селектор НЕ применяется</strong>
```

```
        <em>Здесь тем более НЕ применяется</em>
        <strong>Селектор НЕ применяется</strong>
    </div>
    <strong>Селектор НЕ применяется</strong>
</body>
```

Браузер Internet Explorer 6 не поддерживает данный псевдокласс.

- `:lang(язык)` — указывает только на элементы, для которых установлен выбранный язык (присутствует атрибут `lang` со значением, указывающим на тот же язык).

  В качестве языка могут выступать следующие значения:

  - `ru` — русский;
  - `en` — английский;
  - `de` — немецкий;
  - `fr` — французский;
  - `it` — итальянский.

Псевдоэлементы

Еще одним содержимым селектора могут быть псевдоэлементы. Они указываются точно так же, как и псевдоклассы, то есть после двоеточия.

Существуют следующие псевдоэлементы.

- `:after` — позволяет добавить определенный текст (в том числе и символы Unicode) в конец элемента веб-страницы. Это делается следующим образом (рис. 2.1):

  ```
  тег:after{
      content: "текст";
  }
  ```

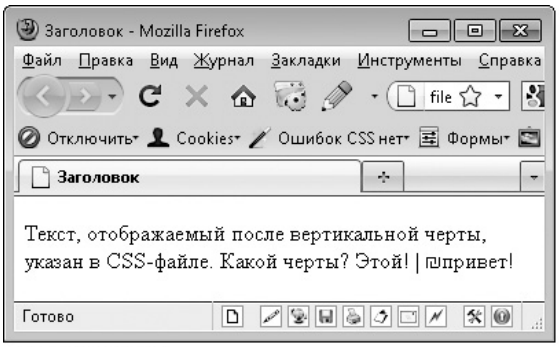
  Браузер Internet Explorer 6 не поддерживает данный псевдоэлемент.

- `:before` — дает возможность добавить определенный текст (в том числе и символы Unicode) в начало элемента веб-страницы. Это делается следующим образом:

  ```
  тег:before{
      content: "текст";
  }
  ```

Браузер Internet Explorer 6 не поддерживает данный псевдоэлемент.

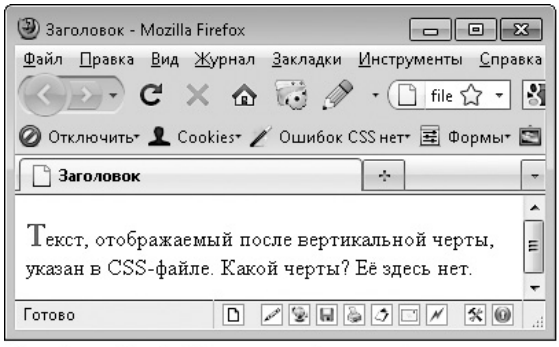
- `:first-letter` — позволяет применить свойства CSS к первому символу текста, написанного в данном элементе веб-страницы. Как правило, данная возможность используется для создания буквицы (рис. 2.2).




```
<p class="testp">Текст,
 отображаемый после
 вертикальной черты,
 указан в CSS-файле.
 Какой черты? Этой! |</p>
```

```
p.testp:after{
    content: " \20aa привет!";
}
```

Рис. 2.1. Результат использования псевдоэлемента :after




```
p.testp:first-letter{
    color: #ff0000;
    font-size: 18pt;
    font-weight: bold;
}
```

```
<p class="testp">Текст, отображаемый после вертикальной черты, указан в
 CSS-файле. Какой черты? Её здесь нет.</p>
```

Рис. 2.2. Создаем буквицу средствами псевдоэлемента :first-letter

- `:first-line` — позволяет применить свойства CSS к первой строке текста, написанного в данном элементе веб-страницы. Для данного псевдоэлемента разрешено использовать лишь следующие CSS-свойства:
  - все свойства, управляющие шрифтом;
  - все свойства, управляющие цветом;
  - свойства `clear`, `line-height`, `letter-spacing`, `text-decoration`, `text-transform`, `vertical-align` и `word-spacing`.

Группировка селекторов

В процессе верстки нередко возникает ситуация, когда одни и те же свойства CSS, имеющие одинаковые значения, присваиваются сразу нескольким селекторам. Повторение одного и того же кода увеличивает не только время верстки макета, но и размер будущего сайта, и, соответственно, время открытия его страниц.

Например:

```
p.new{
   color: #ff0000;
   font-weight: bold;
}
#footer div.notice{
   color: #ff0000;
   font-weight: bold;
}
```

С целью уменьшения кода и улучшения жизни верстальщика селекторы с одинаковым содержанием разрешено группировать. Для этого достаточно перечислить несколько селекторов через запятую, после чего единожды указать для всех перечисленных селекторов нужные свойства CSS:

```
p.new,
#footer div.notice{
   color: #ff0000;
   font-weight: bold;
}
```

В приведенном примере CSS-свойства `color` и `font-weight` присваиваются сразу двум селекторам: `p.new` и `#footer div.notice`.

Свойства CSS

Селекторы представляют собой оболочку, внутри которой находятся свойства CSS и их значения. С каждой версией языка CSS количество доступных свойств увели-

чивается. Но в отличие от HTML такого понятия, как устаревшее CSS-свойство, не существует. Если свойство есть, то оно всегда актуально и его можно применять.

Язык CSS развивается, и на данный момент последней версией языка CSS является CSS3. От предыдущей версии языка она отличается не только новыми свойствами, но и новыми видами селекторов, псевдоэлементов и псевдоклассов. Последние нововведения мы уже рассмотрели в предыдущей главе.

Как и следовало ожидать, все нововведения CSS 3 не поддерживаются браузером Internet Explorer 6 и другими устаревшими браузерами. Поэтому применять их не рекомендуется. Впрочем, существуют способы научить браузер Internet Explorer 6 понимать новые технологии, чем активно пользуются разработчики сайтов. О наиболее популярных способах будет рассказано далее в книге.

Возможные значения

Свойства CSS мы подробно рассмотрим в следующих разделах главы. Сейчас же просто поговорим о значениях, которые могут принимать свойства CSS.

В качестве значения может выступать ключевое слово, произвольная строка, число, размер в определенных единицах, цвет или адрес.

Ключевые слова. Большинство свойств CSS могут принимать лишь определенное значение из конечного множества возможных значений. Такие значения называются ключевыми словами. Примером могут служить следующие значения: `right`, `justify`, `auto`, `bold`, `solid` и т. д.

Все значения, представляющие собой ключевые слова, пишутся без кавычек:

```
*{
   font-weight: bold;
   border: 1px solid #ff0000;
}
```

Строки. Произвольные строки в качестве значений CSS используются очень редко. Но, несмотря на это, вы уже встречались со строками в значениях. Вспомните CSS-свойство `content`, которое применялось в псевдоэлементах `after` и `before`.

Строки в качестве значений CSS-свойств всегда пишутся в одинарных или двойных кавычках.

Числа. В качестве значения CSS-свойства может также выступать обычное число. Например, числом может задаваться «жирность» шрифта в свойстве `font-weight`.

Число может также использоваться вместо размера, то есть в значении CSS-свойства может указываться только число без определенной единицы измерения. В этом случае браузер будет использовать единицу измерения по умолчанию для данного свойства CSS. Однако правильнее всегда указывать единицу измерения.

Значения в виде чисел всегда задаются без кавычек.

Размер. Если необходимо указать какой-либо размер, используется значение в виде числа с определенной единицей измерения. Причем единица измерения приводится сразу после числа, без пробела: `1px`, `12pt`, `1.3em` и т. д.

Единицы измерения бывают относительными и абсолютными.

Относительные определяют размер относительно какого-либо элемента. Величина такой единицы измерения не является постоянной и может меняться в зависимости от условий: других свойств CSS, которые применяются к данному элементу веб-страницы, или технических характеристик компьютерного оборудования у посетителя (монитора).

Абсолютные единицы измерения существуют сами по себе, они знакомы нам из повседневной жизни, например миллиметр. Абсолютные единицы измерения всегда постоянны. Они во всех странах мира имеют один и тот же размер. Величина 1 миллиметра на измерительной линейке всегда одинакова, в какой бы стране вы ни купили измерительную линейку.

К относительным единицам измерения относятся следующие.

- `em` — размер относительно шрифта текущего элемента. Значение `1em` равно размеру шрифта текущего элемента, тогда как значение `1.2em` говорит о том, что размер элемента больше размера шрифта в 1,2 раза.
- `ex` — размер относительно высоты символа `x` (в нижнем регистре).
- `px` — размер в пикселах.
- `%` — размер в процентах. В этом случае размер определяется относительно родительского элемента или окна браузера, если родительским элементом является тег `body`. Таким образом, значение `100%` равно размеру родительского тега, тогда как значение `80%` говорит о том, что размер текущего элемента меньше размера родительского элемента на 20 %.

К абсолютным единицам измерения относятся:

- `in` — дюйм (1 дюйм = 2,54 см);
- `cm` — сантиметр;
- `mm` — миллиметр;

- `pt` — пункт (1 пункт = 1/72 дюйма);
- `pc` — пика (1 пика = 12 пунктов).

Чаще всего при верстке используются следующие единицы измерения:

- `px` — для указания ширины, высоты и толщины элемента или его границы;
- `em` — для задания межстрочного интервала;
- `pt` — для указания размера шрифта;
- `%` — для задания величины относительно родительского элемента (ширины, высоты, толщины, межстрочного интервала, размера шрифта и т. д.), то есть для того, чтобы величина данного элемента автоматически изменялась при изменении величины родительского элемента.

Единицу измерения следует указывать всегда. Исключение составляет значение `0`. Для него можно не приводить единицу измерения.

Значения в виде размеров всегда задаются без кавычек.

Цвет. Существует три способа указать значение в виде цвета.

Самый простой — воспользоваться специальным ключевым словом, представляющим собой английское название цвета: `black`, `white` и т. д. Полный перечень возможных ключевых слов представлен в табл. 2.1.

Таблица 2.1. Ключевые слова, определяющие цвет

Ключевое слово	Цвет	Шестнадцатеричный код
white	Белый	#ffffff или #fff
silver	Серый	#c0c0c0
gray	Темно-серый	#808080
black	Черный	#000000 или #000
maroon	Темно-красный	#800000
red	Красный	#ff0000 или #f00
orange	Оранжевый	#ffa500
yellow	Желтый	#ffff00 или #ff0
olive	Оливковый	#808000
lime	Светло-зеленый	#00ff00 или #0f0
green	Зеленый	#008000
aqua	Голубой	#00ffff или #0ff
blue	Синий	#0000ff или #00f
navy	Темно-синий	#000080
teal	Сине-зеленый	#008080
fuchsia	Розовый	#ff00ff или #f0f
purple	Фиолетовый	#800080

Более сложный способ задать цвет — указать шестнадцатеричный код, определяющий цветовые компоненты нужного цвета. Как легко догадаться из названия, шестнадцатеричный код указывается в шестнадцатеричной системе счисления, числа в которой могут состоять из цифр от 1 до f (1, 2, 3, 4, 5, 6, 7, 8, 9, 0, a, b, c, d, e, f).

Итак, шестнадцатеричный код цвета представляет собой значение в следующем формате: `#aabbвв`, где:

- `aa` — две цифры, каждая от 0 до f, определяющие красный компонент цвета;
- `бб` — две цифры, каждая от 0 до f, задающие зеленый компонент цвета;
- `вв` — две цифры, каждая от 0 до f, определяющие синий компонент цвета.

Каждый компонент цвета может иметь значение от `00` до `ff` в шестнадцатеричной системе счисления. В десятичной же системе счисления каждый компонент цвета может иметь 256 тоновых градаций, от 0 до 255. Соответственно, всего с помощью шестнадцатеричного кода можно указать 255 × 255 × 255 = 16 581 375 оттенков цвета, что значительно больше 17 цветов, которые можно указать с помощью ключевых слов, перечисленных в табл. 2.1.

Пример шестнадцатеричного кода: `#ffff00`. Как видно из табл. 2.1, данный код определяет желтый цвет.

Запоминать шестнадцатеричные коды цветов не нужно. Как правило, необходимый цвет выбирается в программе Adobe Photoshop. В ней же можно узнать шестнадцатеричный код этого цвета (рис. 2.3).

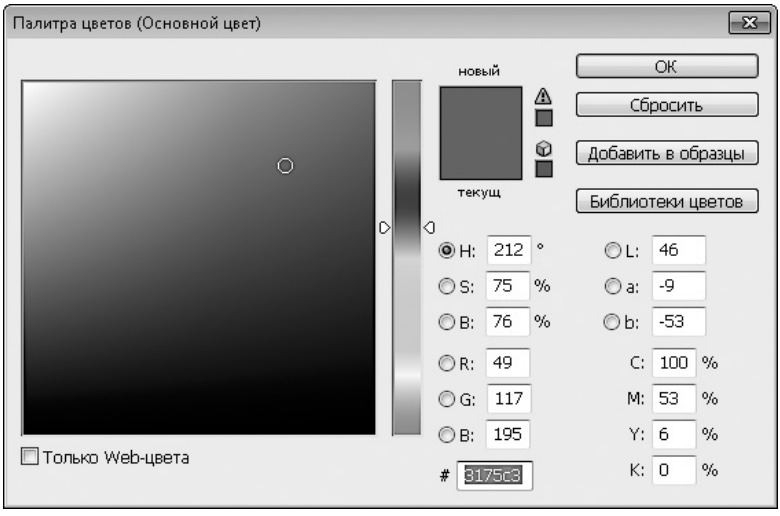


Рис. 2.3. Определение шестнадцатеричного кода цвета с помощью программы Adobe Photoshop

Третий способ задания цвета подобен предыдущему. Только вместо цветовых компонентов в шестнадцатеричной системе счисления используются те же цветовые компоненты, но в десятичной системе счисления. Итак, основной синтаксис задания цвета в RGB следующий: `rgb(aa, бб, вв)`, где:

- `aa` — число от 0 до 255, определяющее красный компонент цвета;
- `бб` — число от 0 до 255, задающее зеленый компонент цвета;
- `вв` — число от 0 до 255, определяющее синий компонент цвета.

Выяснить значения цветовых компонентов в системе RGB также можно с помощью программы Adobe Photoshop. Для этого используется тот же диалог, который показан на рис. 2.3. Только смотреть нужно на значения, написанные рядом с переключателями `R`, `G` и `B`.

В формате `rgb(aa, бб, вв)` вместо чисел в десятичной системе счисления можно использовать проценты. Например, `rgb(100%, 20%, 33%)`. В этом случае 100 % будет соответствовать числу 255, а 0 % — числу 0.

Адрес. Для некоторых свойств CSS в качестве значения нужно указывать относительный или абсолютный адрес файла в Интернете. Например, если вы хотите задать фоновую картинку для какого-либо элемента, будет нелегко обойтись без адреса.

Основной формат записи адреса следующий: `url("адрес")`. Например, `url("../img/bg_body.png")`.

Заключать адрес в двойные или одинарные кавычки обязательно только в том случае, если в адресе есть пробелы. Но, несмотря на это, рекомендуется всегда использовать кавычки.

Стандартные значения

Среди всего разнообразия значений существует несколько таких, которые применяются для всех либо для большинства свойств CSS.

В первую очередь следует упомянуть ключевое слово `inherit`. Оно говорит о том, что значение свойства у текущего тега должно быть точно таким же, как значение этого же свойства у родительского тега, то есть значение наследуется от родительского тега.

Однако следует учитывать, что ключевое слово `inherit` не распознается версиями 6 и 7 браузера Internet Explorer.

Многие свойства CSS поддерживают также ключевое слово `auto`. Оно позволяет отменить значение, которое ранее было задано для данного тега. При использовании данного значения CSS-свойству будет присвоено значение по умолчанию.

Наследование свойств CSS

Некоторые свойства CSS применяются не только к селектору, для которого вы их указали, но и к другим, дочерним по отношению к данному селектору, тегам. Другие же свойства CSS применяются только к указанному селектору. Так проявляет себя механизм наследования.

Все очень просто. Существуют свойства CSS, которые наследуются, и CSS-свойства, которые никогда не наследуются.

Если свойство наследуется, оно будет применяться для всех тегов, вложенных внутрь тега, для которого данное свойство изначально указано. Но только в том случае, если для вложенного тега явно не задано другое значение данного CSS-свойства. Например, посмотрите на данный CSS-код:

```
body{ color: #444; }

p.warning{ color: #ff0000; }
```

Он позволяет задать тексту, размещенному во всех тегах на веб-странице, черный цвет. И только текст, расположенный в теге `p` с классом `warning`, будет отображаться красным цветом.

Вот мы и подошли к причине, по которой в CSS иногда используется селектор `body`. С его помощью через наследование можно ненавязчиво задать CSS-свойства всем тегам на веб-странице.

Впрочем, того же эффекта можно добиться и с помощью селектора `*`.

Какие свойства CSS наследуются, а какие нет? В приложении 4 вы сможете найти подробный ответ на поставленный вопрос. Если вкратце, то наследуется большинство шрифтовых свойств, а также цвет текста.

Универсальные свойства

В CSS существует такое понятие, как универсальное свойство. Под ним понимается свойство CSS, которое заменяет собой сразу несколько отдельных свойств. Например, свойство `background` объединяет в себе следующие свойства: `background-color`, `background-image`, `background-position`, `background-repeat`.

То есть следующий CSS-код:

```
h1.title div.center{

  background-color: #fff;

  background-image: url('../img/test.png');
```

```
   background-repeat: no-repeat;
   background-position: left top;
}
```

можно безболезненно заменить данным кодом:

```
h1.title div.center{
   background: #fff url('../img/test.png') no-repeat left top;
}
```

Использование универсальных свойств позволяет ускорить процесс создания сайта и уменьшить размер CSS-файла, поэтому пренебрегать ими не стоит. Далее в книге мы будем рассматривать преимущественно универсальные свойства.

Подключение CSS к HTML

Перед тем как приступить к рассмотрению свойств CSS, следует научиться использованию CSS не только в теории, но и на практике, а именно рассмотреть способы, которыми можно применять CSS в HTML-документе.

Внутренние стили

Самый простой способ применить CSS в HTML-документе — это воспользоваться атрибутом `style`, который поддерживается всеми тегами языка HTML. В этом случае даже не нужно указывать селектор, поскольку свойства CSS применяются к тому тегу, в атрибуте `style` которого вы их задаете.

Итак, варианты синтаксиса данного способа задания CSS следующие:

```
style="свойство: значение;"
style="свойство: значение; свойство: значение;"
```

Например:

```
<p style="font-size: 9pt; font-weight: bold; line-height:
1.3em;">пример текста</p>
```

Любой внутренний стиль имеет вес, равный 1000. Поэтому значения, указанные с помощью данного способа, всегда замещают собой значения свойств CSS, указанные другими способами (даже с использованием ключевого слова `!important`).

Несмотря на такое преимущество внутренних стилей, применять их следует лишь в крайнем случае, так как это сводит на нет все плюсы CSS:

- усложняет верстку макета — структура HTML-документа засоряется лишним кодом, из-за чего поиск закрывающих тегов, вложенных элементов и т. д. усложняется;
- увеличивает размер веб-страницы — свойства CSS указаны в самой HTML-странице, и скачивать их приходится при каждом открытии страницы.

Глобальные стили

Под глобальными стилями понимаются свойства CSS, которые указаны внутри парного тега `style`. Сам тег `style` можно использовать лишь внутри тега `head`.

Общий формат записи данного тега следующий:

```
<head>
   <style type="text/css">
      селектор{
         свойство: значение;
      }
   </style>
</head>
```

Как видно из примера, в глобальных стилях свойства CSS применяются к селектору. И без указания селектора их нельзя использовать.

Применение глобальных стилей предпочтительнее использования внутренних стилей. Но все равно применять их стоит лишь в крайнем случае, так как при использовании глобальных стилей свойства CSS по-прежнему указываются внутри HTML-документа. А значит, посетителю приходится скачивать их при каждом открытии веб-страницы.

Связанные стили

Связанные стили являются наиболее оптимальным способом применения CSS. В этом случае весь CSS-код находится в отдельном файле (или нескольких файлах), а в HTML-документе указывается ссылка на этот CSS-файл.

Благодаря связанным стилям достигаются следующие преимущества:

- структура HTML-документа становится более наглядной;
- файл с CSS-кодом скачивается лишь при первом открытии сайта, после чего браузер берет файл из своего кэша, не скачивая его снова. Благодаря этому скорость открытия страниц сайта увеличивается.

Итак, весь CSS-код должен находиться в отдельном файле с расширением CSS. Как и HTML-документ, это обычный текстовый файл. Но только расширение у него изменено с TXT на CSS.

Сам CSS-файл должен быть расположен в Интернете. Как правило, CSS-файлы размещают в папке `CSS`. Делается это для удобства; никаких особых правил, требующих этого, не существует.

После того как CSS-файл создан, его нужно подключить к HTML-документу. Для этого в тег `head` HTML-документа добавляют одинарный тег `link`:

```
<link rel="stylesheet" type="text/css" media="all" href="путь
и имя CSS-файла">
```

В дальнейшем мы будем использовать именно этот способ подключения CSS к HTML-странице. Рассмотрим простой пример использования связанных стилей (листинг 2.1).

Листинг 2.1. Использование связанных стилей

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <link type="text/css" rel="stylesheet" href="css/style.css" media="all" />
</head>
<body>
    <p class="testp">Этот текст выглядит иначе, чем кажется из вашего редактора
HTML.</p>
</body>
</html>
```

В данном примере к HTML-документу подключается CSS-файл `style.css`, расположенный в папке `css`, которая, в свою очередь, размещена в той же папке, что и наш HTML-документ.

Если вы откроете файл `style.css` в Блокноте, то сможете увидеть следующее содержимое:

```
p.testp{
    font-size: 9pt;
    font-style: italic;
}
p.testp:first-letter{
    color: #ff0000;
    font-size: 18pt;
    font-weight: bold;
}
```

На рис. 2.4, *внизу* показано, как выглядит наш пример в браузере. А рис. 2.4, *вверху* отображает, как выглядел бы HTML-документ без подключения CSS.

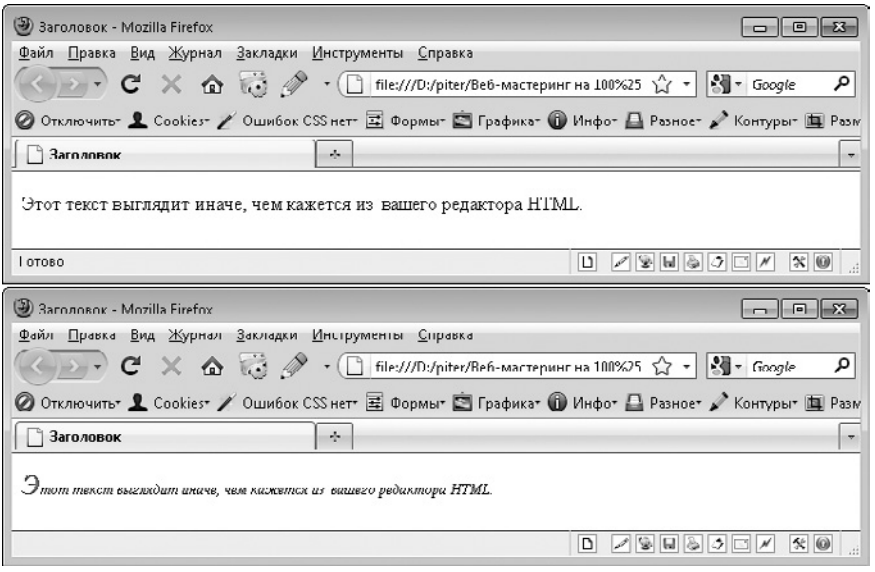


Рис. 2.4. Страница без применения CSS *(вверху)* и с использованием CSS *(внизу)*

Атрибут media. Возможно, вы заметили, что тег `link` имеет атрибуты, которые нужно указывать при подключении CSS. Описывать большинство из них не имеет

смысла, поскольку они обязательны, а их значения не отличаются большим разнообразием — для подключения CSS используется только один вариант значения и других быть не может.

Однако атрибут `media` является приятным исключением из правил. Чаще всего вы не будете его указывать, но в редких случаях его использование может пригодиться.

Атрибут `media` позволяет задать, в каком случае нужно подключать данный CSS-файл к HTML-документу. Точнее, он позволяет определить тип носителя (вид устройства), для которого предназначен данный CSS-файл.

Если вы не укажете атрибут `media` либо в качестве его значения зададите `all`, то CSS-файл будет подключаться всегда, независимо от устройства, на котором запущен браузер и открыта данная веб-страница.

Помимо значения `all`, атрибут `media` может принимать следующие значения:

- `print` — принтеры и другие печатающие устройства; то есть CSS-файл будет подключен к HTML-документу только при распечатке веб-страницы на принтере, благодаря чему вы можете управлять видом веб-страницы после печати, удаляя с веб-страницы ненужную информацию и правильно форматируя нужную;
- `aural` — речевые синтезаторы, а также программы для воспроизведения текста вслух (например, речевые браузеры);
- `braille` — устройства, основанные на системе Брайля, которые предназначены для слепых людей;
- `handheld` — карманные компьютеры и аналогичные им аппараты;
- `projection` — проектор;
- `screen` — экран монитора;
- `tv` — телевизор.

ПРИМЕЧАНИЕ

Тип носителя также можно задавать с помощью команды @media, указывать которую нужно либо в CSS-файле, либо в теге style. Ее синтаксис подобен применению селекторов:

```
@media тип_носителя {
  селектор{
    CSS-свойство: значение;
  }
}
```

В качестве примера рассмотрим следующий HTML-код (листинг 2.2).

Листинг 2.2. Использование отдельных стилей для разных типов носителей

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <link type="text/css" rel="stylesheet" href="css/style.css" media="screen" />
    <link type="text/css" rel="stylesheet" href="css/print.css" media="print" />
</head>
<body>
    <p><img alt="" src="img/error2.png" />На печати этот текст выглядит иначе, чем
в вашем браузере.</p>
</body>
</html>
```

Файл `style.css`:

```
p img{
    float: left;
    padding-right: 11px;
}
p{
    font-size: 11pt;
    font-style: italic;
    width: 233px;
}
```

Файл `print.css`:

```
p{ font-size: 10pt; }
p img{ display: none; }
```

Файлы `style.css` и `print.css` отличаются друг от друга. Соответственно, и вид итоговых страниц различается (рис. 2.5).

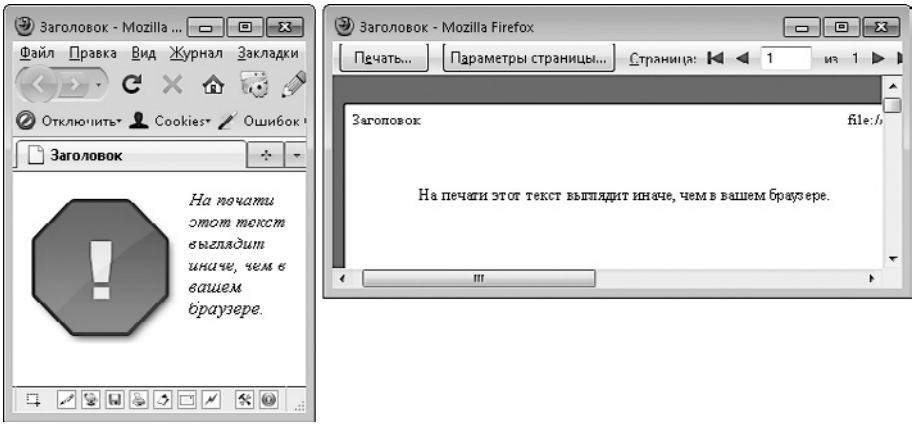


Рис. 2.5. Вид веб-страницы: в браузере *(слева)*, при печати *(справа)*

Импорт CSS

И последний способ подключения CSS — возможность импорта CSS внутрь связанных или глобальных стилей. В этом случае код CSS хранится в отдельном CSS-файле. Данный способ не имеет премуществ по сравнению с использованием связанных стилей. Однако сам механизм подключения CSS к HTML-документу отличается.

Для импорта CSS применяется команда `@import`, которая имеет следующие варианты синтаксиса:

```
@import url("имя файла") типы носителей;

@import "имя файла" типы носителей;
```

Под `типом носителя` понимается значение, аналогичное атрибуту `media` тега `link`. При этом через запятую можно указать несколько значений. Например: `@import "/style/smart.css" print, handheld;`.

Однако следует помнить, что браузер Internet Explorer 6 игнорирует тип носителя, указанный в команде `@import`.

Как уже говорилось, импортировать CSS можно только в глобальный или связанный стиль.

Глобальный стиль. Для импорта в глобальный стиль необходимо внутрь тега `style` поместить код:

```
@import url("путь и имя CSS-файла");
```

Например:

```
<head>
  <title>Пример</title>
  <style type="text/css">
   @import url("css/header.css");
   H1 {
    font-size: 120%;
    color: #ff0000;
   }
  </style>
 </head>
```

Связанный стиль. При использовании связанных стилей импорт производится той же командой:

```
@import "путь и имя CSS-файла";
```

Но указывать эту команду нужно не в HTML-документе, а в CSS-файле, который вы связываете тегом `link`.

Программное обеспечение

Как и для HTML, для CSS существуют различные классы программ, облегчающих работу верстальщика.

Редакторы CSS

Хоть CSS-файлы и можно создавать в обычном Блокноте, лучше для их создания использовать специальные программы, облегчающие работу верстальщика. Речь идет о редакторах CSS.

Если вы пользуетесь каким-либо полнофункциональным редактором HTML, то отдельный редактор CSS вам не нужен, поскольку, скорее всего, редактор CSS уже встроен в состав вашего редактора HTML. Например, именно так обстоят дела в программе phpDesignerPro, которую мы упоминали в главе 1.

Рассмотрим на примере программы phpDesignerPro основные преимущества использования редактора CSS (рис. 2.6):

- подчеркивание неизвестных свойств CSS и других ошибок (рис. 2.6, *слева вверху*);
- упрощение навигации по CSS путем создания закладок на все описанные классы и идентификаторы (рис. 2.6, *справа вверху*);
- автозавершение свойств CSS, а также их значений (рис. 2.6, *внизу*).

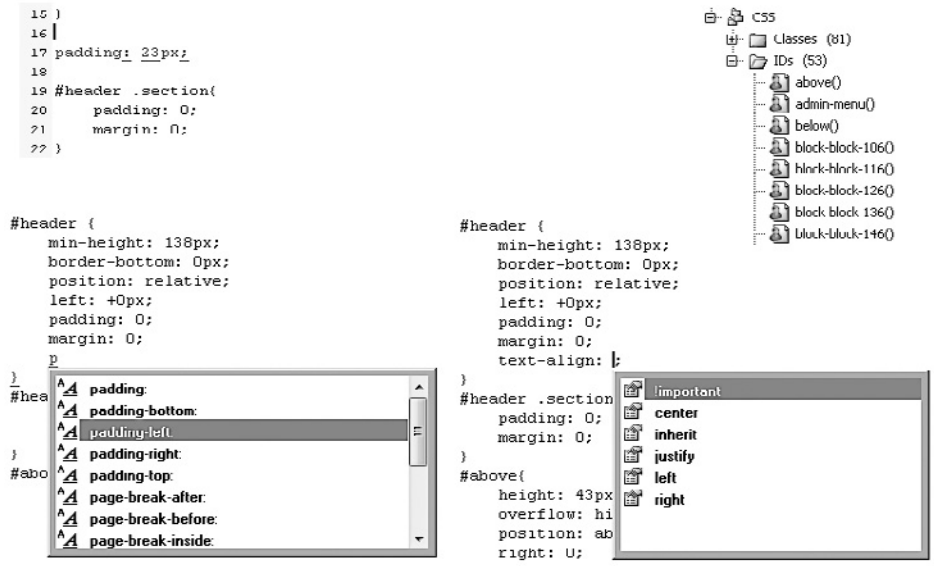


Рис. 2.6. Преимущества редактора CSS

Если в состав вашего редактора HTML не входит редактор CSS, вы можете подобрать себе отдельный редактор CSS. Однако среди бесплатных вариантов выбор небольшой.

- Free CSS Toolbox (http://www.blumentals.net/csstool/). Небольшой редактор CSS, содержащий в себе все основные возможности своих собратьев, а кроме того, CSS-валидатор и компрессор.
- SnapCSS (http://www.improvingcode.com/snapcss/). Редактор CSS, умеющий автоматически форматировать и стандартизировать ваш код.
- TopStyle (http://www.topstyle4.com/). Триал-версия данного редактора CSS не ограничена по времени, поэтому данный редактор также можно условно отнести к бесплатным.

- CodeLobster PHP Edition (http://www.codelobster.com/). Бесплатный редактор CSS/HTML/PHP/JavaScript.
- Stylizer (http://www.stylizerapp.com/). Визуальный редактор CSS-кода, позволяющий просматривать изменения на веб-странице в режиме реального времени. Он также дает возможность выделять элементы на веб-странице щелчком кнопки мыши.

Валидация CSS

Код CSS является не менее важной частью веб-страницы, чем сам HTML-документ. Поэтому нет ничего удивительного в существовании валидаторов, позволяющих проверить правильность написания CSS-кода. Такие валидаторы дают возможность проверить код CSS на соответствие спецификациям CSS2.1 или CSS3, благодаря чему можно быстро обнаружить ошибочные CSS-свойства или их значения.

Для валидации CSS можно воспользоваться сервисом, расположенным по адресу http://jigsaw.w3.org/css-validator/ (рис. 2.7). Он позволяет:

- ввести CSS-код (или HTML-код вместе с CSS) для проверки в текстовую область на сайте;
- указать путь к CSS-файлу, который нужно проверить, в Интернете;
- выбрать для проверки CSS-файл, расположенный на вашем компьютере.

Вспомогательные сервисы

Помимо редакторов и валидаторов, существуют и другие сервисы и программы для работы с CSS.

Форматирование кода. Сервис http://www.cssportal.com/format-css/ поможет вам отформатировать CSS-код удобным вам способом. Он поддерживает четыре способа форматирования.

- Formatted CSS (Форматированный CSS) — свойства CSS сдвигаются вправо на четыре пробела, между селекторами добавляется одна пустая строка:

```
body{
    margin: 0;
    padding: 0;
    font-family: Arial, Tahoma, Myriad, sans-serif;
    vertical-align: top;
```

```
    font-weight: normal;
}

p, div, td, input, textarea{
    font-size: 12pt;
    color: #000;
    line-height: 1.3em;
}
```

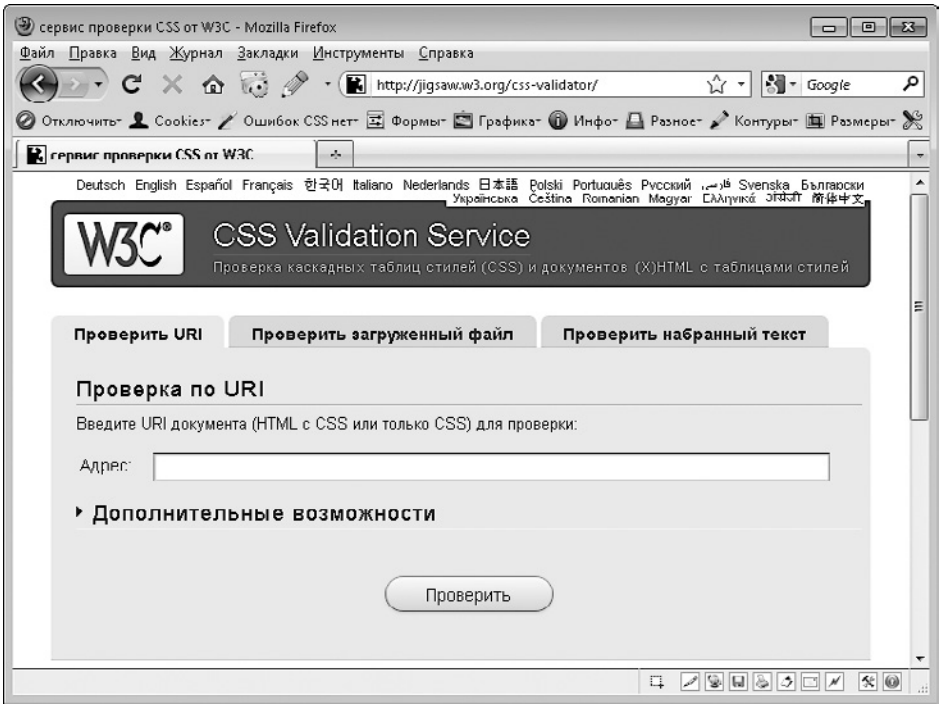


Рис. 2.7. Сервис валидации CSS

- Properties in Alphabetical Order (Свойства в алфавитном порядке) — свойства CSS сдвигаются вправо на четыре пробела, между селекторами добавляется одна пустая строка, все свойства внутри селектора сортируются в алфавитном порядке:

```
body{
    font-family: Arial, Tahoma, Myriad, sans-serif;
    font-weight: normal;
    margin: 0;
```

```
    padding: 0;
    vertical-align: top;
}

p, div, td, input, textarea{
    color: #000;
    font-size: 12pt;
    line-height: 1.3em;
}
```

- Longest Property to Shortest (Лесенкой) — свойства CSS сдвигаются вправо на четыре пробела, между селекторами добавляется одна пустая строка, все свойства внутри селектора сортируются по длине (вверху самые длинные строки, внизу самые короткие):

```
body{
    font-family: Arial, Tahoma, Myriad, sans-serif;
    font-weight: normal;
    vertical-align: top;
    padding: 0;
    margin: 0;
}

p, div, td, input, textarea{
    line-height: 1.3em;
    font-size: 12pt;
    color: #000;
}
```

- Compact (Компактно) — селекторы и свойства CSS записываются в одну строку (каждый селектор с новой строки):

```
body{margin: 0;padding: 0;font-family: Arial, Tahoma, Myriad,
sans-serif;vertical-align: top;font-weight: normal;}
p, div, td, input, textarea{font-size: 12pt;color: #000;line-
height: 1.3em;}
```

Минимизация кода. Чем меньше символов содержит CSS-файл, тем быстрее он будет загружаться. Эта истина банальна, однако следовать ей не так-то просто.

Казалось бы, ничего сложного — записать все селекторы и свойства CSS в одну строку, удаляя лишние пробелы и табуляции. Но как тогда редактировать CSS? Разбираться и CSS-файле и редактировать его становится весьма затруднительно.

Поэтому и были придуманы сервисы минимизации кода. Создавая CSS, верстальщик может пользоваться любым удобным для себя вариантом форматирования. И только перед размещением CSS-файла в Интернете используется один из сервисов минимизации кода, создающий оптимизированный вариант CSS-файла.

Наиболее популярны следующие сервисы минимизации кода:

- http://tools.w3clubs.com/cssmin/ — выполняет минимизацию кода CSS в автоматическом режиме;
- http://www.csscompressor.com — дает возможность выбрать дополнительные способы, которыми будет выполнена минимизация кода CSS:
  - сжатие цветов (например, запись `#ffffff` заменяется аналогичной `#fff`);
  - сжатие насыщенности шрифта (у свойства `font-weight` значения в виде ключевых свойств заменяются числовыми значениями);
  - запись селекторов в нижнем регистре;
  - удаление лишних слешей;
  - удаление точки с запятой в последнем свойстве селектора;
- http://www.generateit.net/css-optimize — позволяет выбрать дополнительные способы, которыми будет выполнена минимизация кода CSS, а также различные возможности форматирования CSS:
  - сортировку селекторов по алфавиту;
  - сортировку свойств внутри селектора по алфавиту;
  - перегруппировку селекторов с целью уменьшения размера файла;
  - оптимизацию универсальных свойств CSS;
  - сжатие цветов (например, запись `#ffffff` заменяется аналогичной `#fff`);
  - сжатие насыщенности шрифта (у свойства `font-weight` значения в виде ключевых свойств заменяются числовыми значениями);
  - запись селекторов и свойств в нижнем регистре;
  - удаление лишних слешей;
  - удаление точки с запятой в последнем свойстве селектора;
  - удаление свойств, не существующих в спецификации CSS.

CSS-хаки

Одна и та же страница в разных браузерах и даже в различных версиях одного браузера может выглядеть по-разному. И в большей степени это касается именно CSS, а не HTML.

Разница в 1 пиксел в расположении элемента на странице может испортить дизайн веб-страницы (рис. 2.8, *внизу*). И такие случаи нередки. Особенно если вы верстаете с оглядкой на браузер Internet Explorer 6.

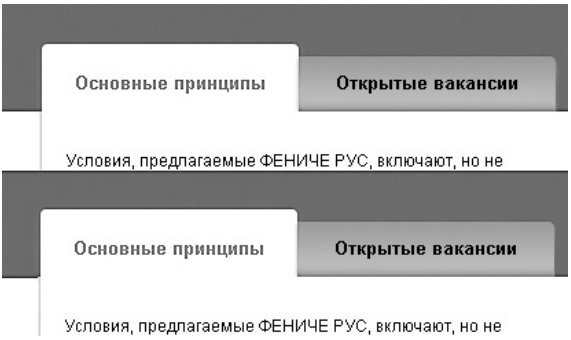


Рис. 2.8. В одном из браузеров блок с вкладками отображается на один пиксел выше *(внизу)*, чем в других *(вверху)*. Разница малозаметна, но и такая разница недопустима при верстке макета

Что же делать, если в большинстве браузеров элемент отображается так, как нужно, а в каком-то одном браузере возникают проблемы с ним? В этом случае можно попробовать переверстать страницу другим способом. Но если все известные вам способы верстки испробованы, остается только одно — прибегнуть к CSS-хакам.

Под CSS-хаком понимается:

- намеренное внесение ошибок в написание селектора или свойства CSS, чтобы данные селектор/свойство понимал только определенный браузер;
- особая запись селектора, которую понимают не все браузеры;
- создание отдельного CSS-файла, который будет подключаться только для определенной версии браузера Internet Explorer.

Условные комментарии

Условные комментарии позволяют задавать блоки кода, которые будут выполняться только в определенной версии браузера Internet Explorer. В остальных же версиях браузера Internet Explorer, а также в других браузерах их содержимое будет считаться обычным комментарием.

Существуют следующие варианты условных комментариев:

- `<!--[if IE]>содержимое<![endif]-->` — выполнять содержимое комментария, если браузером является любая версия Internet Explorer;
- `<!--[if IE 5]>содержимое<![endif]-->` — выполнять содержимое комментария, если браузером является версия Internet Explorer 5 (включая любые подверсии);
- `<!--[if IE 5.0]>содержимое<![endif]-->` — выполнять содержимое комментария, если браузером является Internet Explorer 5.0 (именно эта подверсия);
- `<!--[if IE 5.5]>содержимое<![endif]-->` — выполнять содержимое комментария, если браузером является Internet Explorer 5.5 (именно эта подверсия);
- `<!--[if IE 6]>содержимое<![endif]-->` — выполнять содержимое комментария, если браузером является версия Internet Explorer 6;
- `<!--[if IE 7]>содержимое<![endif]-->` — выполнять содержимое комментария, если браузером является версия Internet Explorer 7;
- `<!--[if IE 8]>содержимое<![endif]-->` — выполнять содержимое комментария, если браузером является версия Internet Explorer 8;
- `<!--[if IE 9]>содержимое<![endif]-->` — выполнять содержимое комментария, если браузером является версия Internet Explorer 9;
- `<!--[if lt IE версия]>содержимое<![endif]-->` — выполнять содержимое комментария, если в качестве браузера выступает Internet Explorer меньшей версии, чем указана; в качестве версии может выступать одно из значений, перечисленных выше;
- `<!--[if lte IE версия]>содержимое<![endif]-->` — выполнять содержимое комментария, если в качестве браузера выступает Internet Explorer версии, меньшей или равной указанной;
- `<!--[if gt IE версия]>содержимое<![endif]-->` — выполнять содержимое комментария, если в качестве браузера выступает Internet Explorer большей версии, чем указана;
- `<!--[if gte IE версия]>содержимое<![endif]-->` — выполнять содержимое комментария, если в качестве браузера выступает Internet Explorer версии, большей или равной указанной.

Условные комментарии работают только в HTML. Не пытайтесь применять их внутри CSS-файлов.

Итак, чаще всего условные комментарии используются для того, чтобы подключить отдельный CSS-файл для определенной версии браузера Internet Explorer. Делается это подобным образом:

```
<head>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <!--[if IE 6]>
        <link rel="stylesheet" type="text/css" href="ie6.css"
/>
   <![endif]-->
</head>
```

Internet Explorer

Браузер Internet Explorer, без сомнения, является самым популярным. Верстальщики знают и еще одну непреложную истину — больше всего проблем возникает именно в браузере Internet Explorer. Поэтому чаще всего CSS-хаки применяются именно к нему.

Internet Explorer 6. Чаще всего CSS-хаки необходимы для поддержки браузера Internet Explorer 6. И дело не только в различиях в понимании свойств CSS между данным браузером и другими, более современными браузерами. Обычно CSS-хаки используются для того, чтобы указать наборы свойств CSS, которые заменяли бы собой не поддерживаемые браузером Internet Explorer 6 CSS-свойства.

Чтобы написать селектор, CSS-свойства в котором будут применяться только браузером Internet Explorer 6, достаточно воспользоваться следующей конструкцией:

```
* html селектор {
      свойства
}
```

Казалось бы, вполне обычная конструкция. Однако все браузеры, за исключением Internet Explorer 6, считают ее ошибочной и игнорируют все свойства CSS, указанные в ней. Например:

```
div#footer{
   padding: 0 23px;
}
* html div#footer{
```

```
    padding: 0 22px;
}
```

Еще один хак, который мы рассмотрим, влияет на отдельное свойство. Он позволяет указать свойство, которое будет распознаваться всеми браузерами, кроме Internet Explorer 6.

Чтобы задать свойству значение, которое не будет применяться в браузере Internet Explorer 6, достаточно внутри любого селектора указать правило `свойство /**/: значение;`. Например, запись:

```
div#footer{
    padding: 0 22px;
    padding /**/: 0 23px;
}
```

аналогична рассмотренному нами выше коду.

Internet Explorer 7. Браузер Internet Explorer 7 является неким промежуточным этапом между устаревшими и современными браузерами. Обычно для него не проверяют корректное отображение верстки, делая это лишь для Internet Explorer 6 и 8.

Тем не менее вам может пригодиться знание CSS-хаков для Internet Explorer 7:

- `div[class^="myClass"] селектор {`
  `правила`
  `}`
- `*+html селектор {`
  `правила`
  `}`
- `*:first-child+html селектор {`
  `правила`
  `}`

Как можно заметить, все перечисленные хаки действуют на весь селектор, а не на отдельное CSS-свойство.

Существует также CSS-хак с обратным действием: позволяющий указать селектор, содержимое которого будет обрабатываться всеми браузерами за исключением Internet Explorer 7:

```
html>/**/body селектор {
      правила
}
```

Internet Explorer 8. Конкретно для всей линейки браузера Internet Explorer 8 CSS-хаков нет. Существует только для Internet Explorer 8 beta 2:

```
html:first-child селектор [attr|=a-b] {
      правила
}
```

Современные версии. Есть CSS-хак сразу для двух версий браузера Internet Explorer: 8 и 9. Данный хак действует на отдельное свойство (то есть позволяет указать свойство, значение из которого будет применяться только в браузерах Internet Explorer 8 и 9):

```
селектор{
      свойство: значение\0/;
}
```

Например:

```
.myClass{
      color:red\0/;
}
```

Mozilla Firefox

Для браузера Mozilla Firefox также существует CSS-хак. Он позволяет добавить любое количество селекторов, которые будут обрабатываться только данным браузером:

```
@-moz-document url-prefix() {
   селектор {
      правила
   }
}
```

Можно также воспользоваться следующим хаком:

```
*>селектор {
   правила
}
```

Opera

Данный хак позволяет указать любое количество селекторов, которые будут обрабатываться только браузером Opera:

```
@media all and (-webkit-min-device-pixel-ratio:10000),
not all and ( -webkit-min-device-pixel-ratio : 0 ) {
   селектор {
      правила
   }
}
```

Safari

Для браузера Safari можно использовать следующие CSS-хаки:

- ```
  body:first-of-type селектор {
     правила
  }
  ```
- ```
  селектор:not(:root:root) {
     правила
  }
  ```

Google Chrome

Для браузера Google Chrome можно применять следующий CSS-хак:

```
body:nth-of-type(1) селектор {
   правила
}
```

Смешанные хаки

Существует также набор CSS-хаков, работающих сразу для нескольких браузеров либо для нескольких версий одного браузера.

Safari 3.0 и Google Chrome:

```
@media screen and (-webkit-min-device-pixel-ratio:0) {
   селектор{
      правила
   }
}
```

Mozilla Firefox, Opera, Konqueror, Safari:

```
*|html селектор{
   правила
}
```

Mozilla Firefox, Internet Explorer 7 и выше:

```
html>body селектор {
   правила
}
```

Изучаем свойства: отступ

Вот мы и подошли вплотную к свойствам CSS. Начиная со следующего абзаца, мы будем учиться применять CSS в тех случаях, когда это необходимо.

Любой тег включает в себя следующие составляющие:

- свойство `margin` — некое пустое пространство за пределами тега, которое отделяет один тег от другого;
- свойство `border` — граница определенного цвета, которая обрамляет тег;
- свойство `padding` — некое пустое пространство внутри тега, которое отделяет содержимое тега от границы;
- само содержимое тега.

Проще всего изучить этот вопрос на примере.

На рис. 2.9 плавающему блочному тегу `div` под именем «тег А» задан светлый фон, а еще одному такому же тегу под именем «тег Б» задан темный фон.

Рис. 2.9. Два плавающих блочных тега div

Возможно, вы удивитесь, почему блочный тег не занимает всю ширину окна браузера. Дело в том, что он плавающий. Ему присвоено CSS-свойство `float`, благодаря чему вы можете задавать ширину блочного тега и он приобретает некоторые свойства, подобные свойствам строчных тегов. Об этом мы поговорим далее в книге.

Обратите внимание, что текст начинается не с самого верха блочного тега. Это никак не связано ни со свойством `padding`, ни со свойством `margin`. Просто под строку текста в HTML отводится место определенной высоты (так называемая высота строки). И по умолчанию высота строки несколько выше любой буквы используемого шрифта.

Как видно из рисунка, теги находятся вплотную друг к другу, так что между их фоном нет ни одного пиксела пустого пространства.

Кроме того, текст, расположенный внутри тега, начинается с самого начала тега, как слева, так и справа. А если бы не было высоты строки, то и сверху текст плотно прилегал бы к границе блочного тега.

padding

Основные сведения:

- значение по умолчанию: 0;
- наследуется: нет;

- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `padding: [значение | проценты] {1, 4} | inherit`.

ПРИМЕЧАНИЕ

Запись [значение | проценты] {1, 4} | inherit читается примерно так: в качестве значения данного свойства может выступать либо ключевое слово inherit, либо от одного до четырех написанных через пробел числовых значений или значений в процентах.

Свойство `padding` позволяет задать отступ от границы тега до начала его содержимого.

Свойство `padding` — универсальное. С его помощью можно задать отступ сразу для всех границ тега. В то же время существуют подобные свойства, позволяющие задать отступ только для следующих элементов:

- `padding-left` — левой границы тега;
- `padding-right` — правой границы тега;
- `padding-top` — верхней границы тега;
- `padding-bottom` — нижней границы тега.

Отступ задается числовым значением с указанием единицы измерения. Как правило, используется отступ в пикселах, то есть единица измерения `px`.

Кроме числового значения, может применяться ключевое слово `inherit`. Оно указывает браузеру, что отступ у данного тега такой же, как и у родительского тега. Однако использовать это значение не рекомендуется, так как браузер Internet Explorer версий 6 и 7 его не понимает.

Для свойств `padding-left`, `padding-right`, `padding-top` и `padding-bottom` задается только одно значение отступа. Например:

```
div {

   padding-left: 7px;

}
```

А для универсального свойства `padding` можно через пробел задать сразу четыре значения в такой последовательности: отступ сверху, отступ справа, отступ снизу, отступ слева. Таким образом, отступы задаются по часовой стрелке. Например:

```
div {

   padding: 7px 3px 23px 11px;

}
```

Представленный блок кода определяет отступ сверху 7 пикселов, справа — 3 пиксела, снизу — 23 пиксела и слева — 11 пикселов. На рис. 2.10 показано, как будет выглядеть знакомый нам блочный тег с именем «тег Б» после применения данного CSS-кода.

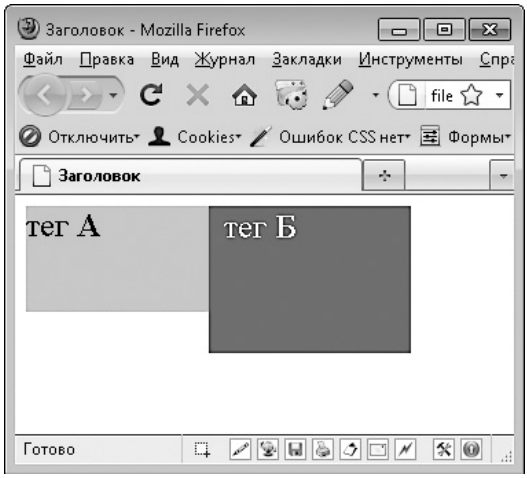


Рис. 2.10. Применение свойства padding

Прежде всего в глаза бросается изменившаяся высота и ширина «тега Б». Дело в том, что в данном примере мы задали фиксированную ширину и высоту для двух этих тегов. Однако отступ, определяемый тегом `padding`, в состав ширины и высоты не входит.

Отсюда можно сделать вывод, что реальная ширина тега на странице равна ширине тега плюс заданные ему размеры `padding` слева и размеры `padding` справа. Соответственно, реальная высота тега на странице равна заданной ему высоте плюс размер `padding` сверху и размер `padding` снизу.

Итак, `padding` позволяет задать пустое пространство между содержимым тега и его краями, тем самым сделать текст более читабельным, а дизайн более привлекательным.

Существует несколько особенностей, с которыми вы можете встретиться, задавая `padding` строчному тегу внутри блока текста. Посмотрите на рис. 2.11.

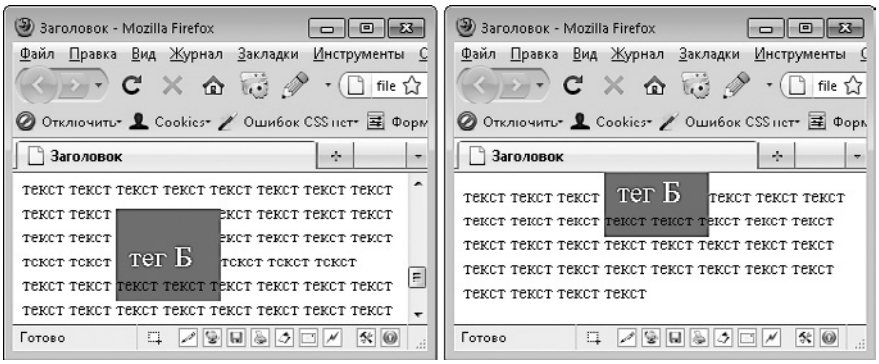


Рис. 2.11. Padding для строчных тегов внутри текста

Мы можем сделать следующие выводы:

- `padding` слева и справа работает корректно;
- если над строчным тегом, для которого задан `padding`, есть текст, то `padding` сверху отображается корректно;
- если над строчным тегом, для которого задан `padding`, нет текста (примыкает к границе родительского элемента), то `padding` сверху обрезается (см. рис. 2.11, *справа*);
- `padding` не обтекается текстом снизу, однако наличие фона показывает, что сам `padding` браузером учитывается.

margin

Основные сведения:

- значение по умолчанию: 0;
- наследуется: нет;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `margin: [значение | проценты | auto] {1,4} | inherit.`

С помощью свойства `margin` вы можете задать внешний отступ до границы тега, то есть отделить тег от других тегов на странице (рис. 2.12).

Как видно на рис. 2.12, «тег Б» сместился относительно «тега А». Это произошло именно из-за того, что для «тега Б» были заданы `margin` слева и сверху. При этом, как видно из размеров фона, высота и ширина тега не изменились.

Рис. 2.12. Использование свойства margin

Как и `padding`, свойство `margin` является универсальным свойством, в котором через пробел задаются отступы сразу для четырех сторон тега:

- верхней;
- правой;
- нижней;
- левой.

Таким образом, отступы указываются по часовой стрелке.

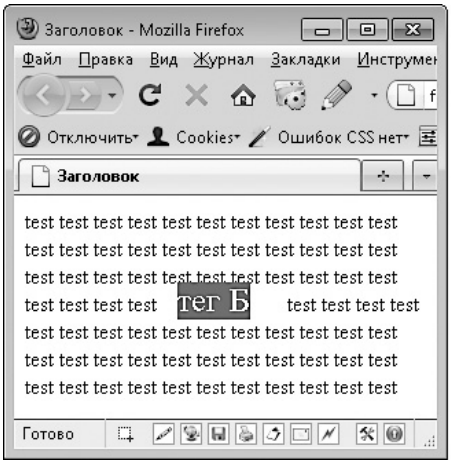
Свойство `margin` заменяет собой свойства `margin-top`, `margin-right`, `margin-bottom` и `margin-left`.

Отступы могут задаваться как в виде числа с единицей измерения (как правило, `px`), так и ключевым словом `inherit`. Данное ключевое слово говорит о том, что величина отступа должна быть точно такой же, как и у родительского тега. Однако Internet Explorer версий 6 и 7 не понимает значения `inherit`.

Если вы решили задать `margin` строчному тегу, будьте готовы к тому, что внешние отступы вверху и внизу будут проигнорированы (рис. 2.13). При этом внешние отступы слева и справа будут работать.

Отступы по умолчанию

Неопытный верстальщик может крайне удивиться, увидев, что веб-страница начинается не с начала экрана. Например, задав какому-либо тегу внутри страницы фон и заметив, что фон начинается не с начала экрана.




```
.div_b{
    margin: 34px 23px 23px 11px;
    background: #777;
    color: #fff;
    line-height: 0.7em;
    font-size: 18pt;
}
```

Рис. 2.13. Особенности margin внутри блока текста

Чтобы потом не ломать голову над этой загадкой, сразу выясним, что для тега `body` по умолчанию заданы отступы. Сделано это, наверное, для того, чтобы написанный на веб-странице текст по умолчанию не прижимался к краю экрана.

Если вам нужно избавиться от этих отступов, достаточно в CSS задать следующий селектор:

```
body{
    padding: 0;
    margin: 0;
}
```

Изучаем свойства: тип тега

В данном разделе рассмотрим свойства, влияющие на тип тега и само его существование.

display

Основные сведения:

- значение по умолчанию: `inline`;
- наследуется: нет;

- применяется: ко всем элементам;
- версия CSS: 2;
- синтаксис: `display: block | inline | inline-block | inline-table | list-item | none | run-in | table | table-caption | table-cell | table-column-group | table-column | table-footer-group | table-header-group | table-row | table-row-group`.

Свойство `display` позволяет изменить тип любого тега либо вообще удалить его со страницы. С его помощью строчный тег можно сделать блочным и наоборот. Но это далеко не все возможности данного свойства.

Тип тега задается одним из следующих значений данного свойства.

block. Преобразует тег в блочный. С его помощью вы можете сделать так, чтобы теги `span`, `a`, `li` вели себя, как будто они являются блочными тегами (рис. 2.14).

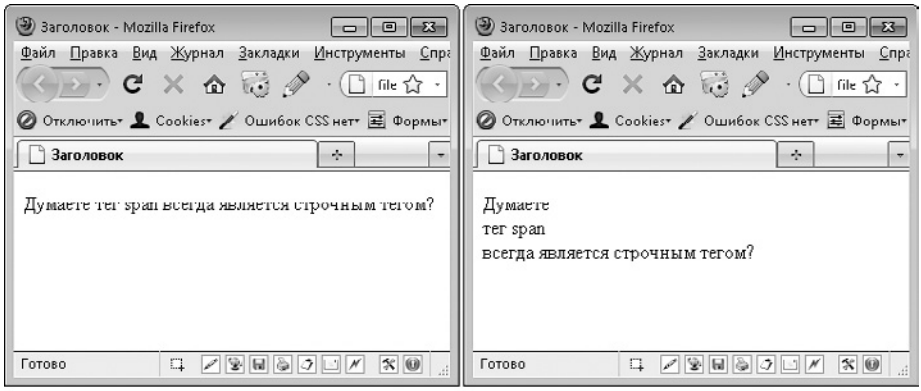


Рис. 2.14. Используем правило display: block;

Для тега `li` версии 6 и 7 браузера Internet Explorer понимают значение `block` как значение `list-item`.

Это значение может применяться, если вы хотите в качестве ссылки использовать изображение, поверх которого располагается произвольный текст. В этом случае достаточно сделать следующее.

1. Создать тег `a` с текстом внутри.
2. Сделать тег `a` блочным.

3. Задать тегу необходимые размеры, центрирование текста и в качестве фона указать нужное вам изображение.

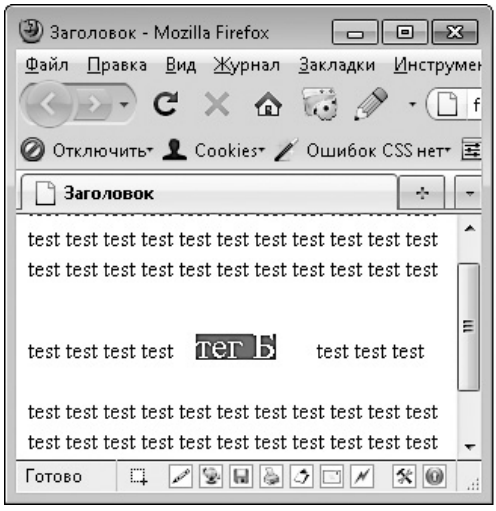
Также значение `block` может использоваться для создания пунктов меню, расположенных в одну строку.

inline. Преобразует тег в строчный. С его помощью вы можете сделать так, чтобы теги `div` и другие блочные теги вели себя так, будто они являются строчными.

inline-block. Очень полезное значение, позволяющее создать тег строчного типа, который поддерживает отдельные возможности блочных тегов.

В частности, тегам данного типа можно задавать ширину и высоту, тогда как к обычным строчным тегам свойства `width` и `height` неприменимы.

Кроме того, если вы еще помните, при рассмотрении свойств `margin` и `padding` мы заметили, что для строчных тегов, расположенных в блоке текста, отступы сверху и снизу игнорируются. Эта особенность не затрагивает тегов, для которых задано значение `inline-block` (рис. 2.15).




```
.div_b{
    margin: 34px 23px 23px 11px;
    display: inline-block;
    background: #777;
    color: #fff;
    line-height: 0.7em;
    font-size: 18pt;
}
```

Рис. 2.15. Применение margin к тегу типа inline-block внутри блока текста

Версии 6 и 7 браузера Internet Explorer корректно применяют значение `inline-block` только для тегов строчного типа.

inline-table. Позволяет создать тег, который будет вести себя как таблица (тег `table`), но при этом окажется строчного типа, то есть его будет обтекать содержимое веб-страницы и другие теги.

Данное значение не поддерживается 6-й и 7-й версиями браузера Internet Explorer.

list-item. Преобразует тег в элемент списка — в тег `li`, то есть в блочный тег с маркером слева от содержимого.

none. Полностью удаляет тег с веб-страницы (рис. 2.16). В любой момент удаленный таким образом тег можно отобразить с помощью JavaScript.

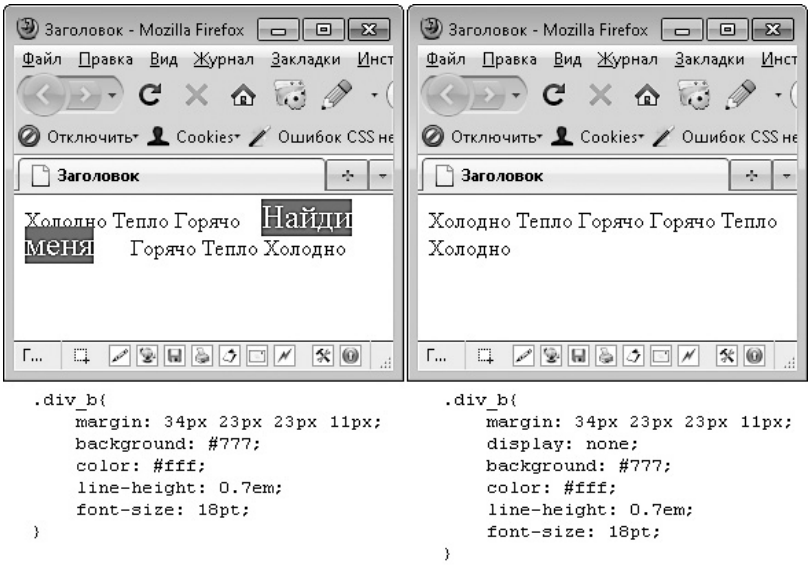


Рис. 2.16. Применение правила display: none;

run-in. В зависимости от контекста делает тег строчным или блочным. Вряд ли вы когда-нибудь будете применять данное свойство.

Это значение не поддерживается 6-й и 7-й версиями браузера Internet Explorer.

table. Преобразует тег в таблицу — в аналог тега `table`, то есть тегу присваиваются все особенности вывода тега `table`.

Данное значение не поддерживается 6-й и 7-й версиями браузера Internet Explorer.

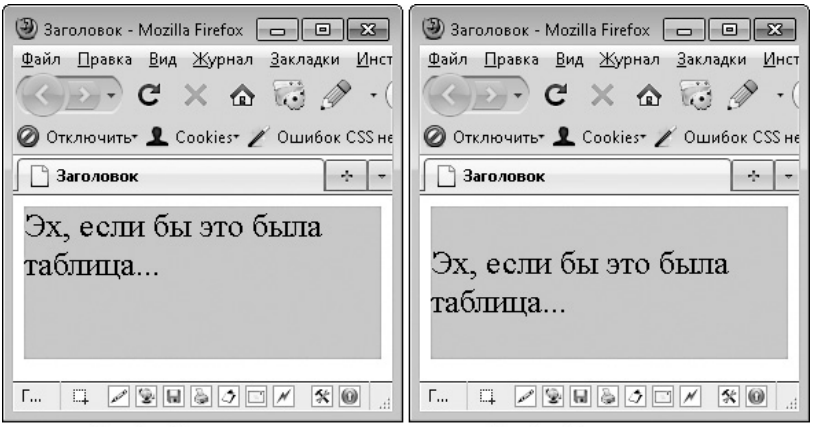
table-caption. Преобразует тег в заголовок таблицы — в аналог тега `caption`.

Данное значение не поддерживается 6-й и 7-й версиями браузера Internet Explorer.

table-cell. Преобразует тег в ячейку таблицы — в аналог тега `td` или `th` (рис. 2.17).

Данное значение не поддерживается 6-й и 7-й версиями браузера Internet Explorer.

table-footer-group. Преобразует тег в футер таблицы — в аналог тега `tfoot`.




```
.div_test{
    height: 111px;
    background: #ccc;
    font-size: 18pt;
    vertical-align: middle;
}
```

```
.div_test{
    height: 111px;
    background: #ccc;
    font-size: 18pt;
    vertical-align: middle;
    display: table-cell;
}
```

Рис. 2.17. Правило display: table-cell;

Данное значение не поддерживается 6-й и 7-й версиями браузера Internet Explorer.

table-header-group. Преобразует тег в шапку таблицы — в аналог тега `thead`.

Данное значение не поддерживается версиями 6 и 7 браузера Internet Explorer.

table-row-group. Преобразует тег в группу строк таблицы — в аналог тега `tbody`.

Данное значение не поддерживается версиями 6 и 7 браузера Internet Explorer.

table-row. Преобразует тег в строку таблицы — в аналог тега `tr`.

Данное значение не поддерживается версиями 6 и 7 браузера Internet Explorer.

table-column-group. Преобразует тег в группу колонок таблицы — в аналог тега `colgroup`.

Данное значение не поддерживается версиями 6 и 7 браузера Internet Explorer.

table-column. Преобразует тег в колонку таблицы — в аналог тега `col`.

Для тега, которому задано значение `table-column`, будет работать свойство `vertical-align`, то есть вы сможете выровнять содержимое такого тега по вертикали.

Если учесть, что использование таблиц для создания макета веб-страницы в последнее время считается плохим тоном, применение значения `table-column`

остается единственным способом выровнять содержимое тега по вертикали (конечно, если не учитывать JavaScript).

Данное значение не поддерживается 6-й и 7-й версиями браузера Internet Explorer.

visibility

Основные сведения:

- значение по умолчанию: `visible`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 2;
- синтаксис: `visibility: visible | hidden | collapse | inherit`.

Свойство `visibility` по своему действию напоминает значение `none` свойства `display`. Оно также позволяет при необходимости скрыть содержимое тега. Однако у свойства `visibility` есть одно существенное отличие — оно скрывает содержимое тега, но сам тег со страницы не удаляется, то есть место, которое занимает тег, по-прежнему остается зарезервированным за скрытым тегом.

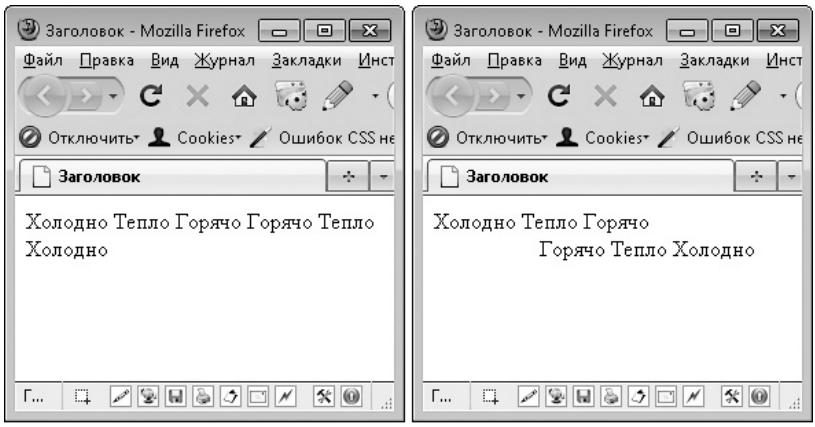
Можно сказать, что свойство `visibility` позволяет сделать тег полностью прозрачным.

Проще всего понять действие свойства `visibility` на примере (рис. 2.18).

Как видно на рис. 2.18, текст «Найди меня» успешно скрывается с помощью свойства `display`. Однако если использовать свойство `visibility`, найти данный текст не составляет большого труда, хоть сам текст и скрыт. Место, которое он занимал бы, по-прежнему выделяется тегу. И мы можем видеть, где находится скрытый текст, по пустому пространству между словами.

Итак, свойство `visibility` может принимать следующие значения:

- `visible` — отобразить содержимое тега;
- `hidden` — скрыть содержимое тега, а также фон и рамку вокруг него;
- `collapse` — для строк и колонок таблицы аналогично значению `none` свойства `display`, в остальных случаях аналогично значению `hidden` свойства `visibility`;
- `inherit` — наследовать значение от родительского тега.




```
.div_b{
    margin: 34px 23px 23px 11px;
    display: none;
    background: #777;
    color: #fff;
    line-height: 0.7em;
    font-size: 18pt;
}
```

```
.div_b{
    margin: 34px 23px 23px 11px;
    visibility: hidden;
    background: #777;
    color: #fff;
    line-height: 0.7em;
    font-size: 18pt;
}
```

```
Холодно Тепло Горячо <span class="div_b">Найди меня</span>
 Горячо Тепло Холодно
```

Рис. 2.18. Применение правила visibility: hidden;

Обратите внимание: браузер Internet Explorer не поддерживает значение `collapse` данного свойства. А версии 6 и 7 браузера Internet Explorer также не поддерживают значение `inherit`.

Изучаем свойства: границы

Любой тег состоит из внешнего (`margin`) и внутреннего (`padding`) отступов, а также границы.

border

Основные сведения:

- значение по умолчанию: зависит от использования;
- наследуется: нет;
- применяется: ко всем элементам;
- версия CSS: 1;

- синтаксис: `border: [border-width || border-style || border-color] | inherit`.

Граница тега находится между внешним и внутренним отступами. Она задается свойством `border`, в значении которого через пробел указывается толщина границы, ее тип и цвет: `border: толщина тип цвет`. Например, `border: 1px solid #fff;`.

Свойство `border` является универсальным. Оно заменяет собой сразу три отдельных свойства: `border-width`, `border-style` и `border-color`.

border-width. Позволяет задать толщину сразу для всех сторон границы либо отдельно для каждой из сторон.

- Если вы указываете одно значение для данного свойства, то указанная толщина применяется для всех сторон тега.
- Если вы указываете два значения через пробел, то первое значение устанавливает толщину верхней и нижней границ, а второе — левой и правой.
- Если вы указываете три значения через пробел, то первое значение задает толщину верхней границы, второе — толщину левой и правой границ, а третье — толщину нижней границы.
- При указании четырех значений через пробел они применяются отдельно для каждой из сторон: верхней, правой, нижней и левой, то есть по часовой стрелке.

Данный алгоритм задания значений для одной, двух, трех или четырех сторон действует и в других свойствах CSS, которые могут влиять как на весь тег, так и на отдельные его стороны. В том числе и в уже рассмотренных нами свойствах `padding` и `margin`.

Свойство `border-width` также является универсальным. Оно заменяет собой свойства `border-top-width`, `border-right-width`, `border-bottom-width` и `border-left-width`, позволяющие задать толщину конкретной границы тега.

В качестве значений перечисленных свойств могут выступать:

- число в пикселах;
- ключевое слово `thin`, равнозначное значению `2px`;
- ключевое слово `medium`, равнозначное значению `4px`;
- ключевое слово `thick`, равнозначное значению `6px`;
- ключевое слово `inherit` (версии 6 и 7 браузера Internet Explorer не понимают данное значение).

Обратите внимание, что граница не входит в состав ширины и высоты тега, то есть при определении реальной ширины и/или высоты тега вы должны учитывать и размер заданной ему границы. Фактически истинная ширина тега равна сумме следующих свойств: `margin-left`, `border-left-width`, `padding-left`, `width`, `padding-right`, `border-right-width`, `margin-right`. Это же касается и высоты тега. Не следует забывать о размере границы при точном позиционировании тегов на странице.

border-style. Данное универсальное свойство позволяет задать стиль отображения всей границы тега либо конкретно каждой его стороны. Оно заменяет собой свойства `border-top-style`, `border-right-style`, `border-bottom-style` и `border-left-style`, задающие стиль отдельно для каждой из сторон.

В качестве значений перечисленных свойств могут выступать следующие ключевые слова:

- `none` — скрыть границу;
- `inherit` — наследовать значение свойства родителя (версии 6 и 7 браузера Internet Explorer не понимают данное значение);
- `solid` — граница в виде сплошной линии;
- `dotted` — граница из точек (Internet Explorer 6 при толщине границы `1px` отображает `dotted` как `dashed`);
- `dashed` — граница из пунктирной линии;
- `double` — граница в виде двойной сплошной линии;
- `groove`, `ridge`, `inset`, `outset` — данные стили применяются реже.

Как выглядит граница при использовании рассмотренных выше значений, показано на рис. 2.19.

border-color. Дает возможность задать цвет границы либо отдельных ее сторон (через пробел). Данное универсальное свойство позволяет заменить свойства `border-top-color`, `border-right-color`, `border-bottom-color` и `border-left-color`.

Как задавать значения в виде цвета, было сказано в начале данного раздела. Помимо значений в виде цвета, в данных свойствах можно использовать следующие значения:

- `transparent` — прозрачный цвет границы;
- `inherit`.

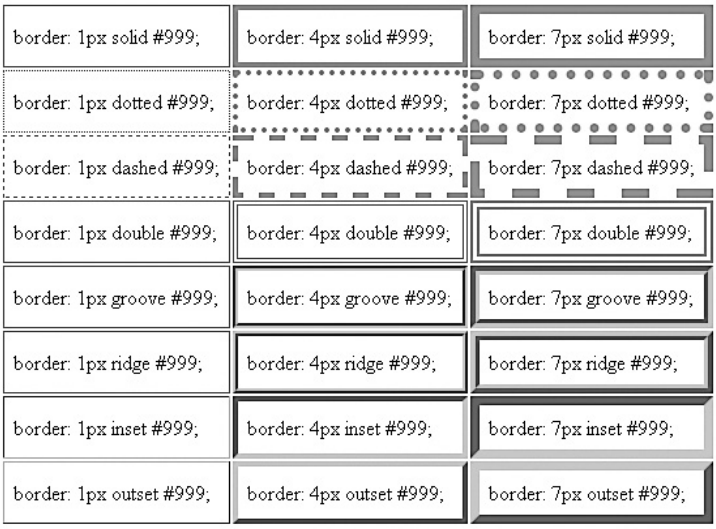


Рис. 2.19. Варианты стилей границы

outline

Основные сведения:

- значение по умолчанию: нет;
- наследуется: нет;
- применяется: ко всем элементам;
- версия CSS: 2;
- синтаксис: `outline: outline-color || outline-style || outline-width | inherit`.

Как и `border`, свойство `outline` позволяет задать границу вокруг тега. Синтаксис свойства `outline` подобен синтаксису свойства `border`: `outline: толщина тип цвет`.

Значения свойства outline

Свойство `outline` является универсальным. Оно заменяет собой три отдельных свойства: `outline-width`, `outline-style` и `outline-color`.

- `outline-width` — ширина границы в одном из следующих значений:
  - число в пикселах;
  - ключевое слово `thin`, равнозначное значению `1px`;

- ключевое слово `medium`, равнозначное значению `2px`;
- ключевое слово `thick`, равнозначное значению `3px`;
- ключевое слово `inherit`.

- `outline-style` — тип границы в виде одного из уже известных вам по свойству `border` ключевых слов: `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`, `inherit`.
- `outline-color` — цвет границы в одном из следующих значений:
  - значение, представляющее цвет;
  - ключевое слово `invert`, говорящее о том, что цвет границы должен быть противоположным цвету фона;
  - ключевое слово `inherit`.

Свойство `outline` отличается от `border` тем, что граница, заданная свойством `outline`, никак не влияет на размеры тега и его положение на странице. Чтобы понять это, посмотрите на рис. 2.20.

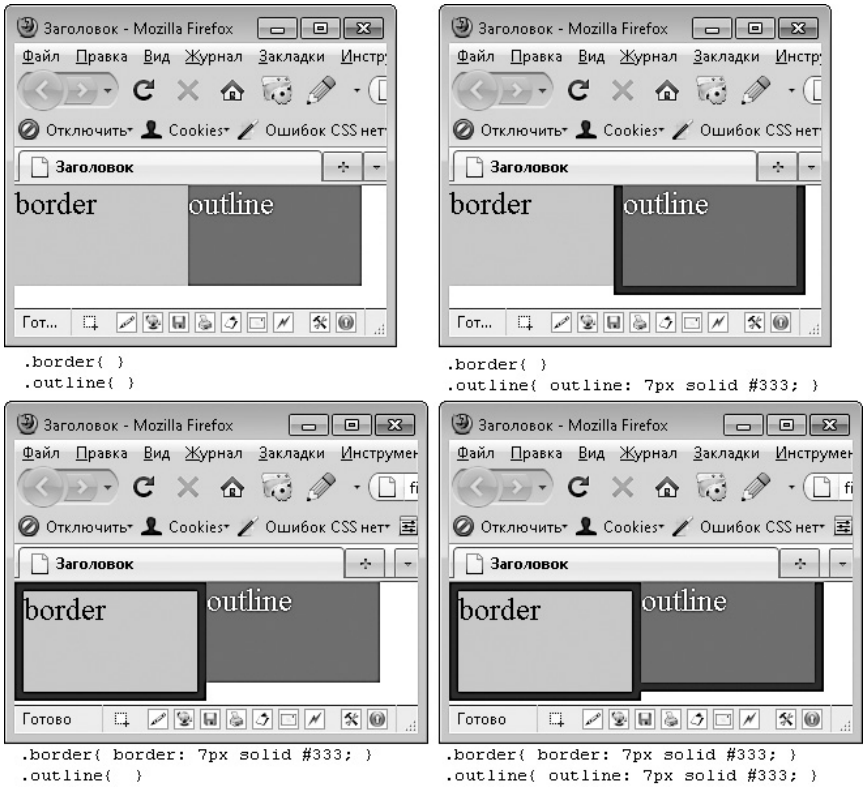


Рис. 2.20. Разница между свойствами border и outline

Верхний левый рисунок показывает теги без свойств `border` и `outline`. Оба тега имеют одинаковую ширину и высоту.

Верхний правый рисунок показывает эффект от применения свойства `outline`. В результате использования данного свойства:

- реальная ширина и высота тега не изменилась;
- тег находится на том же месте, на котором располагался без указания свойства `outline`;
- граница тега нарисована за пределами самого тега; причем граница может выходить за границу экрана или перекрываться рядом расположенными тегами.

Нижний левый рисунок показывает эффект от применения свойства `border`. В результате использования данного свойства:

- реальная ширина и высота тега увеличились на размер границы, из-за чего рядом расположенные теги сместились;
- тег смещен относительно своего прежнего положения на размер границы.

На нижнем правом рисунке одному тегу присвоено свойство `border`, а другому тегу — свойство `outline`. Граница, которая задана свойством `outline`, зашла на границу, определенную свойством `border`.

Обратите также внимание, универсальное свойство `outline` и все свойства, которые оно заменяет, не поддерживаются версиями 6 и 7 браузера Internet Explorer.

Обводка активной ссылки в Firefox

Свойство `outline` применяется очень редко, поэтому рассматривать его не имело бы никакого смысла, если бы не одно «но».

В браузере Mozilla Firefox обводка используется по умолчанию для выделения активных в данный момент ссылок, а также элементов страницы, на которых в данный момент установлен фокус (рис. 2.21).

Рис. 2.21. Обводка по умолчанию в браузере Mozilla Firefox

Иногда такая особенность существенно портит дизайн (особенно если ссылка или кнопка представляют собой картинку), и клиенты просят избавиться от обводки.

Сделать это несложно. Достаточно обнулить значение свойства `outline` для нужных тегов правилом `outline: none;`.

Закругление границ

В современном дизайне очень часто используются закругленные границы. Это красиво и на сегодняшний день достаточно модно.

Раньше создание закругленных границ не имело никакого отношения к самим границам. Все делалось набором картинок, в который входили отдельно картинки границы и отдельно картинки уголков для нее. После чего создавался «бутерброд» из тегов `div`, вложенных друг в друга. Каждому тегу в качестве фона присваивалась своя картинка. В итоге получалось изображение рамки с закругленными границами.

Далее в книге мы рассмотрим данный процесс подробнее. Но уже сейчас можно сказать, что это весьма трудоемкий процесс. И к тому же очень часто приводящий к различным «глюкам» вывода веб-страницы в старых браузерах.

ПРИМЕЧАНИЕ

Под «глюком» понимается эффект, при котором веб-страница при выводе в браузере иногда отличается от того, как она должна выглядеть. Часто «глюк» пропадает при прокрутке страницы. Если прокрутить веб-страницу вниз, а потом опять вернуться к месту, в котором были проблемы с выводом, то окажется, что теперь никаких проблем нет и все выводится так, как было задумано.

Все чаще закругление границ делают не отдельными картинками, а с помощью специальных свойств CSS. Но вот беда: мало того, что это нестандартизированные свойства и что называются эти свойства по-разному в различных браузерах, так еще и в браузере Internet Explorer свойства, закругляющие границу, появились только в 9-й версии.

Но лень, которая является частью человеческой природы, берет свое, и все чаще закругление границ делают свойствами CSS. Для этого указывают целый набор свойств:

```
-moz-border-radius: радиус; /* для Firefox */

-khtml-border-radius: радиус; /* для Konqueror */

-webkit-border-radius: радиус; /* для Chrome и Safari*/

border-radius: радиус; /* для Opera, iOS, Android */
```

Каждое свойство — для отдельного браузера. Хорошо хоть, что значения у этих свойств одинаковы. Но плохо, что браузер Internet Explorer до версии 9 ни одно из перечис-

ленных свойств не понимает. К этой проблеме верстальщики относились философски, радуясь и тому, что в браузерах, отличных от Internet Explorer, сайт выглядит красиво. А потом появилось решение, позволяющее легко закруглять углы рамки и в браузере Internet Explorer вплоть до 6-й версии. Данное решение работает совместно со свойствами `-moz-border-radius`, `-khtml-border-radius`, `-webkit-border-radius`, `border-radius` и позволяет научить браузер Internet Explorer понимать свойство `border-radius`. Рассмотрим его подробнее.

Подготавливаем основу. Сначала создадим блок, для которого в дальнейшем будем закруглять углы.

Для этого достаточно написать небольшой HTML-код:

```
<div class="divb">Данная колонка без закругленных рамок, но тоже красивая.</div>
```

После чего подключить к HTML-файлу CSS со следующим описанием селектора:

```
.divb{
    width: 133px;
    padding: 11px;
    margin: 11px;
    border: 1px solid #000;
}
```

В итоге получится обычная рамка с расположенным внутри нее текстом (рис. 2.22, *справа*), а нам нужна рамка с закругленными уголками, как на рис. 2.22, *слева*.

Закругляем рамку во всех браузерах, кроме Internet Explorer. Следующим этапом является добавление CSS-кода, закругляющего рамку для всех браузеров, кроме ранних версий Internet Explorer. Добавим CSS-код в селектор, созданный ранее:

```
.divb{
    width: 133px;
    padding: 11px;
    margin: 11px;
    border: 1px solid #000;

    -moz-border-radius: 7px;
```

```
    -khtml-border-radius: 7px;
    -webkit-border-radius: 7px;
    border-radius: 7px;
}
```

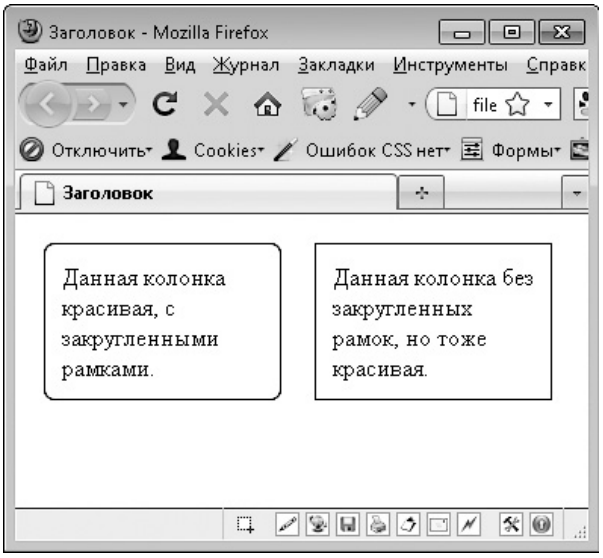


Рис. 2.22. Закругление границ

Мы добавили четыре свойства с одинаковым значением. Формат значения у всех этих свойств одинаков. В качестве значения может выступать:

- число в пикселах — определяет радиус скругления для всех углов рамки тега;
- через пробел четыре числа в пикселах — определяют радиус скругления отдельно для каждого угла рамки тега. Таким образом вы можете закруглить только углы на отдельной стороне рамки. Например: `-khtml-border-radius: 7px 0 0 7px;`.

В итоге наша рамка преобразилась (см. рис. 2.22, *слева*) для всех современных браузеров.

Закругляем рамку в браузерах Internet Explorer 6, 7, 8. Теперь перейдем к более сложным вещам. Для закругления рамки в прошлых версиях браузера Internet Explorer используется сторонний файл, который нужно разместить в одной из папок вашего сайта. Точнее, целый архив файлов, который можно скачать с сайта http://css3pie.com/.

ПРИМЕЧАНИЕ

Данный архив вы можете также скачать вместе с файлами для настоящей книги с сайта http://piter.com/. Архив располагается в папке tech/pie общего архива для данного издания.

Распакуйте скачанный архив в папку с CSS-файлами на вашем сайте. В результате в папку добавятся следующие файлы: `PIE.htc`, `PIE.js`, `PIE.php`, `PIE_uncompressed.htc`, `PIE_uncompressed.js`. Из них мы будем иметь дело только с файлами `PIE.htc` и `PIE.php`, однако другие файлы удалять нельзя.

Для скругления уголков лучше всего использовать файл `PIE.php`. Однако если вы создаете сайт не на хостинге (как мы сейчас), то PHP работать не будет. По этой причине вместо него придется использовать файл `PIE.htc`.

ПРИМЕЧАНИЕ

Если вы применяете файл PIE.htc на хостинге, проверьте, действительно ли рамки будут закруглены в браузере Internet Explorer прежних версий. Если закругления не произошло, попробуйте подключить файл PIE.php вместо PIE.htc.

После того как файлы размещены в папке сайта, остается только добавить дополнительные свойства CSS в наш селектор:

```
.divb{
    width: 133px;
    padding: 11px;
    margin: 11px;
    border: 1px solid #000;

    -moz-border-radius: 7px;
    -khtml-border-radius: 7px;
    -webkit-border-radius: 7px;
    border-radius: 7px;
    behavior: url("css/PIE.htc");
    position: relative;
    zoom: 1;
}
```

Мы добавили еще три свойства.

- `behavior: url("css/PIE.htc");` — задает путь к файлу `PIE.htc` или `PIE.php` обязательно относительно корня вашего сайта. Недопустимо в пути указывать двоеточие (`..`) и другие символы обратной навигации.
- `position: relative;` и `zoom: 1;` — данные свойства используются для того, чтобы перенести наш тег со «слоя» по умолчанию на отдельный «слой». Иначе в браузере Internet Explorer 6 могут возникать различные «глюки» в отображении тега, для которого применяется технология PIE. В принципе, вы можете указывать либо свойство `position`, либо свойство `zoom`. Сразу оба этих свойства задавать не обязательно.

Итак, мы изучили метод быстрого закругления рамок, работающий для всех браузеров. Как видите, для закругления нужно использовать сразу семь свойств CSS плюс отдельный файл `PIE.htc` или `PIE.php`.

Тени

Еще одной задачей, с которой часто сталкиваются верстальщики, является верстка макета с тенями вокруг определенного блока.

Решить данную задачу можно стандартным способом, то есть создать картинки с изображением тени, после чего добавить их точно так же, как и картинки закругленной рамки.

Но стандартный способ трудоемок. Куда проще добавить блоку тень с помощью уже известного нам файла `PIE.htc` (но лучше, конечно, `PIE.php`). Ведь он позволяет сделать и это.

Тень добавляется точно таким же способом, как и закругление рамки. Только вместо `-moz-border-radius`, `-khtml-border-radius`, `-webkit-border-radius`, `border-radius` используются другие свойства.

Итак, у нас есть HTML- и CSS-код, создающий блок с рамкой, который мы написали выше. Сначала добавим в него свойства для создания тени:

```
.divb{
    width: 133px;
    padding: 11px;
    margin: 11px;
    border: 1px solid #000;

    -moz-box-shadow: 0px 3px 7px #02041c;
```

```
    -webkit-box-shadow: 0px 3px 7px #02041c;
    box-shadow: 0px 3px 7px #02041c;
}
```

Мы добавили три CSS-свойства с одинаковым значением, каждое свойство для отдельного браузера. В значении этих свойств через пробел указываются следующие данные:

- сдвиг тени в пикселах по горизонтали;
- сдвиг тени в пикселах по вертикали;
- радиус размытия тени;
- цвет тени.

Таким образом, задаем четыре значения. Но возможен и другой синтаксис данных свойств, при котором для свойства через пробел указывается пять значений:

- сдвиг тени в пикселах по горизонтали;
- сдвиг тени в пикселах по вертикали;
- радиус размытия тени;
- положительное число, указывающее уровень растяжение тени, или отрицательное число, которое определяет уровень сужения тени;
- цвет тени.

Мы добавили тень для всех браузеров, кроме версий 6, 7, 8 браузера Internet Explorer. Чтобы добавить тень для старых версий браузера Internet Explorer, нужно поместить в папку на сайте файлы PIE, после чего добавить к селектору уже знакомые нам свойства:

```
.divb{
    width: 133px;
    padding: 11px;
    margin: 11px;
    border: 1px solid #000;

    -moz-box-shadow: 0px 3px 7px #02041c;
    -webkit-box-shadow: 0px 3px 7px #02041c;
    box-shadow: 0px 3px 7px #02041c;
    behavior: url("css/PIE.htc");
```

```
    position: relative;
    zoom: 1;
}
```

Назначение добавленных нами трех свойств полностью аналогично их назначению при закруглении рамки. В итоге у нас появилась тень вокруг рамки (рис. 2.23, *вверху слева*). Рамку можно также убрать (рис. 2.23, *вверху справа*) или закруглить (рис. 2.23, *внизу слева*). Закруглить можно и без рамки (рис. 2.23, *внизу справа*).

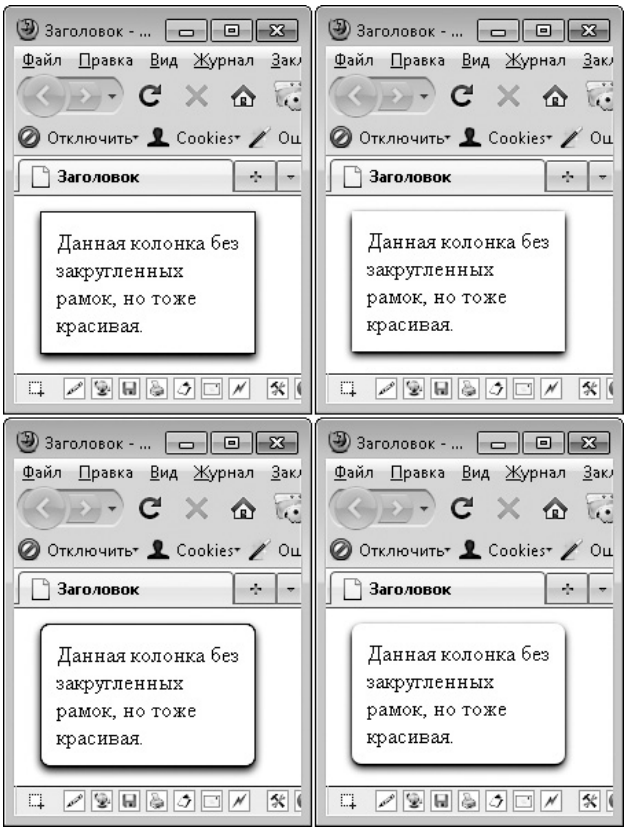


Рис. 2.23. Использование PIE

Таким образом, можно одновременно и добавлять тени, и закруглять рамку:

```
.divb{
    width: 133px;
    padding: 11px;
```

```
    margin: 11px;


    -moz-border-radius: 7px;
    -khtml-border-radius: 7px;
    -webkit-border-radius: 7px;
    border-radius: 7px;
    -moz-box-shadow: 0px 3px 7px #02041c;
    -webkit-box-shadow: 0px 3px 7px #02041c;
    box-shadow: 0px 3px 7px #02041c;
    behavior: url("css/PIE.htc");
    position: relative;
    zoom: 1;
}
```

Изучаем свойства: размер

Вы уже знаете из главы, посвященной HTML, что:

- ширина блочного тега равна ширине окна браузера;
- ширина строчного тега равна содержимому этого тега;
- высота тегов обоего типа определяется их содержимым.

Но это только по умолчанию. И с помощью CSS легко можно указать любую нужную ширину и высоту тега.

width

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к блочным элементам;

- версия CSS: 1;
- синтаксис: `width: значение | проценты | auto | inherit`.

Для указания ширины тега служит свойство `width`. В качестве его значения чаще всего выступает число в пикселах (`px`). Ширина также может задаваться в процентах (`%`) относительно родительского тега, дюймах (`in`), пунктах (`pt`).

Если вам необходимо отменить ширину, которая была ранее задана в одном из селекторов, то в качестве значения данного свойства следует использовать ключевое слово `auto`. В этом случае в качестве ширины будет применяться ширина по умолчанию для HTML-тега (для блочного — на всю ширину экрана).

Это может понадобиться, если в первом селекторе вы указывали ширину тега для всех страниц сайта, а во втором селекторе — ширину того же тега для конкретной страницы сайта:

```
#myblock{ width: 968px; }

body.contacts #myblock{ width: auto; }
```

Данное свойство можно применять только к блочным тегам. Если вам нужно задать ширину строчного тега, следует дополнительно воспользоваться свойством `display`, чтобы изменить тип строчного тега на `inline-block`. Только после этого вы сможете задавать ему ширину.

Как обычно, 6-я и 7-я версии браузера Internet Explorer не поддерживают значение `inherit` данного свойства.

height

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к блочным и заменяемым элементам;
- версия CSS: 1;
- синтаксис: `height: значение | проценты | auto | inherit`.

Чтобы указать высоту тега, достаточно воспользоваться свойством `height`. Значение данного свойства может указываться в пикселах (`px`), процентах (`%`) относительно родительского тега, дюймах (`in`), пунктах (`pt`). Можно также использовать ключевое слово `auto`, чтобы отменить высоту, заданную тегу ранее.

Данное свойство можно применять только к блочным тегам, а также к тегам, у которых для свойства `display` указано значение `inline-block`.

min-height и min-width

Основные сведения:

- значение по умолчанию: 0;
- наследуется: нет;
- применяется: ко всем элементам, кроме встроенных и таблиц;
- версия CSS: 2;
- синтаксис: `min-height: значение | проценты | inherit; min-width: значение | проценты | inherit`.

Свойства `height` и `width`, которые мы рассмотрели ранее, жестко задают высоту и ширину тега. По этой причине размер тега не может быть больше или меньше указанной для него ширины/высоты. Правда, если содержимое не помещается внутри тега, оно выйдет за его пределы. Но размер места, выделяемого тегу, от этого не изменится. И фон, и рамки, и другие элементы, которые назначены тегу, не растянутся и не сузятся (рис. 2.24).

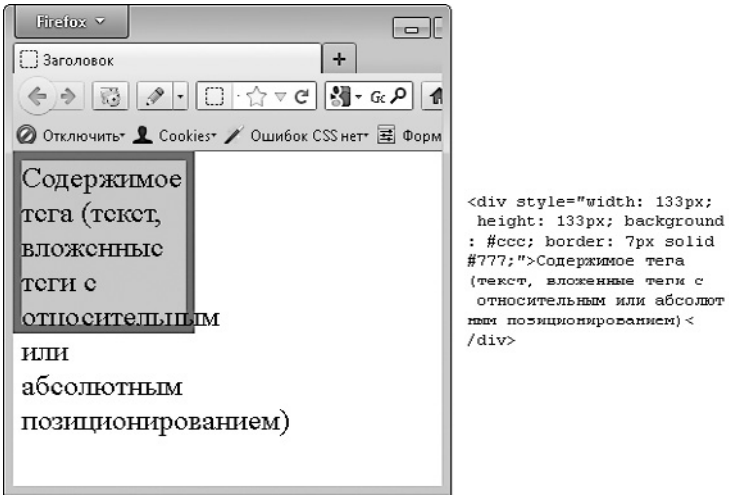


Рис. 2.24. Если текст не помещается внутри тега, он выйдет за его пределы

Иногда возникает необходимость задать тегу минимальную высоту или ширину, то есть чтобы размер тега мог быть больше заданного (если содержимое тега не помещается в заданных пределах). Но при этом чтобы тег не мог быть меньше заданного

размера, даже если тег вообще не имеет содержимого. Для этого и предназначены свойства `min-height` и `min-width`. Их синтаксис аналогичен синтаксису уже знакомых вам свойств `height` и `width`.

При использовании минимальной высоты и минимальной ширины возникает только одна проблема — данные свойства не поддерживаются браузером Internet Explorer 6. Но это не значит, что в упомянутом браузере нельзя задать минимальную ширину или высоту тега.

min-height для Internet Explorer 6. Internet Explorer 6 не поддерживает свойства `min-height`, однако само свойство `height` ведет себя точно так же, как `min-height` в более новых версиях браузера. Поэтому для браузера Internet Explorer 6 вместо `min-height` можно использовать свойство `height`:

```
#footer{
   min-height: 333px;
}
* html #footer{
   height: 333px;
}
```

min-width для Internet Explorer 6. Для задания минимальной ширины в Internet Explorer 6 чаще всего применяются правила expression. Они являются подобием языка JavaScript. Их можно использовать внутри CSS-файла для того, чтобы задавать различные условия, определяющие, какое именно значение будет назначено CSS-свойству. Однако expression поддерживаются только браузером Internet Explorer.

Чтобы определить минимальную ширину, применяется подобная запись:

```
#footer{
   min-width: 1002px;
  width:expression(document.body.clientWidth < 1002? "1002px":
"auto" );
}
```

В селекторе сначала указывается свойство `min-width`, после чего в том же селекторе с помощью expression определяется, чему равна ширина окна браузера пользователя. Если ширина меньше минимальной, то свойству `width` присваивается значение минимальной ширины. В противном случае свойству `width` присваивается значение по умолчанию.

При необходимости подобным выражением expression можно заменить и свойство `min-height`. Для этого достаточно заменить `document.body.clientWidth` на `document.body.clientHeight`. И, конечно, присваивать результат выполнения expression не свойству `width`, а свойству `height`.

max-height и max-width

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: нет;
- применяется: ко всем элементам, кроме встроенных и таблиц;
- версия CSS: 2;
- синтаксис: `max-height: значение | проценты | none | inherit; max-width: значение | проценты | none | inherit`.

В противоположность свойствам с префиксом `min-` свойства `max-height` и `max-width` позволяют указать максимальные размеры тега. Таким образом, тег может быть меньшего размера, если в нем недостаточно содержимого. А вот большего размера он быть не сможет.

Данные свойства не поддерживаются браузером Internet Explorer 6. По этой причине при их использовании следует дополнительно указывать expression, которые бы заменяли свойства для браузера Internet Explorer 6.

Так, для замены свойства `max-width` используется подобное выражение:

```
width: expression( document.body.clientWidth > 776 ? "777px" :
"auto" );
```

А для замены свойства `max-height` такое:

```
height: expression( this.scrollHeight > 332 ? "333px" :
"auto" );
```

overflow

Основные сведения:

- значение по умолчанию: `visible`;
- наследуется: нет;

- применяется: к блочным элементам;
- версия CSS: 2;
- синтаксис: `overflow: auto | hidden | scroll | visible | inherit;`.

Перед тем как рассматривать назначение свойства `overflow`, еще раз обратим внимание на одну особенность использования свойств `width` и `height`.

Содержимое тега (текст, вложенные теги с относительным или абсолютным позиционированием) может выходить за его пределы даже в том случае, если тегу задана конкретная ширина и/или высота. Достаточно тексту превысить размеры тега, и текст выйдет за его пределы.

А все потому, что свойству `overflow` по умолчанию присвоено значение `visible`. Ведь именно свойство `overflow` определяет, что будет происходить, если текст и другое содержимое тега выйдут за его пределы.

Свойство `overflow` может иметь следующие значения:

- `visible` — содержимое тега выходит за его пределы и отображается в браузере (рис. 2.25, *вверху слева*);
- `hidden` — все содержимое, которое выходит за пределы тега, скрывается (рис. 2.25, *вверху справа*);
- `scroll` — независимо от того, выходит ли содержимое за пределы тега, тегу добавляются полосы прокрутки (рис. 2.25, *внизу слева*);
- `auto` — если содержимое выходит за пределы тега по горизонтали и/или вертикали, тегу добавляется горизонтальная и/или вертикальная полоса прокрутки, позволяющая увидеть ту часть содержимого, которая не поместилась в отведенных размерах (рис. 2.25, *внизу справа*).

Обратите внимание: свойство `overflow` применяется только совместно со свойствами `height` и/или `width`:

- если тегу задано только свойство `width`, то `overflow` будет действовать исключительно по горизонтали (если содержимое выходит за пределы тега по вертикали, оно будет отображаться независимо от значения свойства `overflow`);
- если тегу задано только свойство `height`, то `overflow` будет действовать исключительно по вертикали (если содержимое выходит за пределы тега по горизонтали, оно будет отображаться независимо от значения свойства `overflow`);
- если тегу заданы и свойство `width`, и свойство `height`, то `overflow` будет действовать во всех направлениях.

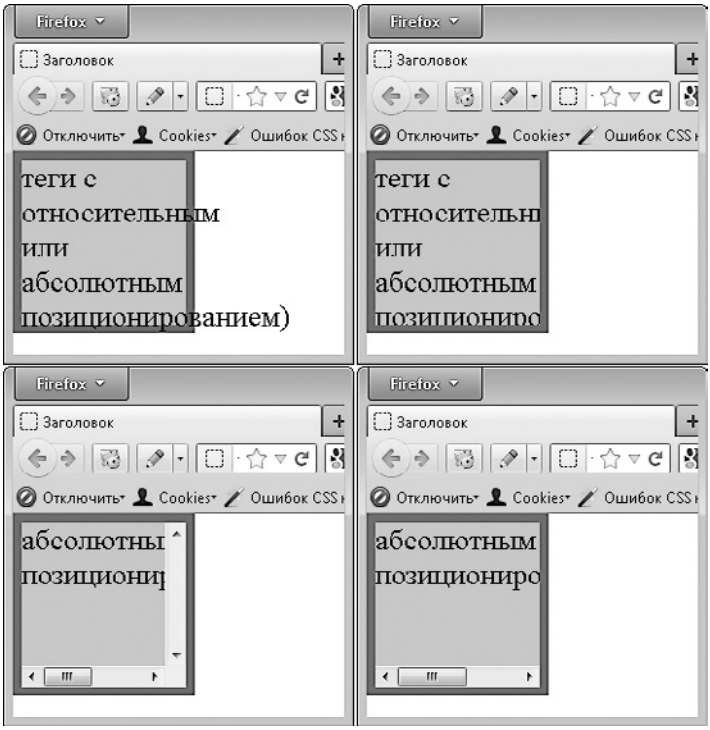


Рис. 2.25. Значения свойства overflow

Такое совместное использование свойств вполне логично. Иначе какой смысл задавать правила отображения выходящего за пределы тега содержимого, если сами пределы не указаны?

clip

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к блочным элементам;
- версия CSS: 2;
- синтаксис: `clip: rect(Y1, X1, Y2, X2) | auto | inherit`.

Свойство `clip` по своему назначению несколько напоминает свойство `overflow`. Оно позволяет указать область внутри тега, которая будет отображаться. Все, что не входит в указанную область, будет скрыто.

Во всех браузерах, кроме 6-й и 7-й версий Internet Explorer, свойство `clip` имеет следующий синтаксис: `rect(Y1, X1, Y2, X2)` (рис. 2.26, *справа*), где:

- `Y1` — расстояние от верха тега, на котором начинается видимая область по вертикали;
- `X1` — расстояние от левой стороны тега, на котором заканчивается видимая область по горизонтали;
- `Y2` — расстояние до низа тега, на котором заканчивается видимая область по вертикали;
- `X2` — расстояние от левой стороны тега, на котором начинается видимая область по горизонтали.

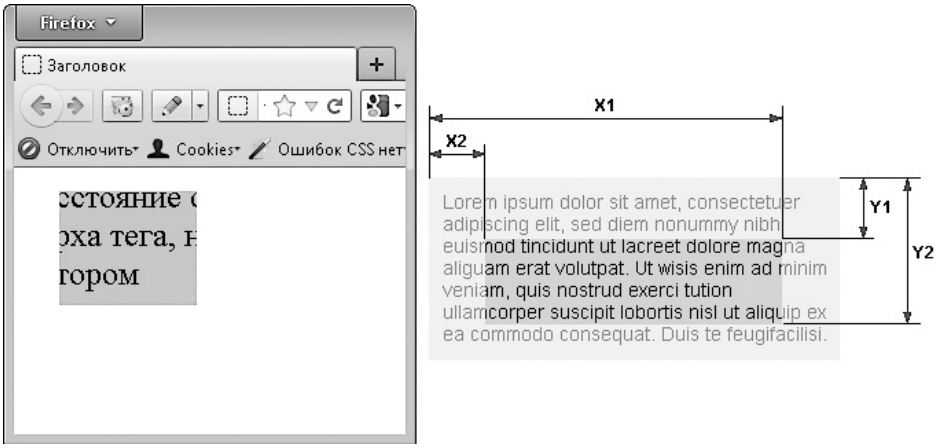


Рис. 2.26. Результат применения свойства clip

В браузерах Internet Explorer 6 и 7 синтаксис свойства `clip` несколько отличается. А именно, для разделения значений вместо запятых должны использоваться пробелы: `rect(Y1 X1 Y2 X2)`.

Обратите внимание: свойство `clip` работает только в том случае, если тегу задано абсолютное позиционирование, то есть, помимо свойства `clip`, для тега определено свойство `position` со значением `absolute`.

Изучаем свойства: позиционирование

Известная поговорка гласит, что написанное пером не вырубишь топором. Замечание, в общем-то, верное, но только не для CSS. Мы научились уже удалять теги с помощью CSS. Теперь рассмотрим, как изменять расположение тегов на веб-странице. Для этого предназначены свойства, влияющие на позиционирование элементов.

text-align

Основные сведения:

- значение по умолчанию: `left`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис:
  - CSS2.1: `text-align: center | justify | left | right | inherit`;
  - CSS3: `text-align: center | justify | left | right | start | end`.

На самом деле свойство `text-align` предназначено для выравнивания текста, написанного в теге. И выравнивание самих тегов является лишь побочным умением данного свойства. Однако до раздела, посвященного работе с текстом (см. раздел «Изучаем свойства: текст» данной главы), еще далеко. А свойство `text-align` очень скоро нам понадобится именно для позиционирования элементов.

Итак, свойство `text-align` позволяет выровнять текст, картинку или любой тег, за исключением `table`, одним из следующих способов (рис. 2.27):

- `left` — по левому краю;
- `right` — по правому краю;
- `center` — по центру;
- `justify` — по ширине (то есть одновременно выровнять и по левому и по правому краю).

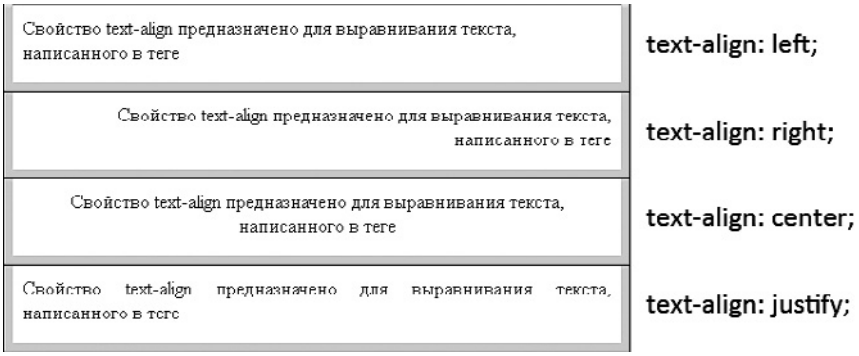


Рис. 2.27. Варианты свойства text-align

Выравнивается все, что находится внутри тега, которому присваивается свойство `text-align`.

Центрирование блока

Думаю, вы уже не раз встречали сайты, на которых все содержимое находится по центру страницы. И блок с содержимым, расположенный по центру, имеет фиксированную ширину (рис. 2.28).

Рис. 2.28. Пример центрированного блока с содержимым сайта

Несмотря на кажущуюся простоту, у начинающих верстальщиков очень часто возникают трудности с таким типом верстки. А все потому, что они не знают одной особенности центрирования, о которой мы сейчас поговорим.

Итак, давайте подумаем, как же можно центрировать блок? Конечно, самый простой способ — установить для блока одинаковый `padding` слева и справа. В этом случае блок размещается по центру. Но вот беда, ширина блока зависит от ширины экрана. А если задать еще свойство `min-width`, то при нехватке ширины экрана будет появляться горизонтальная прокрутка.

Как же центрировать блок с фиксированной шириной? Ну конечно, для этого применяется свойство `text-align` со значением `center`. Однако, если вы попробуете его применить, ничего не произойдет.

Проблема заключается в том, что свойство `text-align` не будет центрировать блочные теги до тех пор, пока блочным тегам не сбросят левый и правый `margin`, то есть для левого и правого `margin` не будет указано значение `auto`.

Полный CSS-код, позволяющий центрировать блок, выглядит следующим образом:

```
body{
   text-align: center;
}
#block_center{
   width: 900px;
   margin: 0 auto;
   text-align: left;
}
```

Обратите внимание, поскольку свойство `text-align` наследуется, центрированному блоку следует задать выравнивание по левому краю. Конечно, если вы не хотите, чтобы текст в нем также располагался по центру.

Общепринято, что реальная ширина центрированного блока не должна превышать 1024 пикселов. Дело в том, что верстальщики по-прежнему ориентируются на мониторы шириной 1024 пиксела. И если ширина отцентрированного блока будет превышать 1024 пиксела, на таких мониторах будет появляться горизонтальная прокрутка. А этого лучше избегать.

И если вам кажется, что такой древности, как мониторы шириной 1024 пиксела, на свете давно уже не существует, то вспомните о популярных нетбуках. На большинстве из них ширина экрана как раз составляет 1024 пиксела.

position

Основные сведения:

- значение по умолчанию: `static`;
- наследуется: нет;

- применяется: ко всем элементам, за исключением генерируемых;
- версия CSS: 2;
- синтаксис: `position: absolute | fixed | relative | static | inherit`.

Свойство `text-align` лишь косвенно относится к позиционированию элементов. Когда говорят о позиционировании, подразумевают свойство `position`.

С помощью свойства `position` можно сдвинуть любой тег относительно его текущего местоположения либо относительно родительского тега. Правда, размер сдвига задается свойствами `left`, `top`, `bottom` и `right`, но это уже мелочи.

По умолчанию любой тег находится на том месте, где вы его указали в HTML. Это эквивалентно значению `static` свойства `position`. Другие значения данного свойства:

- `absolute` — назначает тегу абсолютное позиционирование, при котором тег позиционируется относительно экрана либо относительно первого родительского тега, для которого также задано позиционирование;
- `relative` — назначает тегу относительное позиционирование, при котором тег позиционируется относительно его текущего расположения;
- `fixed` — назначает тегу фиксированное позиционирование, при котором тег зафиксирован в определенном месте окна браузера и не покидает своего положения при прокрутке (не поддерживается браузером Internet Explorer 6).

Как только вы укажете свойству `position` любое значение за исключением `static` и `inherit`, тег из двухмерного пространства переносится в трехмерное. До этого момента все содержимое веб-страницы находилось в двухмерном пространстве. Но после применения свойства `position` пространство становится трехмерным. А сам тег, для которого было применено данное свойство, помещается на отдельный слой этого пространства. Причем данный слой имеет определенную глубину, то есть может располагаться ниже или выше других слоев.

Чтобы лучше понять идеи позиционирования, нам понадобятся свойства `left`, `top`, `bottom` и `right`.

left, right, top, bottom

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;

- применяется: ко всем элементам;
- версия CSS: 2;
- синтаксис:
  - `left: значение | проценты | auto | inherit;;`
  - `right: значение | проценты | auto | inherit;;`
  - `top: значение | проценты | auto | inherit;;`
  - `bottom: значение | проценты | auto | inherit;.`

Как только тег выходит за пределы двухмерного пространства, на него начинают действовать свойства:

- `left` — смещение слева либо расположение тега относительно левой границы;
- `top` — смещение сверху либо расположение тега относительно верхней границы;
- `bottom` — смещение снизу либо расположение тега относительно нижней границы;
- `right` — смещение справа либо расположение тега относительно правой границы.

Относительное позиционирование

При относительном позиционировании расположение тега задается относительно его текущего местоположения. При этом в качестве значения свойств `left`, `top`, `bottom` и `right` выступает смещение:

```
<p>Что написано пером, того <span style="position: relative;
top: +11px;">не</span> вырубишь топором</p>
```

На рис. 2.29 показаны результаты относительного позиционирования.

Рис. 2.29. Использование position: relative;

Например:

- `left: +7px` — сместит тег на 7 пикселов вправо;
- `left: -3px` — сместит тег на 3 пиксела влево;
- `top: +7px` — сместит тег на 7 пикселов вниз.

Абсолютное позиционирование

При абсолютном позиционировании тег располагается относительно слоя, на котором он находится. В качестве слоя может выступать:

- ближайший родительский элемент, для которого задано свойство `position` со значениями `relative`, `absolute` или `fixed`;
- если такого родительского элемента нет, то экран браузера (тег `body`).

Само же расположение тега определяется свойствами `top`, `left`, `right`, `bottom`. Как правило, задаются два из перечисленных свойств: `left` или `right` плюс `top` или `bottom`. Например:

- `left: 0; top: 0;` — тег расположен в левом верхнем углу слоя;
- `right: 7px; top: 0;` — тег размещен вверху слоя, на расстоянии 7 пикселов от правой границы слоя;
- `left: -11px; bottom: 7px;` — тег расположен на 11 пикселов левее левой границы слоя и на расстоянии 7 пикселов от нижней границы слоя.

В браузере Internet Explorer 6 для абсолютно позиционированных элементов нельзя одновременно задавать свойства `top` и `bottom` или `left` и `right`.

Рассмотрим HTML-пример:

```
<div class="thisparent">Что написано пером, того <div class=
"thischild">не</div> вырубишь топором</div>
```

Для удобства зададим классу `thisparent` следующие CSS-свойства:

```
.thisparent{

    background: #ccc;

    margin: 33px;

    padding: 11px;

}
```

На рис. 2.30, *слева* показано, что у нас получилось.

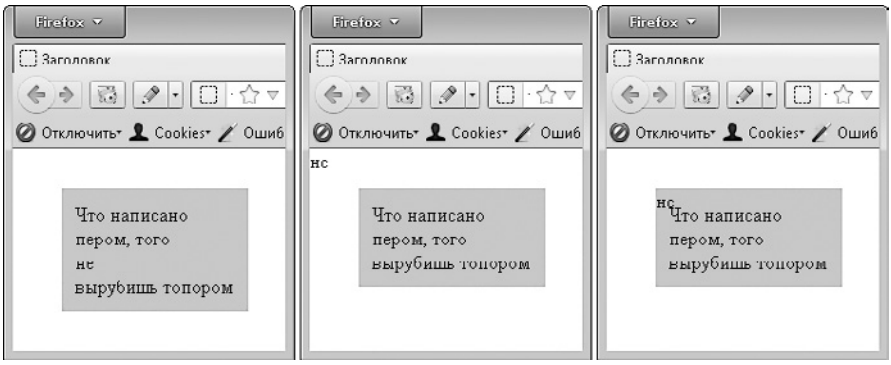


Рис. 2.30. Начинаем позиционировать абсолютно...

Теперь зададим классу `thischild` абсолютное позиционирование:

```
.thischild{
    position: absolute;
    left: 0;
    top: 0;
}
```

Как видите (рис. 2.30, *в центре*), тег с классом `thischild` стал позиционироваться относительно экрана браузера, что не очень подходит для нашего примера. Сделаем так, чтобы тег с классом `thischild` позиционировался относительно родительского элемента (рис. 2.30, *справа*). Для этого достаточно классу `thisparent` добавить следующие свойства:

```
.thisparent{
    position: relative;
    left: +0px;
}
```

Свойство `left` необходимо указывать для того, чтобы тег окончательно переместился на новый слой.

А теперь можно поэкспериментировать со свойствами `top`, `left`, `right`, `bottom` (рис. 2.31).

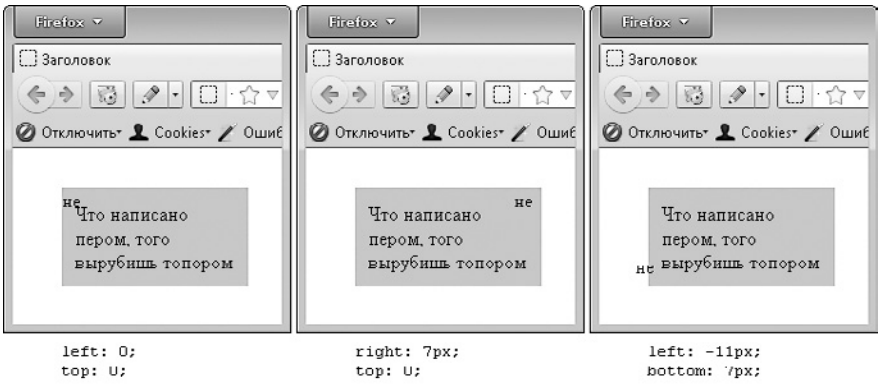


Рис. 2.31. Использование position: absolute;

Фиксированное позиционирование

При фиксированном позиционировании элемент располагается на одном месте в пределах окна браузера и не смещается ни при каких обстоятельствах (при прокрутке страницы).

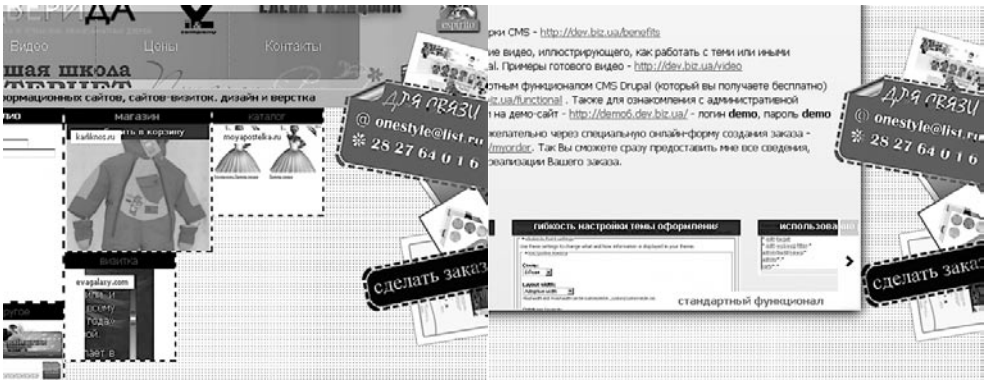
Рассмотрим пример:

```
<div class="stikers">
   <a href="/contacts">
         <img  src="/sites/all/themes/mlm/images/stikers.png"
alt="Контакты" />
   </a>
</div>
```

И CSS:

```
.stikers{
   position: fixed;
   top: 91px;
   right: 0;
}
```

Как видно из рис. 2.32, независимо от того, какая часть страницы видна сейчас, блок «Для связи» находится в одном и том же месте.



а *б*

Рис. 2.32. Фиксированное позиционирование

У фиксированного позиционирования есть недостаток — браузер Internet Explorer 6 не поддерживает значения `fixed`. К счастью, решить проблему можно с помощью expression. Достаточно в CSS добавить дополнительный селектор для Internet Explorer 6:

```
* html .stikers{

   position: absolute;

   top: expression(document.getElementsByTagName( 'body' )[0].
scrollTop + "px");

}
```

Однако работать данный способ будет лишь в том случае, если в HTML-документе не задан `DOCTYPE`.

Кроме того, еще один недостаток данного хака заключается в дрожании картинки в Internet Explorer 6 при прокрутке. Чтобы решить эту проблему, нужно дополнительно добавить следующий CSS-код:

```
body {

  background: url('fakeimg.gif') no-repeat;

  background-attachment: fixed;

}
```

z-index

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к любым позиционированным элементам;
- версия CSS: 2;
- синтаксис: `z-index: число | auto | inherit`.

Мы уже знаем, что содержимое веб-страницы может находиться не только в двухмерном пространстве, словно обычный лист бумаги. При использовании свойства `position` пространство становится трехмерным, а значит, появляется глубина. При этом становится возможным располагать теги друг над другом. Для этого достаточно указать тегам разную глубину и разместить их с помощью свойств `left`, `right`, `bottom`, `top` так, чтобы они пересекались.

Свойство `z-index` как раз и предназначено для задания глубины тега. Чем больше значение, которое вы зададите в данном свойстве, тем выше будет тег относительно других тегов. Таким образом, поверх других тегов будет отображаться тот из них, для которого задано наибольшее значение `z-index`.

Рассмотрим простой пример:

```
<div class="tagA">тег A</div>

<div class="tagB">тег B</div>
```

И CSS:

```
.tagB{

    text-align: center;

    background: #ccc;

    padding: 11px;

    width: 133px;

    height: 133px;

    position: absolute;

    left: 11px;
```

```
    top: 11px;


    z-index: 7;
}
.tagA{
    text-align: center;
    background: #777;
    padding: 11px;
    width: 133px;
    height: 133px;
    position: absolute;
    left: 23px;
    top: 53px;


    z-index: 11;
}
```

В итоге блоки «тег А» и «тег Б» располагаются друг над другом (рис. 2.33). Причем блок «тег А» находится над блоком «тег Б», несмотря на то что в HTML блок «тег Б» указан позже блока «тег А».

zoom

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: нет;
- применяется: ко всем элементам;
- версия CSS: не стандартизирован;
- синтаксис: `zoom: <значение> | <проценты> | normal`.

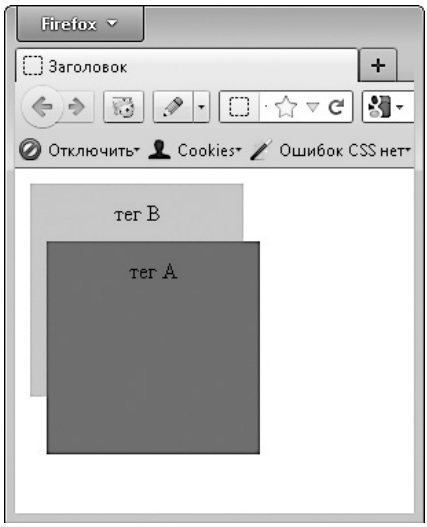


Рис. 2.33. Использование глубины

Официально свойства `zoom` не существует. Впервые оно появилось в браузере Internet Explorer 5.5 как расширение, после чего стало поддерживаться браузерами Chrome и Safari.

Основное назначение данного свойства — масштабирование элемента веб-страницы. С его помощью вы можете увеличить элемент либо уменьшить его.

Одним из побочных действий данного свойства является то, что тег, для которого применяется свойство `zoom`, помещается на отдельный слой. Именно по этой причине свойство `zoom` применяется в некоторых хаках.

А поскольку вы можете столкнуться со свойством `zoom`, изучая различные хаки, оно приведено в данной книге.

float

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: нет;
- применяется: ко всем элементам (за исключением позиционированных);
- версия CSS: 1;
- синтаксис: `float: left | right | none | inherit`.

Свойство `float` предназначено для создания плавающих тегов. Содержимое плавающего тега будет обтекаться другими тегами.

К значениям данного свойства относятся следующие:

- `left` — создает плавающий тег, выровненный по левому краю (все другие элементы на странице будут обтекать данный тег с правой стороны);
- `right` — создает плавающий тег, выровненный по правому краю (все другие элементы на странице будут обтекать данный тег с левой стороны).

В качестве примера рассмотрим следующий HTML-код:

```
<div class="tagF">На этом мы заканчиваем изучение свойства position...</div>

Свойство float предназначено для создания плавающих тегов. Содержимое плавающего тега будет «обтекаться» другими тегами. Но без наглядного примера понять все это довольно сложно.
```

И CSS-код:

```
.tagF{
    background: #ccc;
    margin-right: 11px;
    width: 133px;
    float: left;
}
```

Результат виден на рис. 2.34, *вверху*. Если же вместо `float: left;` использовать правило `float:right;`, то блок будет пристыкован к правому краю страницы (рис. 2.34, *внизу*).

Если внимательно посмотреть на CSS-код примера, можно заметить свойство `width`, задающее ширину плавающего тега. Для плавающего тега всегда нужно задавать ширину. Причем ширину в процентах при этом использовать нельзя.

clear

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: нет;

- применяется: к блочным и плавающим элементам;
- версия CSS: 1;
- синтаксис: `clear: none | left | right | both | inherit`.

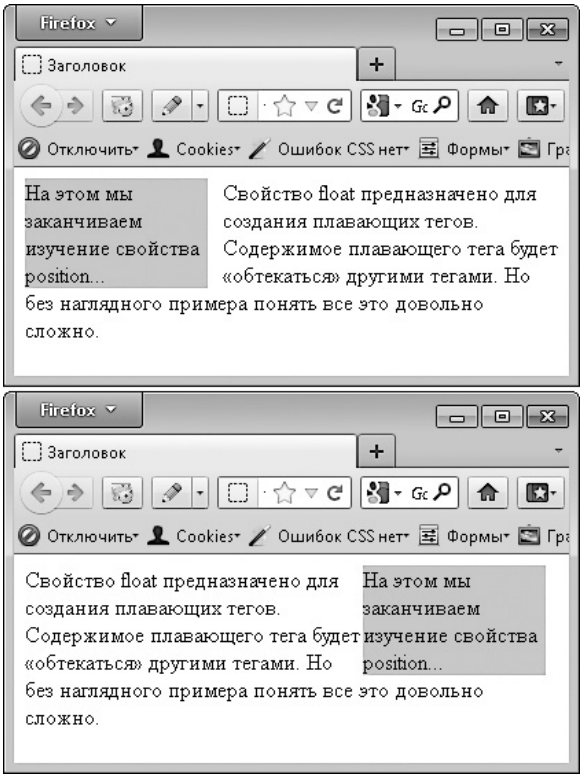


Рис. 2.34. Использование свойства float

Плавающий элемент обтекают все теги, которые в HTML расположены ниже плавающего тега. Однако на практике довольно часто возникают ситуации, когда обтекать плавающий элемент должны только несколько расположенных после плавающего элемента тегов.

Чтобы лучше это понять, рассмотрим простой пример:

```
<div>

   <div class="tagF">На этом мы заканчиваем изучение свойства
position...</div>

   Свойство float предназначено для создания плавающих тегов.
```

```
</div>

<div>

   <div class="tagF">На этом мы заканчиваем изучение свойства
position...</div>

   Свойство float предназначено для создания плавающих тегов.

</div>
```

И знакомый нам CSS:

```
.tagF{

    background: #ccc;

    margin: 0 11px 11px 11px;

    width: 133px;

    float: left;

}
```

В примере показан список из двух элементов, в каждом из которых есть плавающий элемент. Казалось бы, все должно быть нормально: сначала идет один элемент, а после него второй. Но на практике содержимое страницы превращается в кашу (рис. 2.35).

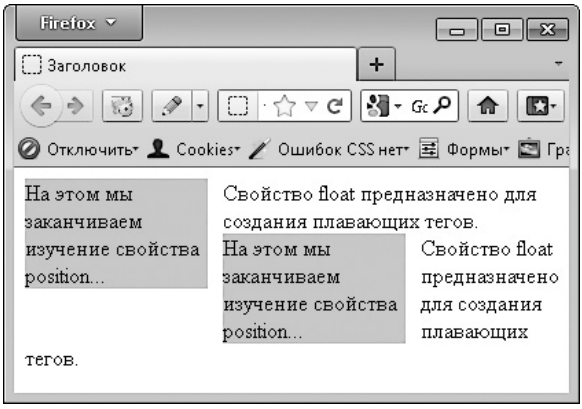


Рис. 2.35. Наслоение плавающих элементов

Исправить эту кашу поможет свойство `clear`. Оно позволяет отменить действие всех плавающих элементов, которые размещены выше тега со свойством `clear`, на теги, расположенные ниже тега со свойством `clear`.

Можно сказать, что оно проводит некую незримую границу. До этой границы плавающие теги работают. А после нее те же плавающие теги уже не оказывают никакого влияния. Хотя новые плавающие теги, написанные после тега со свойством `clear`, также будут работать.

В качестве значения свойства `clear` могут выступать следующие ключевые слова:

- `left` — отменить обтекание слева (действие правила `float: left;`);
- `right` — отменить обтекание справа;
- `both` — отменить любое обтекание.

Подправим наш пример, чтобы страница отображалась так, как нужно (рис. 2.36):

```
<div style="clear: both;">

<div class="tagF">На этом мы заканчиваем изучение свойства position...</div>

Свойство float предназначено для создания плавающих тегов.

</div>

<div style="clear: both;">

<div class="tagF">На этом мы заканчиваем изучение свойства position...</div>

Свойство float предназначено для создания плавающих тегов.

</div>
```

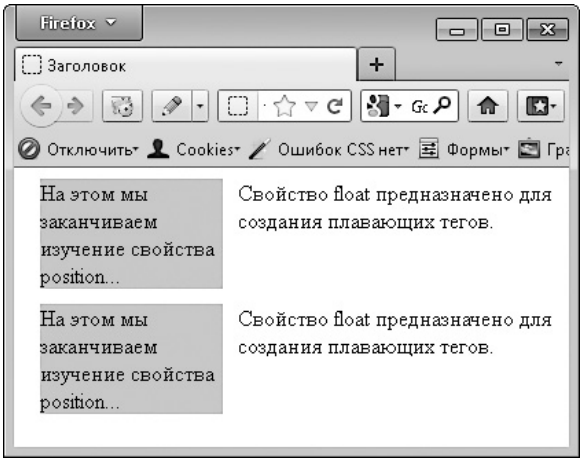


Рис. 2.36. Использование свойства clear

Удобнее всего создать специальный класс для применения свойства `clear`:

```
div.clear{
    clear: both;
    height: 0px;
    overflow: hidden;
}
```

После чего применять его в HTML-документе при необходимости:

```
<div class="clear"> </div>
```

Неразрывный пробел внутри пустого тега нужен для того, чтобы не возникало ошибок с правильным определением размеров пустого тега.

Верстаем макет страницы

Наших теоретических знаний уже достаточно для того, чтобы начать верстать дивами. Поэтому пришла пора на практике научиться верстать двух- и трехколоночные макеты.

Учиться этому мы будем на примере PSD-макета, который мы закончили в главе, посвященной HTML.

Центрирование блока

Итак, если взглянуть на наш PSD-макет снова (см. рис. 1.22), можно заметить, что все содержимое на нем располагается по центру экрана.

Как этого добиться, вы уже знаете, достаточно добавить следующий CSS-код:

```
#block_center{
    text-align: left;
    width: 1024px;
    margin: 0 auto;
}
```

Хорошо, что мы заранее поместили все содержимое веб-страницы в тег `div`. Теперь мы можем присвоить этому тегу идентификатор `block_center`, и центрирование будет готово (рис. 2.37). Правило `text-align: center;` тегу `body` задавать не обязательно.

Рис. 2.37. Центрирование блока

Создание двух колонок, вариант 1

Еще одной задачей, которую часто приходится решать верстальщику, является создание нескольких колонок без использования таблиц. За примером далеко ходить не нужно. Даже в нашем макете есть две колонки: левая колонка с каталогом товаров и правая колонка с контентом сайта.

Каким же образом можно создать две колонки без применения таблиц?

Можно воспользоваться абсолютным позиционированием, разместив рядом друг с другом два отдельных `div`. Однако впоследствии возникнет большая проблема — как расположить футер так, чтобы на него не наезжала ни одна из созданных колонок. Ведь, поскольку колонки позиционируются абсолютно, их формально не будет существовать в HTML-макете (под них не будет отводиться место) и футер будет размещен под колонками.

Первый придуманный нами вариант не подходит. Остается второй вариант — воспользоваться плавающими элементами.

Итак, присвоим тегу `div`, в который взято все содержимое левой колонки нашей верстки, идентификатор `left_sidebar`. После чего добавим следующий CSS:

```
#left_sidebar{
    width: 182px; /* задаем ширину левой колонки */
    float: left; /* плавающая колонка с левого края */
    margin: 0 11px; /* отступ слева и справа для красоты */
}
```

И вуаля — справа от колонки появилось содержимое (рис. 2.38).

Рис. 2.38. Создаем левую колонку

Но возникла проблема. Мы забыли, что все содержимое страницы обтекает плавающий элемент не только справа (или слева), но и снизу. А в нашем случае если правая колонка занимает по высоте большее пространство, то под левой колонкой должно оставаться пустое место.

Впрочем, возникшую проблему легко решить. Для этого достаточно присвоить идентификатор, допустим `main_content`, тегу `div`, в котором находится все содержимое правой колонки. После чего задать правой колонке левый отступ, равный по ширине реальному размеру левой колонки (рис. 2.39):

```
#main_content{
    margin-left: 204px; /* задаем отступ слева для колонки */
}
```

Рис. 2.39. Двухколоночная верстка

И, конечно, не стоит забывать о ситуации, когда левая колонка по высоте больше правой. В таком случае нужно отменить плавающий элемент для футера и всего, что находится ниже левой и правой колонок.

В HTML перед закрывающим тегом `div`, в котором у нас находятся левая и правая колонки, нужно поставить конструкцию `<div class="clear"> </div>`. Кроме того, необходимо добавить в CSS определение селектора `clear`:

```
div.clear{
   clear: both;
   height: 0px;
   overflow: hidden;
}
```

Создание двух колонок, вариант 2

Представленный выше вариант двухколоночной верстки хорош, но только до того момента, как вы начнете применять внутри правой колонки свойство `clear`. Как только вы это сделаете, то сразу обнаружите основной недостаток данной верстки: все, что находится ниже тега со свойством `clear`, смещается вниз, под левую колонку. По этой причине на практике используется другой, доработанный вариант.

Данный вариант довольно сложен в понимании. Он работает, это факт. Но почему он работает, для большинства верстальщиков остается загадкой.

Сначала нужно доработать CSS-код левой колонки, необходимо разместить левую колонку на отдельном слое, то есть задать позиционирование:

```
#left_sidebar{
    width: 182px; /* задаем ширину левой колонки */
    float: left; /* плавающая колонка с левого края */
    margin: 0 11px; /* отступ слева и справа для красоты */
    position: relative; /* перемещаем колонку на слой */
}
```

После этого левой колонке необходимо задать правый отрицательный отступ, равный реальной ширине колонки:

```
#left_sidebar{
    width: 182px; /* задаем ширину левой колонки */
    float: left; /* плавающая колонка с левого края */
    margin: 0 11px; /* отступ слева и справа для красоты */
    position: relative; /* перемещаем колонку на слой */
    margin-right: -193px; /* исключаем содержимое колонки */
}
```

Задав правый отрицательный отступ, мы перекрыли тем самым правый отступ, который был указан ранее. По этой причине при данном способе верстки внутри тега `left_sidebar` создается еще один вложенный тег, внутри которого уже находится содержимое колонки. И все `padding` и `margin`, за исключением `margin` с отрицательным отступом, задаются вложенному тегу.

С левой колонкой разобрались. Теперь переходим к правой колонке — с идентификатором `main_content`. Прежде всего удалим все, что было написано в селекторе `#main_content` для первого варианта двухколоночной верстки. После чего добавим новые записи:

```
#main_content{
    float: right; /* плавающая колонка с правого края */
    width: 100%; /* шириной родительского тега */
}
```

Полученный вариант показан на рис. 2.40.

Рис. 2.40. Двухколоночная верстка, вариант 2

Левая колонка находится слева, а правая — справа. Правда, правая колонка наезжает на левую, но это легко исправить. Достаточно внутри тега с идентификатором `main_content` создать вложенный тег, например, с идентификатором `main_content_post`. А уже во вложенном теге разместить все содержимое колонки. После чего необходимо создать селектор со следующим содержимым:

```
#main_content_post{
    margin-left: 204px;
}
```

То есть просто задать левый отступ нужной ширины.

Данный способ является наиболее правильным методом создания двухколоночной (и многоколоночной) верстки.

Создание трех колонок

Довольно часто возникает необходимость создать верстку из трех колонок: левой и правой небольшой ширины и центральной с контентом страницы.

Такой макет выполняется по подобию варианта 2 макета для создания двухколоночной верстки.

Левая колонка при этом имеет точно такой же CSS-код:

```
#left_sidebar{
    width: 182px; /* задаем ширину левой колонки */
    float: left; /* плавающая колонка с левого края */
    margin: 0 11px; /* отступ слева и справа для красоты */
    position: relative; /* перемещаем колонку на слой */
    margin-right: -193px; /* исключаем содержимое колонки */
}
```

Центральная колонка имеет CSS-код, аналогичный CSS-коду правой колонки рассмотренного ранее макета:

```
#main_content{
    float: right; /* плавающая колонка с правого края */
    width: 100%; /* шириной родительского тега */
}
```

После чего добавляется CSS-код для правой колонки (присвоим ей идентификатор `right_sidebar`):

```
#right_sidebar{
    width: 182px; /* задаем ширину правой колонки */
    float: right; /* плавающая колонка с правого края */
```

```
    position: relative; /* перемещаем колонку на слой */
    margin-left: -193px; /* исключаем содержимое колонки */
}
```

И добавляем CSS-код для тега, вложенного внутрь центральной колонки (то есть тега с идентификатором `main_content_post` в предыдущем варианте верстки):

```
#main_content_post{
    margin: 0 204px;
}
```

Что касается HTML-макета, то в нем колонки должны размещаться в следующем порядке: сначала левая колонка, потом правая и только потом центральная.

Изучаем свойства: списки

Отдельный набор свойств CSS позволяет управлять нумерованными и ненумерованными списками, то есть тегами `ul`, `ol` и `li`.

list-style-type

Основные сведения:

- значение по умолчанию: `disc` (для `<ul>`); `decimal` (для `<ol>`);
- наследуется: да;
- применяется: к тегам `<dd>`, `<dt>`, `<li>`, `<ol>` и `<ul>`, а также ко всем элементам, у которых указано `display: list-item`;
- версия CSS: 1;
- синтаксис: `list-style-type: circle | disc | square | armenian | decimal | decimal-leading-zero | georgian | lower-alpha | lower-greek | lower-latin | lower-roman | upper-alpha | upper-latin | upper-roman | none | inherit`.

Как мы уже знаем, слева от каждого элемента списка (содержимого тега `li`) находится маркер. Какой формы будет этот маркер, зависит от свойства `list-style-type`.

В качестве значения свойства `list-style-type` может выступать одно из ключевых слов. Причем набор допустимых ключевых слов зависит от типа списка, к которому применяется свойство `list-style-type`.

Ключевые слова для ненумерованного списка:

- `circle` — маркер в виде кружка;
- `disc` — маркер в виде точки;
- `square` — маркер в виде квадрата.

Ключевые слова для нумерованного списка:

- `armenian` — традиционная армянская нумерация (6-я и 7-я версии Internet Explorer не поддерживают данное значение);
- `decimal` — арабские числа, например 1, 2, 3, 4...
- `decimal-leading-zero` — арабские числа с нулем впереди для цифр меньше десяти (6-я и 7-я версии Internet Explorer не поддерживают данное значение); например 01, 02, 03...
- `georgian` — традиционная грузинская нумерация (6-я и 7-я версии Internet Explorer не поддерживают данное значение);
- `lower-alpha` — строчные латинские буквы, например a, b, c, d...
- `lower-greek` — строчные греческие буквы (6-я и 7-я версии Internet Explorer не поддерживают данное значение);
- `lower-latin` — это значение аналогично `lower-alpha` (6-я и 7-я версии Internet Explorer не поддерживают данное значение);
- `lower-roman` — римские числа в нижнем регистре, например i, ii, iii, iv, v...
- `upper-alpha` — заглавные латинские буквы, например A, B, C, D...
- `upper-latin` — это значение аналогично `upper-alpha` (6-я и 7-я версии Internet Explorer не поддерживают данное значение);
- `upper-roman` — римские числа в верхнем регистре, например I, II, III, IV, V...

Существует также значение `none`, которое позволяет скрыть маркер, независимо от типа списка.

На рис. 2.41 показаны все возможные варианты маркера.

list-style-image

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: да;

- применяется: к тегам <dd>, <dt>, <li>, <ol> и <ul>, а также ко всем элементам, у которых указано `display: list-item`;
- версия CSS: 1;
- синтаксис: `list-style-image: none | url('путь к файлу') | inherit`.

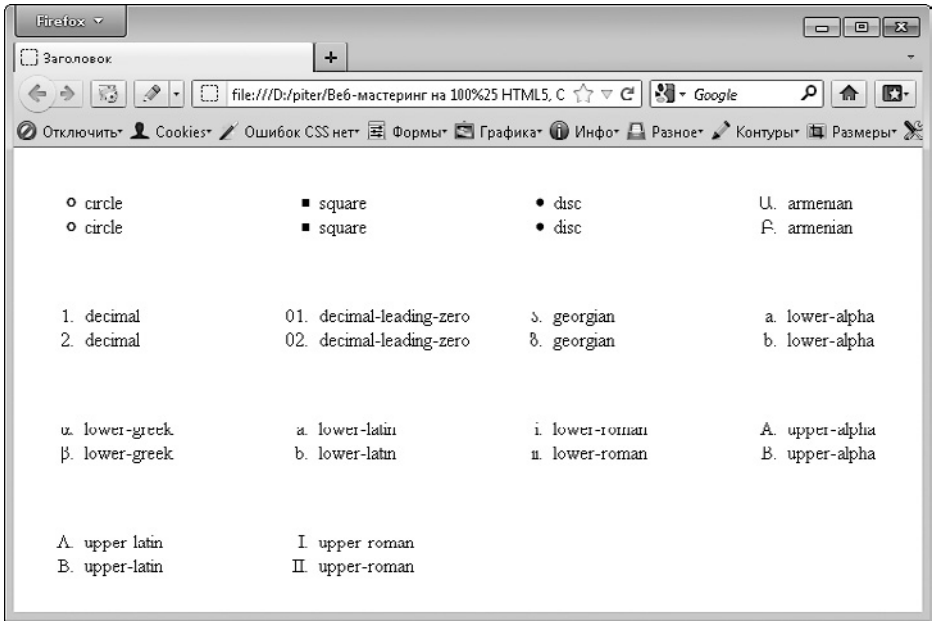


Рис. 2.41. Варианты маркеров и нумерации

Если вас не устраивает ни один из маркеров, которые может предложить свойство `list-style-type`, вам следует обратить внимание на свойство `list-style-image`. С его помощью вы можете указать путь к любому изображению, которое будет использоваться в качестве маркера (рис. 2.42). Например: `list-style-image: url("../img/marker.png");`.

Если необходимо отменить изображение, которое было объявлено ранее, достаточно воспользоваться записью `list-style-image: none;`.

list-style-position

Основные сведения:

- значение по умолчанию: `outside`;
- наследуется: да;

- применяется: к тегам <dd>, <dt>, <li>, <ol> и <ul>, а также ко всем элементам, у которых указано `display: list-item`;
- версия CSS: 1;
- синтаксис: `list-style-position: inside | outside`.


```
<div>
    <ul style="list-style-image: url(
'img/marker.gif');">
        <li>list-style-image</li>
        <li>list-style-image</li>
    </ul>
</div>
```

Рис. 2.42. Применение свойства list-style-image

Свойство `list-style-position` позволяет указать расположение маркера относительно находящегося в теге `li` текста. При значении `outside` маркер выносится за границы текста (рис. 2.43, *слева*). Значение `inside` позволяет расположить маркер внутри текстовой области (рис. 2.43, *справа*).

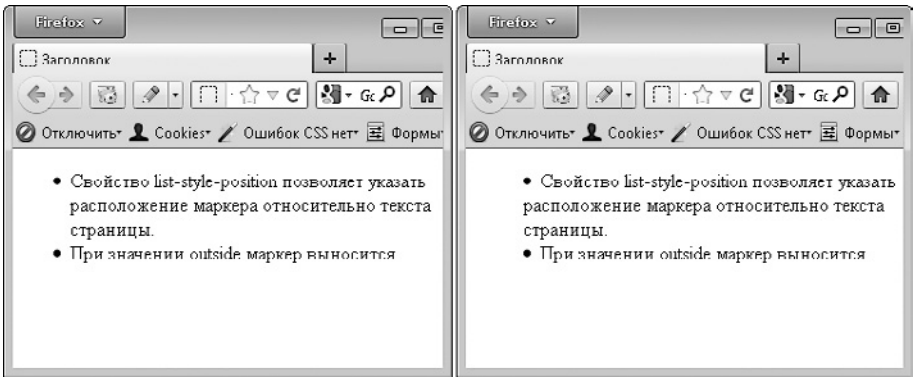


Рис. 2.43. Использование свойства list-style-position

Верстаем горизонтальное меню

Одной из задач, с которыми часто сталкивается верстальщик, является создание горизонтального меню. По правилам семантической верстки любые меню должны

создаваться с помощью ненумерованных списков. Но как создать горизонтальное меню с помощью списка, который обычно отображается вертикально?

Левостороннее меню

Горизонтальное меню, начинающееся от левой границы родительского тега, создать достаточно легко. Единственное, что нужно сделать, — присвоить несколько правил тегу `li`.

Допустим, тег `ul` горизонтального меню имеет идентификатор `primary`. Тогда:

```
ul#primary li{
    display: block; /* преобразуем в блочный элемент */
    float: left; /* преобразуем в плавающий элемент */
    list-style-type: none; /* удаляем маркер */
    list-style-image: none; /* удаляем маркер */
    padding: 0 7px; /* расстояние между пунктами меню */
}
```

Остается только добавить конструкцию `<div class="clear"> </div>` после тега `ul`, чтобы запретить обтекание плавающего меню (рис. 2.44). И, конечно, не забыть добавить определение селектора `clear`:

```
div.clear{
   clear: both;
   height: 0px;
   overflow: hidden;
}
```

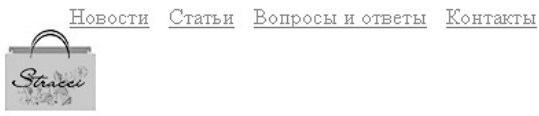


Рис. 2.44. Левостороннее меню

Центрированное меню

Для создания центрированного меню теги `li`, в которые будут взяты пункты меню, нужно преобразовать в теги строчного типа. Однако в таком случае пунктам меню нельзя будет задавать ширину и высоту, а также отступы сверху и снизу, что иногда бывает крайне необходимо. По этой причине вместо строчного типа лучше использовать значение `inline-block` свойства `display`.

Итак, список, в который заключено наше меню, имеет идентификатор `primary`:

```
ul#primary li{
    display: inline-block; /* преобразуем элемент */
    list-style-type: none; /* удаляем маркер */
    list-style-image: none; /* удаляем маркер */
    padding: 0 7px; /* расстояние между пунктами меню */
}
```

У нас получилось все то же левостороннее меню. Однако вы уже знаете о свойстве `text-align`, а значит, легко сможете центрировать пункты меню (рис. 2.45):

```
ul#primary{
    text-align: center;
}
```

Рис. 2.45. Центрированное меню

Изучаем свойства: фон

Интернет давно перестал быть только текстовой средой. Сейчас пользователи любят, чтобы все было красиво. А значит, без картинок не обойтись. В этом разделе мы научимся работать с графикой с помощью CSS.

background

Основные сведения:

- значение по умолчанию: `transparent || none || repeat || scroll || 0% 0%`;
- наследуется: нет;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис:
  - CSS2.1: `background: [background-attachment || background-color || background-image || background-position || background-repeat] | inherit`;
  - CSS3: `background: [<фон>, ]* <последний фон>`, где:

    `<фон> = [background-attachment || background-image || background-position || background-repeat] | inherit`;

    `<последний фон> = [background-attachment || background-color || background-image || background-position || background-repeat] | inherit`;

Каждому тегу на веб-странице можно присвоить свой фон: какую-нибудь картинку либо просто фон определенного типа. Для этого предназначены универсальное свойство `background` и все свойства, которые оно заменяет.

В качестве значения свойства `background` выступает такая последовательность написанных через пробел значений: `цвет картинка позиция повторение`. За каждое из этих значений отвечает отдельное свойство.

background-color. С помощью данного свойства можно задать тегу фоновый цвет. Например, `background-color: #ccc`.

Однако, даже если вам нужно задать только цвет фона, вместо свойства `background-color` лучше воспользоваться свойством `background`. Например, `background: #ccc`. Это и места меньше занимает, и позволяет в любой момент дописать свойство, задав еще и фоновую картинку.

В качестве примера добавим в макет фоновые цвета:

```
body{
    background: #e8e4d8;
}
#block_center{
    background: #fff;
}
div#block_newproduct{
    background: #e5d1ab;
}
```

background-image. Вместе с фоновым цветом можно задать и фоновую картинку. Она будет располагаться поверх фонового цвета.

Впрочем, фоновую картинку можно задать и вместо фонового цвета.

Синтаксис свойства `background-image` следующий:

```
background-image: url("путь к картинке");
```

Например, `background-image: url("../img/picture.png");`.

Однако, даже если вам нужно задать только картинку, вместо свойства `background-image` лучше использовать свойство `background`:

```
background: transparent url("путь к картинке");
```

Представленным выше примером мы задаем фоновую картинку и одновременно удаляем фоновый цвет, если он был. Впрочем, можно обойтись и без удаления фонового цвета: `background: url("путь к картинке");`.

background-position. Фоновая картинка может располагаться в любом месте тега. Ее начальные координаты (координаты левого верхнего угла картинки) определяются с помощью данного свойства:

```
background-position: горизонтальное_смещение вертикальное_смещение;
```

Горизонтальное смещение задается ключевыми словами `left`, `right` и `center`, понятными без пояснения.

Вертикальное смещение указывается ключевыми словами `top`, `bottom` и `center`.

Помимо ключевых слов, для задания позиции фоновой картинки можно использовать смещение в пикселах или процентах. Например, `background-position: 77px 34px`. Первое число задает смещение по горизонтали, начиная от левой границы тега. Второе число указывает смещение по вертикали, начиная от верхней границы тега. К сожалению, смещения от правой и нижней границ тега задавать нельзя.

Что касается универсального свойства, оно пишется вместе с позицией следующим образом:

```
background: transparent url("../img/pict.png") left top;

background: #ccc url("../img/pict.png") right top;
```

background-repeat. Данное свойство также влияет на фоновую картинку. Оно позволяет задать повторение для фоновой картинки, то есть размножить фоновую картинку, если она не закрывает весь тег (рис. 2.46):

- `no-repeat` — не повторять картинку (значение по умолчанию);
- `repeat-x` — повторять картинку по горизонтали;
- `repeat-y` — повторять картинку по вертикали;
- `repeat` — повторять картинку и по горизонтали, и по вертикали.

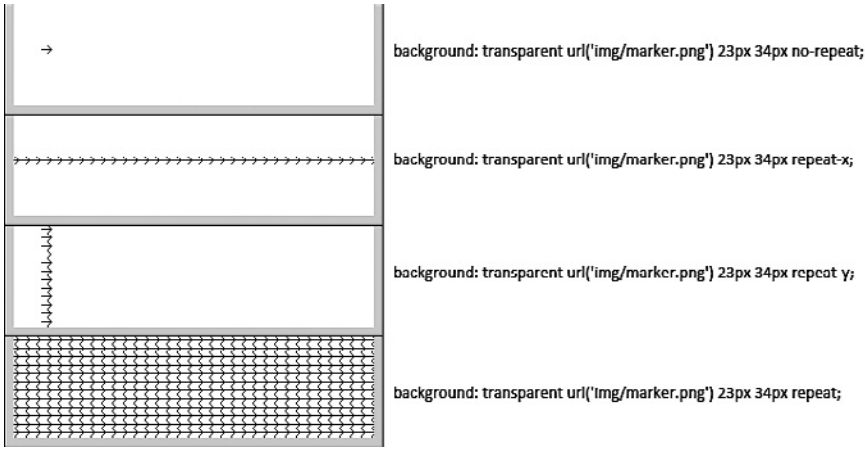


Рис. 2.46. Варианты размножения фоновой картинки

Итак, полный универсальный тег записывается подобным образом:

```
background: #ccc url("../img/pict.png") left top repeat;
```

При этом последовательность значений сохранять не обязательно:

```
background: #ccc url("../img/pict.png") repeat left top;

background: url("../img/pict.png") left top repeat #ccc;
```

background-attachment

Основные сведения:

- значение по умолчанию: `scroll`;
- наследуется: нет;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис:
  - CSS2.1: `background-attachment: fixed | scroll | inherit`;
  - CSS3: `background-attachment: fixed | scroll | local[, fixed | scroll | local]*`.

В состав универсального свойства `background` также входит свойство `background-attachment`. Однако применяют его настолько редко, что лучше задавать его отдельно при необходимости.

А между тем свойство `background-attachment` позволяет задать довольно интересный эффект.

По умолчанию фоновая картинка прокручивается вместе со всем содержимым тега (если пользователь применяет горизонтальную/вертикальную прокрутку для навигации по веб-странице). Значение `fixed` свойства `background-attachment` позволяет зафиксировать фоновую картинку на странице. Например, если вы добавите тегу `body` фиксированную фоновую картинку, расположенную в правом верхнем углу, то даже после горизонтальной прокрутки фоновая картинка будет располагаться в правом верхнем углу. Таким образом, действие свойства `background-attachment` чем-то напоминает правило `position: fixed`.

Применять фиксированное позиционирование фоновой картинки рекомендуется только для тегов `body` и `html`. Потому что браузер Internet Explorer 6 не поддерживает значение `fixed` свойства `background-attachment` для других тегов.

Изучаем свойства: текст

Сегодня без картинок и красивого оформления, без сомнения, не обойтись. Но главным все-таки по-прежнему остается текстовая составляющая веб-страницы. Текст должен легко читаться, обращать на себя внимание и, конечно, быть красивым.

font-size

Основные сведения:

- значение по умолчанию: `medium`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `font-size: абсолютный размер | относительный размер | значение | проценты | inherit`.

Чтобы текст легко читался, размер шрифта должен быть большим. Впрочем, многие дизайнеры любят мелкие шрифты. Однако нельзя забывать, что далеко не у всех пользователей идеальное зрение.

В любом случае при верстке HTML-документа вам нужно задать размер шрифта. Это делается с помощью свойства `font-size`. Рекомендуется указывать размер в пунктах, например `font-size: 12pt;`.

Принято задавать размер шрифта по умолчанию для всех тегов. После чего переопределять его при необходимости. Например, так:

```
body, p, div, td, input, textarea{
   font-size: 12pt;
}
```

ПРИМЕЧАНИЕ

При верстке с нуля достаточно присваивать теги свойству body. Однако при верстке для CMS Drupal, о которой речь пойдет далее (см. главу 5), необходимо также присваивать свойства тегам p, div, td, input, textarea. Дело в том, что в стандартном CSS-файле CMS Drupal этим тегам уже заданы некоторые свойства.

В синтаксисе данного свойства указано, что в качестве его значения может выступать абсолютный или относительный размер.

В данном случае под абсолютным размером понимаются следующие ключевые слова: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`.

Под относительным же размером имеются в виду ключевые слова `larger`, `smaller`, позволяющие уменьшить или увеличить размер шрифта относительно его текущего размера.

color

Основные сведения:

- значение по умолчанию: зависит от настроек браузера, обычно черный цвет;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `color: цвет | inherit`.

Как показывает время, лучший цвет для текста — черный. Но дизайнеры всего мира часто имеют свое видение данного вопроса. По этой причине верстальщику никогда не помешает умение изменять цвет шрифта. Это делается с помощью свойства `color`, например `color: #ff0000;`.

font-style

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `font-style: normal | italic | oblique | inherit`.

Чтобы сделать текст курсивным, достаточно свойству `font-style` присвоить значение `italic`. Значением `oblique` также отображается как курсив, но все же лучше пользоваться значением `italic`.

Если вы хотите сделать текст снова обычным, то воспользуйтесь правилом `font-style: normal;`.

font-weight

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;

- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `font-weight: bold|bolder|lighter|normal|100|200|300|400|500|600|700|800|900`.

С помощью свойства `font-weight` можно задать тексту полужирное начертание. Для этого нужно присвоить данному свойству значение `bold`. Чтобы вернуть обычный шрифт, достаточно присвоить данному свойству значение `normal`.

Этих значений вполне достаточно. Остальные значения данного свойства позволяют варьировать полужирность шрифта в определенных пределах. Однако эта возможность существует не для всех шрифтов.

line-height

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `line-height: множитель | значение | проценты | normal | inherit`.

В некоторых случаях данное свойство является крайне полезным. Оно позволяет задать высоту строки шрифта, то есть определить расстояние между строками, а также расположение строки шрифта по высоте.

В обычных ситуациях высота строки не имеет большого значения, конечно, если расстояние между строками не сильно большое или маленькое.

Но бывают случаи, когда от высоты строки зависит многое. Например, довольно часто текст, расположенный в значении атрибута `value` тега `input` типа `text`, в различных браузерах размещается на разном расстоянии от нижней границы текстового поля. В Mozilla Firefox он располагается по центру текстовой строки, а в браузере Internet Explorer — практически впритык к нижней границе, что выглядит весьма некрасиво и неудобно. Чтобы решить эту проблему, достаточно подобрать значение свойства `line-height` таким образом, чтобы поднять текст в браузере Internet Explorer.

Значения свойства `line-height` чаще всего задаются в `em`. `1em` равен высоте шрифта, поэтому для комфортного чтения нужно задавать значение, примерно равное `1.2em`. Чтобы между строками было свободное расстояние.

letter-spacing

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `letter-spacing: значение | normal | inherit`.

Расстояние между буквами шрифта задается очень редко. Но если вам все-таки понадобилось сделать разреженный или уплотненный текст, помочь в этом может свойство `letter-spacing`. В качестве значения данного свойства чаще всего указывается расстояние между символами шрифта в пикселах.

white-space

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;
- применяется: к блочным элементам;
- версия CSS: 1;
- синтаксис: `white-space: normal | nowrap | pre | pre-line | pre-wrap | inherit`.

С помощью данного свойства можно определить, как именно браузер будет интерпретировать пробелы в тексте. Как вы уже знаете, в HTML несколько идущих подряд пробелов заменяются одним пробелом. Однако свойство `white-space` позволяет отменить данное поведение.

Чаще всего вы будете пользоваться значением `nowrap` свойства `white-space`. С его помощью все пробелы в строке можно сделать неразрывными. Это необходимо, если вы хотите, чтобы определенный текст всегда находился на одной строке и не переносился на следующую. Например, такое поведение часто бывает необходимо для пунктов меню.

В качестве значения свойства `white-space` могут также выступать следующие ключевые слова (рис. 2.47):

- `pre` — преобразует текущий тег в тег `pre` (текст будет отображаться с учетом всех заданных в нем пробелов и переносов);

- `pre-line` — пробелы в тексте не учитываются, текст автоматически переносится на новую строку, если не помещается на текущей (не поддерживается версиями 6 и 7 браузера Internet Explorer);
- `pre-wrap` — все пробелы и переносы сохраняются в тексте, однако, если текст не помещается на текущей строке, он автоматически будет перемещен на новую (не поддерживается версиями 6 и 7 браузера Internet Explorer).

Рис. 2.47. Варианты свойства white-space

text-indent

Основные сведения:

- значение по умолчанию: 0;
- наследуется: да;
- применяется: к блочным элементам;
- версия CSS: 1;
- синтаксис: `text-indent: <значение> | <проценты> | inherit`.

Свойство `text-indent` позволяет задать красную строку, то есть определяет отступ для первой строки каждого абзаца текста.

Значение данного свойства чаще всего указывается в пикселах.

При задании значения в процентах оно вычисляется относительно ширины блока.

Значение свойства `text-indent` может быть как положительным, так и отрицательным. Положительное значение задает красную строку (рис. 2.48, *слева*). Отрицательное значение позволяет создать выступ для первой строки каждого абзаца (рис. 2.48, *справа*).

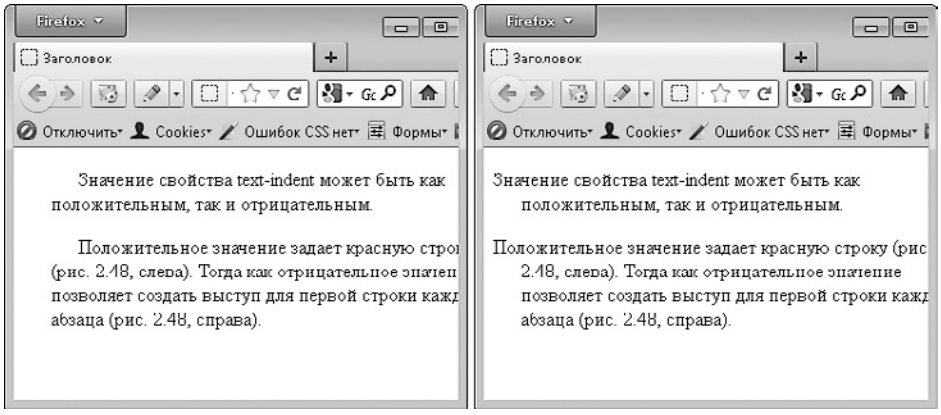


Рис. 2.48. Варианты свойства text-indent

text-decoration

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: нет;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `text-decoration: [ blink | line-through | overline | underline ] | none | inherit`.

Благодаря данному свойству вы легко можете создать подчеркнутый, зачеркнутый и другие варианты текста. Применяемый эффект зависит от используемого значения свойства (рис. 2.49):

- `blink` — создать мигающий текст (поддерживается только браузерами Opera и Firefox);
- `line-through` — создать перечеркнутый текст (в Internet Explorer 7 перечеркивающая линия располагается выше, чем в других браузерах);
- `overline` — создать текст с чертой сверху;
- `underline` — создать подчеркнутый текст;
- `none` — отменить примененный ранее эффект.

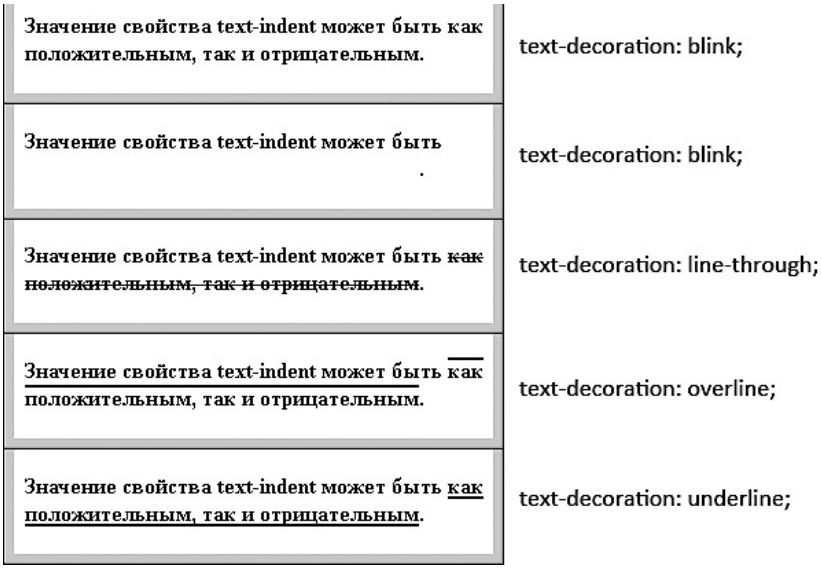


Рис. 2.49. Варианты свойства text-decoration

text-transform

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `text-transform: capitalize | lowercase | uppercase | none | inherit`.

Данное свойство позволяет преобразовывать написанный текст в строчный или прописной (рис. 2.50). Оно поддерживает следующие значения:

- `capitalize` — сделать первый символ каждого слова прописным;
- `lowercase` — сделать все символы в тексте строчными (перевести в нижний регистр);
- `uppercase` — сделать все символы в тексте прописными (перевести в верхний регистр);
- `none` — отображать текст так, как он написан в HTML-документе.

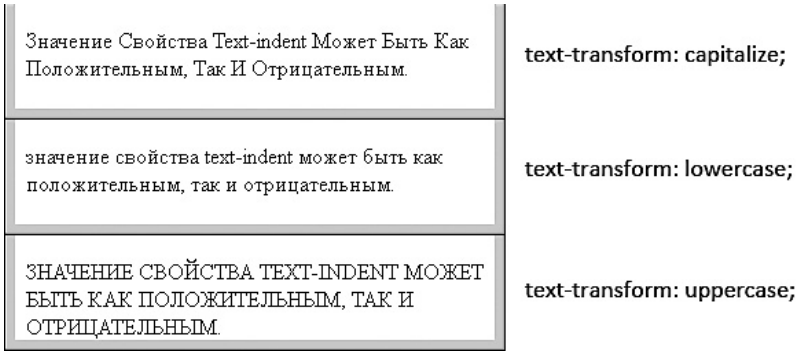


Рис. 2.50. Варианты применения свойства text-transform

font-variant

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `font-variant: normal | small-caps | inherit`.

Свойство `font-variant` позволяет создать капитель, то есть преобразует все строчные буквы в тексте в малые прописные (заглавные уменьшенного размера) (рис. 2.51). Чтобы создать капитель, достаточно присвоить данному свойству значение `small-caps`. Отменить создание капители позволяет значение `normal`.

word-spacing

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;

- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `word-spacing: значение | normal | inherit`.

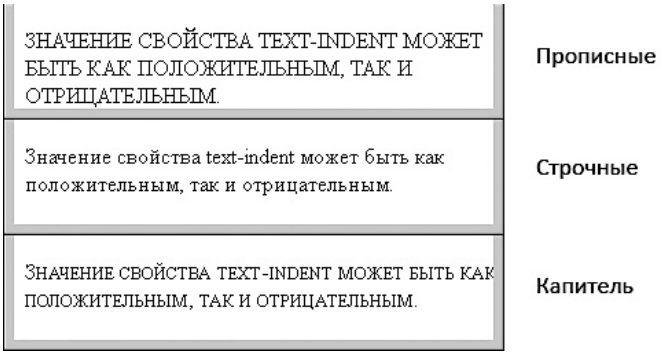


Рис. 2.51. Использование свойства font-variant

Данное свойство позволяет увеличить или уменьшить интервал между словами в тексте.

Стоит учитывать, что данное свойство не будет работать, если тексту присвоено свойство `text-align` со значением `justify`.

unicode-bidi

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: нет;
- применяется: ко всем элементам;
- версия CSS: 2;
- синтаксис: `unicode-bidi: normal | embed | bidi-override | inherit`.

Свойство `unicode-bidi` управляет действием свойства `direction`, которое будет рассмотрено ниже. Возможны следующие значения данного свойства:

- `normal` — направление чтения текста определяется автоматически на основе используемых символов Unicode;
- `embed` — направление чтения текста задается значением свойства `direction`;

- `bidi-override` — направление чтения текста и порядок символов в тексте определяется значением свойства `direction`.

direction

Основные сведения:

- значение по умолчанию: `ltr`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 2;
- синтаксис: `direction: ltr | rtl | inherit`.

В некоторых языках слова читаются справа налево. Свойство `direction` позволяет учитывать особенности используемого языка, определяя, в каком именно направлении читается написанный текст:

- `ltr` — текст читается слева направо и выравнивается по левому краю;
- `rtl` — текст читается справа налево и выравнивается по правому краю.

quotes

Основные сведения:

- значение по умолчанию: зависит от браузера, его настроек и используемой операционной системы. Чаще всего применяются кавычки вида `"/"`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 2;
- синтаксис: `quotes: "левая кавычка" "правая кавычка" | none | inherit`.

В каждом языке используется свой вариант кавычек. Свойство `quotes` позволяет определить, как будут выглядеть кавычки в вашем HTML-документе. Оно влияет на кавычки, добавляемые содержимому тега `q`, а также на текст, к которому применяется стилевое свойство `content` со значением `open-quote` или `close-quote`.

В качестве значения данного свойства выступают левая и правая кавычки, написанные через пробел. Как правило, кавычки задаются символами Unicode. Например, `quotes: "\00ab" "\00bb";`.

В табл. 2.2 представлены наиболее часто используемые кавычки.

Таблица 2.2. Наиболее часто используемые кавычки

Вид	Спецкод	Юникод	Описание
"	&#34;	\0022	Двойная кавычка. Применяется в моноширинных шрифтах для обозначения символа дюйма, угловых секунд
'	&#39;	\0027	Апостроф. Символ угловых минут
«	« или «	\00ab	Открывающая двойная угловая кавычка
»	» или »	\00bb	Закрывающая двойная угловая кавычка
‘	&#8216;	\2018	Открывающая одинарная кавычка
’	&#8217;	\2019	Закрывающая одинарная кавычка
“	&#8220;	\201c	Открывающая двойная кавычка
”	&#8221;	\201d	Закрывающая двойная кавычка
„	&#8222;	\201e	Открывающая двойная кавычка

font-family

Основные сведения:

- значение по умолчанию: шрифт, установленный в браузере по умолчанию (обычно это Times New Roman);
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 1;
- синтаксис: `font-family: имя шрифта [, имя шрифта[, ...]] | inherit`.

С помощью данного свойства определяется шрифт, который будет использован для отображения текста. Достаточно указать имя шрифта в качестве значения данного свойства. Причем, если в имени шрифта есть пробелы, само имя необходимо взять в кавычки.

При этом нужно учитывать следующий момент. Каждая операционная система имеет свой набор установленных шрифтов. Причем количество операционных систем не ограничивается только Windows, Linux и Mac OS. Не стоит также забывать и об операционных системах, установленных на смартфонах и других устройствах, с помощью которых можно просматривать веб-страницы. Поэтому вполне вероятна ситуация, когда на устройстве посетителя не обнаруживается шрифт, заданный вами в свойстве `font-family`.

Если на устройстве посетителя нет подходящего шрифта, то текст веб-страницы отображается шрифтом по умолчанию. И этот шрифт может выглядеть совсем не так, как вы задумывали.

Чтобы не полагаться на случайность, верстальщики задают в свойстве `font-family` сразу несколько шрифтов, перечисляя их через запятую. Например:

```
font-family: "Comic Sans MS", Arial, Tahoma, Myriad;
```

При этом браузер для вывода текста будет применять первый из списка заданных шрифтов. Если первый шрифт не обнаружен в системе пользователя, то для отображения текста применяется второй шрифт. Если второго нет — третий и т. д.

Тип шрифта

В качестве значения свойства `font-family` может выступать не только имя определенного шрифта, но и ключевое слово, задающее тип шрифта. В этом случае для отображения текста будет использоваться шрифт по умолчанию, относящийся к шрифтам выбранного типа.

Свойство поддерживает следующие ключевые слова:

- `serif` — отображать текст любым шрифтом с засечками (антиква);
- `sans-serif` — любым шрифтом без засечек (рубленый, гротеск);
- `cursive` — любым курсивным шрифтом;
- `fantasy` — любым декоративным шрифтом;
- `monospace` — любым моноширинным шрифтом (то есть в котором ширина каждого символа одинакова).

Как правило, тип шрифта указывается в самом конце списка шрифтов и применяется браузером только в крайнем случае, когда другие шрифты из списка найдены не были.

Пример использования свойства `font-family` с указанием типа шрифта:

```
font-family: "Comic Sans MS", Arial, Tahoma, Myriad, sans-
serif;
```

Безопасные шрифты

Под безопасным шрифтом понимается шрифт, который с вероятностью 99,9 % установлен в системе посетителя. Таких шрифтов не очень много. Чаще всего в свойстве `font-family` задается список шрифтов, в котором указаны сходные шрифты для разных операционных систем.

В своих проектах вы можете без опаски использовать следующие варианты значений свойства `font-family`:

Verdana, Geneva, sans-serif

Georgia, Times New Roman, Times, serif

`Courier New, Courier, monospace`

Arial, Helvetica, sans-serif

Tahoma, Geneva, sans-serif

Trebuchet MS, Arial, Helvetica, sans-serif

Arial Black, Gadget, sans-serif

Times New Roman, Times, serif

Palatino Linotype, Book Antiqua, Palatino, serif

Lucida Sans Unicode, Lucida Grande, sans-serif

MS Serif, New York, serif

`Lucida Console, Monaco, monospace`

Comic Sans MS, cursive

Использование нестандартных шрифтов

На практике довольно часто возникает необходимость написать определенный текст нестандартным шрифтом. Вот хочется дизайнеру, чтобы заголовки отображались каким-нибудь причудливым шрифтом, и ничего тут не поделаешь.

Конечно, вы можете поступить просто. Добавить имя нестандартного шрифта в значение свойства `font-family` и надеяться, что дизайнер не скоро заметит, что текст отображается правильным шрифтом только на его компьютере. Ведь у дизайнера на компьютере установлен нестандартный шрифт, поэтому для него текст на странице будет отображаться правильно.

Данный способ прост. Но, говорят, после его применения верстальщики спят плохо. Совесть мучает.

К счастью, существует способ использовать в верстке любой нестандартный шрифт, причем он будет отображаться для всех посетителей, а не только для «избранных». Однако, чтобы показать текст нестандартным шрифтом, браузеру сначала нужно будет скачать этот шрифт. Таким образом, нестандартные шрифты скачиваются браузером перед тем, как открыть веб-страницу. Соответственно, крайне не рекомендуется использовать тяжелые нестандартные шрифты, размером более мегабайта.

Для отображения текста нестандартным шрифтом используется технология Cufon. Она применяет JavaScript для замены используемого шрифта на нестандартный

после открытия веб-страницы. JavaScript сейчас используется повсеместно. И даже если у посетителя отключен JavaScript, он все равно увидит текст на странице, правда оформленный обычным шрифтом.

Подготавливаем шрифт

Перед тем как использовать технологию Cufon, необходимо найти файл нужного вам шрифта в формате TTF, OTF, PFB или PS.

Если шрифт установлен на вашем компьютере, его файл можно найти в папке `%systemroot%\Fonts`. В противном случае нужно искать шрифт в Интернете.

ПРИМЕЧАНИЕ

%systemroot% — это системная переменная Windows. Она указывает на папку Windows (содержит файлы операционной системы). Как правило, %systemroot% указывает на папку C:\Windows, где C:\ — раздел диска, в котором установлена операционная система Windows. Вы можете также открыть нужную папку со шрифтами, введя %systemroot%\Fonts в диалоге Выполнить.

После того как шрифт найден, необходимо преобразовать его в файл специального формата JS, содержащий в себе символы шрифта. Это делается на странице http://cufon.shoqolate.com/generate (рис. 2.52).

Рис. 2.52. Подготавливаем шрифт

Первым делом необходимо выбрать файл шрифта в одном из полей области Select the font you'd like to use (Выберите шрифт, который хотите использовать). При этом не стоит ошибаться с выбором поля. Если вы создаете полужирное начертание шрифта, то следует использовать поле именно для полужирного начертания шрифта — Bold typeface (optional) (Полужирное начертание (опционально)).

Здесь, наверное, следует дать некоторые пояснения. Это только в программах обычное, полужирное, курсивное начертания — это один и тот же шрифт. На самом деле для каждого начертания существует свой файл шрифта. Поэтому, если вам нужно несколько начертаний нестандартного шрифта, вам придется подключить сразу несколько файлов шрифта. Причем создавать JS-файлы шрифта лучше отдельно для каждого начертания.

После того как файл шрифта выбран, следует заполнить остальные поля формы:

- Use the following value as the font-family of the generated font (optional) (Использовать следующее значение в качестве имени создаваемого шрифта (опционально)) — введите имя шрифта (на английском), которое в дальнейшем будет использоваться в HTML-документе для идентификации шрифта;
- The EULAs of these fonts allow Web Embedding (without Adobe Flash) (Лицензионное соглашение на данный шрифт разрешает его встраивание (без Adobe Flash)) — установите данный флажок;
- Include the following glyphs (if available) (Включить следующие символы (если доступны)) — в данной области выберите типы символов, которые будут входить в состав созданного JS-файла, либо установите флажок All (Все);
- I acknowledge and accept these terms (Я принимаю условия соглашения) — установите данный флажок.

После выполнения перечисленных действий остается только нажать кнопку Let's do this! (Сделайте это!) и через некоторое время скачать сгенерированный JS-файл шрифта.

Если вместо JS-файла шрифта открылась страница с сообщением об ошибке, еще раз проверьте расширение выбранного вами файла шрифта. Он должен иметь расширение TTF, OTF, PFB или PS.

Если расширение входит в перечисленный список, значит, создатель шрифта запретил возможность встраивать свое творение с помощью Cufon. И в этом случае ничего не поделаешь, придется выбрать другой шрифт. Однако такие ситуации бывают редко. Мне на практике лишь раз встречался шрифт, запрещенный к встраиванию в Cufon.

Проверьте размер JS-файла шрифта. Если он превышает 500 Кбайт, стоит задуматься, действительно ли вам нужны все символы, которые входят в состав шриф-

та. И пересоздать JS-шрифт, на этот раз выбрав в области Include the following glyphs (if available) (Включить следующие символы (если доступны)) вместо флажка All (Все) флажки Uppercase (Верхний регистр), Lowercase (Нижний регистр), Numerals (Цифры), Punctuation (Пунктуация), Basic Latin (Базовая латиница), Latin-1 Supplement (Группа Latin 1), Russian Alphabet (Русский алфавит).

Подключаем Cufon

После того как JS-файл шрифта создан, можно переходить к подключению Cufon.

Сначала нужно скачать файл Cufon со страницы http://cufon.shoqolate.com/generate/ (вкладка Download (Скачать)). После нажатия Download (Скачать) откроется нужный файл. Выберите пункт Файл ▸ Сохранить в меню браузера и сохраните файл `cufon-yui.js` в папке вашего сайта.

Файл `cufon-yui.js` можно также найти в папке tech/cufon среди файлов, которые вы можете скачать с сайта http://piter.com для данной книги.

Как правило, все JS-файлы размещают в папке js сайта. После того как вы поместите файл `cufon-yui.js` в эту папку, перенесите также в нее и JS-файл шрифта.

Далее создайте в папке js пустой текстовый документ и переименуйте его в `cufon-replace.js` (то есть с расширением JS, а не TXT). В дальнейшем он нам пригодится.

После того как все нужные файлы созданы, остается только подключить их к веб-странице. Для этого внутрь тега `head` необходимо добавить следующие строки:

```
<script type="text/javascript" src="js/cufon-yui.js"></script>

<script type="text/javascript" src="js/cufon-replace.js"></
script>

<script type="text/javascript" src="js/JS-файл вашего шрифта.
js"></script>
```

Настраиваем Cufon

Итак, JS-файл создан, Cufon подключен. Теперь нужно указать теги для текста, в которых будет применяться нестандартный шрифт. Для этого откройте файл `cufon-replace.js` в любом текстовом редакторе и внесите в него строки следующего вида:

```
Cufon.replace('селектор', { fontFamily: 'имя шрифта', hover:
true, ignoreClass:'nocufon' });
```

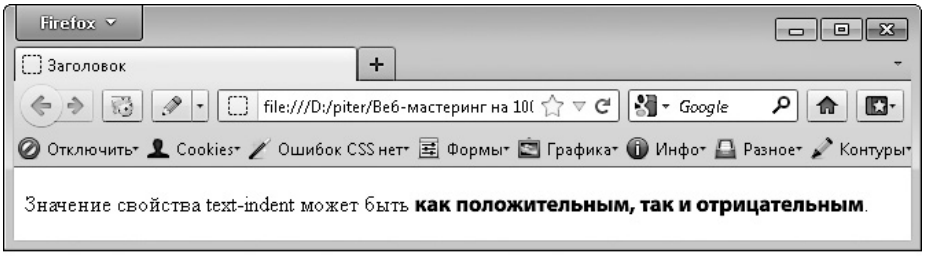
Под именем шрифта имеется в виду имя, которое вы указали в поле Use the following value as the font-family of the generated font (optional) (Использовать следующее значение в качестве имени создаваемого шрифта (опционально)) при создании JS-файла шрифта.

Если вы используете один и тот же шрифт для нескольких тегов, вы можете указать их селекторы в одной строке через запятую.

Например:

```
Cufon.replace('div#content, div#footer .content',  { fontFamily:
'PragmaticaLightC', hover:true, ignoreClass:'nocufon' });
```

Если вы все сделали правильно, ваша страница заметно преобразится (рис. 2.53).




```
<head>
    <title>Заголовок</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <link type="text/css" rel="stylesheet" href="css/style.css" media="screen" />
    <script type="text/javascript" src="js/cufon-yui.js"></script>
    <script type="text/javascript" src="js/cufon-replace.js"></script>
    <script type="text/javascript" src="js/MyriadPro_700.font.js"></script>
</head>
<body>
<p>Значение свойства text-indent может быть <span>как положительным, так и
 отрицательным</span>.</p>
```

```
Cufon.replace('p span',  { fontFamily: 'MyriadPro', hover:true, ignoreClass:
'nocufon' });
```

Рис. 2.53. Использование Cufon

Выбираем шрифт

Одна из проблем, с которой вы можете столкнуться при подключении нестандартных шрифтов, — отсутствие в шрифте символов русского алфавита. С этим ничего не поделаешь: если в шрифте вообще нет русских символов, придется выбрать другой шрифт.

Помочь в выборе шрифта с русскими символами может сайт http://www.fonts-online.ru/. Данный сервис позволяет не только выбрать шрифт, содержащий русские символы, но и сразу увидеть, как будет выглядеть нужная вам строка текста при использовании данного шрифта.

Введите фразу в поле Текст примера и нажмите кнопку Обновить. После чего перейдите в один из разделов с кириллическими шрифтами (рис. 2.54).

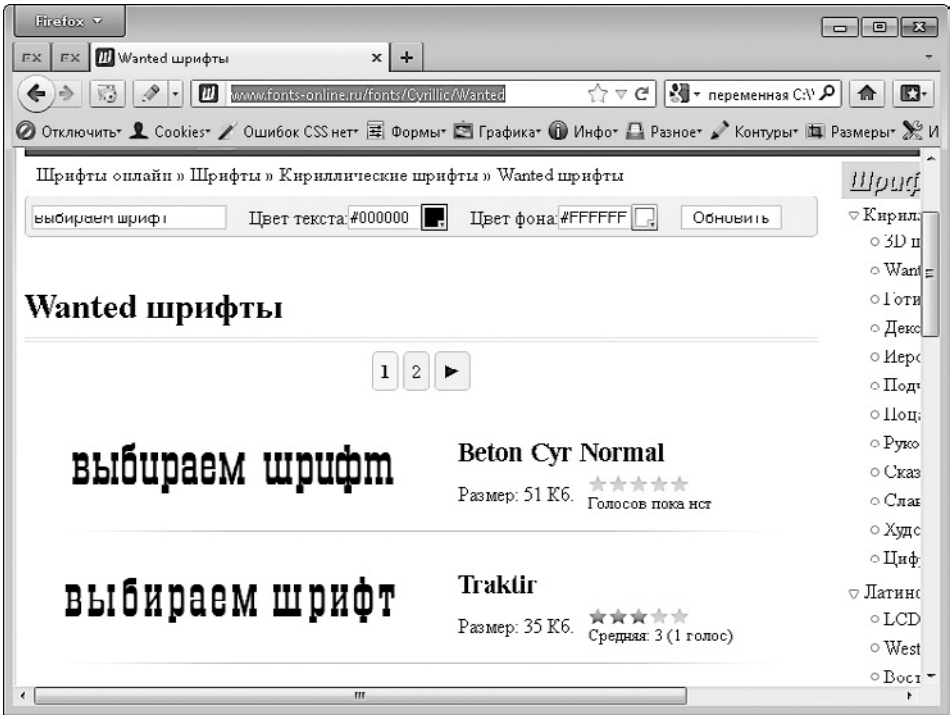


Рис. 2.54. Выбираем шрифт

На сайте все шрифты разбиты по категориям, что существенно облегчает поиск. Среди кириллических доступны шрифты следующих категорий:

- 3D-шрифты — шрифты с тенью или с эффектом объема;
- Wanted шрифты — шрифты прошлых лет, которыми в фильмах о ковбоях часто пишут вывески в салунах, а также объявления о поиске преступников;
- Готические шрифты — рубленые;
- Декоративные шрифты — с необычным начертанием;
- Иероглифические шрифты — шрифты, символы в которых стилизованы под иероглифы или египетские надписи;

- Подчеркнутые шрифты — подчеркнутые, надчеркнутые и перечеркнутые шрифты;
- Поцарапанные шрифты — с эффектом потертости;
- Рукописные шрифты — имитирующие текст, написанный от руки;
- Сказочные шрифты — с завитушками, такие шрифты часто используются в книгах для детей;
- Славянские шрифты — имитирующие славянские символы;
- Художественные шрифты — шрифты с еще более необычным начертанием, чем декоративные;
- Цифровые шрифты — имитирующие надписи на часах и других электронных устройствах.

Вы можете также выбрать цвета фона и текста, чтобы увидеть, как фраза, написанная выбранным шрифтом, будет выглядеть в нужном цвете.

Определение шрифта

Иногда возникает необходимость определить имя шрифта по картинке с надписью, сделанной этим шрифтом. Например, вы обнаружили рекламный буклет или баннер, в котором используется подходящий для вашего сайта шрифт. Как же узнать, что это за шрифт?

Решить поставленную задачу поможет сервис http://www.myfonts.com/WhatTheFont/. Первым делом следует указать путь к картинке, содержащей строку текста, выполненную неизвестным шрифтом (рис. 2.55). Строка текста должна быть на английском языке, с русскими символами сервис работает хуже.

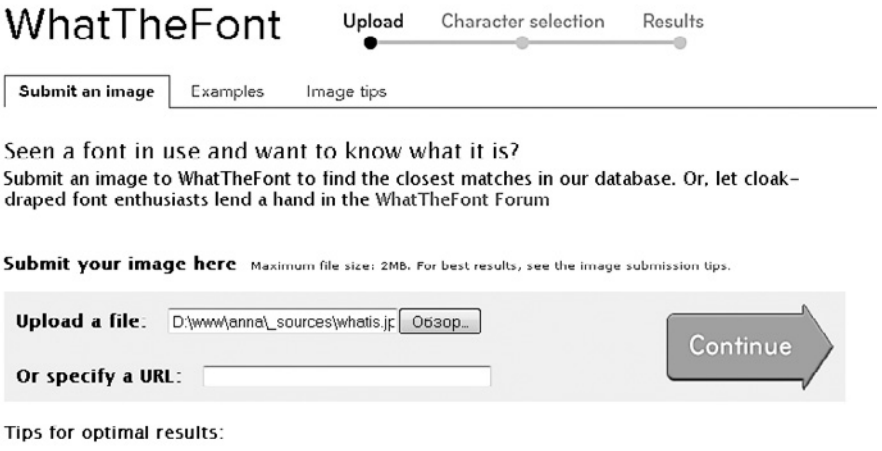


Рис. 2.55. Выбираем картинку со шрифтом

Далее отобразится список символов, которые сервис обнаружил на картинке (рис. 2.56). Рядом с каждым символом расположено текстовое поле, в котором нужно указать, что это за символ. Если сервис ошибся и на самом деле найденная закорючка не является полноценным символом, оставьте соответствующее поле пустым. Так часто бывает с необычными шрифтами. В них отдельные символы нередко определяются сразу как несколько символов.

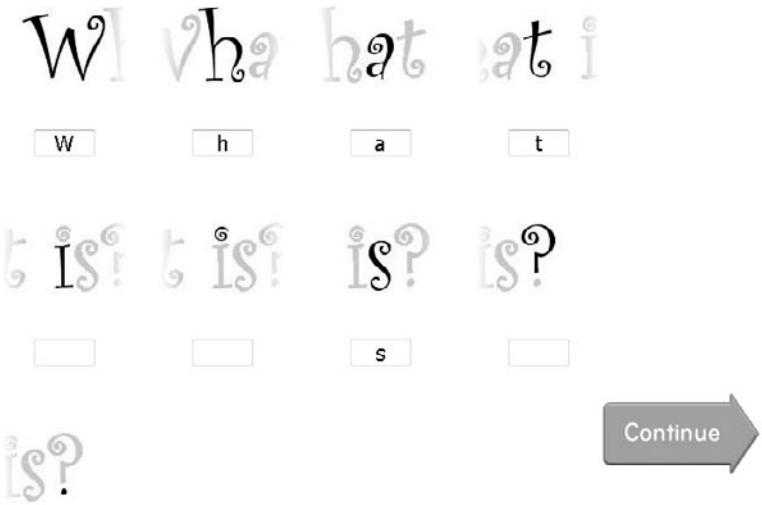


Рис. 2.56. Определяем символы, изображенные на картинке

На последнем шаге сервис отображает список шрифтов, которые напоминают наш (рис. 2.57). Выберите среди них наиболее подходящий.

Тень под текстом

Еще одной нестандартной задачей, над которой часто приходится ломать голову верстальщику, является создание тени под текстом, которую дизайнер нарисовал в макете.

Самый простой метод добавить тень под текстом — сделать и тень, и сам текст картинкой. Но это плохой способ. Настоящие верстальщики так не поступают.

Тень под Cufon-текстом

Если вам нужно создать тень под текстом с нестандартным шрифтом, добавленным Cufon, следует использовать возможности самого Cufon. Другие методы работать не будут.

Рис. 2.57. Просматриваем список найденных шрифтов

Сделать это несложно. Достаточно изменить строку применения нестандартного шрифта в файле `cufon-replace.js` на подобную:

```
Cufon.replace('селектор', { fontFamily: 'имя шрифта', hover:
true, ignoreClass:'nocufon', textShadow: '#110a03 0px 2px' });
```

То есть добавить свойство `textShadow` (рис. 2.58) с тремя написанными через пробел значениями, задающими цвет тени и смещение тени по вертикали и горизонтали. Например, `Cufon.replace('#right h2', { fontFamily: 'MyArialBold', fontWeight: 'bold', textShadow: '#110a03 0px 2px', hover:true, ignoreClass:{notcufon : 1} });`.

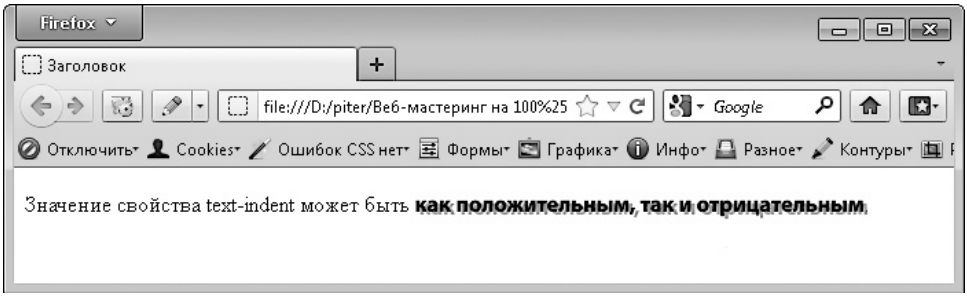


Рис. 2.58. Создание тени под Cufon-шрифтом

Тень под обычным текстом

Если тень нужна для обычного текста, на помощь придет JavaScript. Для этой цели можно использовать один из двух популярных сценариев: `text-shadow.min.js` или `jquery.dropshadow.js`. Однако оба этих сценария работают с библиотекой jQuery, рассматривать которую мы будем в главе 3, посвященной JavaScript. По этой причине отложим тему теней под обычным текстом до более подходящего времени, когда вы уже хоть что-то будете знать о JavaScript и jQuery.

Градиентный текст для Cufon

Еще одной интересной возможностью, которую предоставляет Cufon, является возможность создания градиентного текста, то есть текста, у которого цвет верхней части символов отличается от цвета нижней части символов (рис. 2.59).

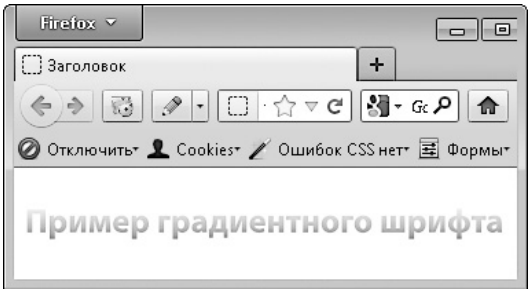


Рис. 2.59. Градиентный текст

Градиентный текст создается добавлением специального свойства в строку назначения Cufon-шрифта в файле `cufon-replace.js`:

```
Cufon.replace('селектор', { fontFamily: 'имя шрифта', hover:
true, ignoreClass:'nocufon', color: "-linear-gradient(цвет
вверху, цвет внизу)" });
```

Например: `Cufon.replace('h1.title, h2.title a', { fontFamily: 'MyArialBold', fontWeight: 'bold', color: "-linear-gradient (#FFFFFF, #AAAAAA)", textShadow: '#110a03 0px 4px', hover:true, ignoreClass:{notcufon : 1} });.`

Другие возможности Cufon

Думаю, вы уже догадались, что внутри фигурных скобок второго параметра метода `Cufon.replace` можно задавать самые разные свойства. Помимо `textShadow` и `color`, Cufon поддерживает множество других свойств (табл. 2.3).

Таблица 2.3. Свойства Cufon

Свойство	Значение по умолчанию	Значения	Описание
autoDetect	false	true, false	Если значение true, свойство font-family берется из CSS, а не из fontFamily. Однако данный способ не работает в браузере Opera
color	Определяется CSS	'red', '#f62315'	Обычный либо градиентный цвет текста
fontFamily	Последний загруженный шрифт	Название шрифта	Название шрифта, который вы хотите использовать
fontSize	Определяется CSS	'14px', '24px', '72px', ..	Размер шрифта в пикселах
fontStretch	'normal'	'160%', 'condensed', 'semi-expanded' и т. д.	Растягивает шрифт вместо подбора подходящего вытянутого варианта
fontStyle	Определяется CSS	'normal', 'italic', 'oblique'	Стиль шрифта
fontWeight	Определяется CSS	100-900, 'normal', 'bold'	Начертание шрифта
forceHitArea	false	true, false	Фиксирует доступную для щелчка область элемента для Internet Explorer
hover	false	true, false, Object	Разрешает обработку события наведения на элемент
hoverables	{ a: true }	{ tag: true, .. }	Определяет, к каким тегам будет применяться свойство hover
ignoreClass	null	'nocufon', 'nocufon skipcufon'	Позволяет отменить эамену шрифта внутри тегов, которые имеют перечисленные классы
letterSpacing	Определяется CSS	'-1px', ..	Расстояние между буквами
modifyText	null	function	Возвращающая новую строку функция, которая срабатывает перед тем, как фрагмент текста будет заменен впервые
onAfterReplace	null	function	Функция возврата, которая срабатывает после того, как элемент был заменен. Получает два аргумента: элемент и свойство
onBeforeReplace	null	function	Функция возврата, которая срабатывает перед тем, как элемент будет заменен. Получает два аргумента: элемент и свойство

Свойство	Значение по умолчанию	Значения	Описание
separate	'words'	'words', 'none' или 'characters'	Определяет, как Cufon будет разделять фрагменты текста на отдельные единицы
textShadow	'none'	'1px 1px #000', ...	Создает тень под текстом
textTransform	Определяется CSS	'uppercase', 'lowercase', 'capitalize', 'none'	Преобразует регистр текста
trim	'advanced'	'simple', 'advanced'	Значение 'simple' удаляет все предшествующие и последующие пробелы в смежных текстовых узлах, значение 'advanced' также влияет на смежные пробелы и переносы строк

Верстаем макет: меню

Продолжаем изучать верстку на примере нашего PSD-макета. На этот раз попробуем создать точно такие же горизонтальное и вертикальное меню, как в нашем макете (рис. 2.60).

Рис. 2.60. Варианты меню в нашем макете

Дополнительное горизонтальное меню

Начнем с верхней части. Меню с пунктами Новости, Статьи, Вопросы и ответы, Контакты на первый взгляд никаких сложностей не сулит. Но есть и в нем свои подводные камни.

В разделе «Верстаем горизонтальное меню» данной главы мы уже верстали дополнительное горизонтальное меню. Используем наработки из этого раздела. Напомню, что тегу `ul` нашего меню мы присвоили идентификатор `primary`:

```
ul#primary li{
    display: block; /* преобразуем в блочный элемент */
    float: left; /* преобразуем в плавающий элемент */
    list-style-type: none; /* удаляем маркер */
    list-style-image: none; /* удаляем маркер */
    padding: 0 7px; /* расстояние между пунктами меню */
}
```

Теперь нужно настроить параметры текста в меню. Для этого CSS-свойства необходимо применять к тегу `a`, а не к тегу `li`:

```
ul#primary li a{
    color: #d96729; /* цвет пунктов меню */
    font-size: 10pt; /* размер шрифта */
    font-weight: bold; /* полужирное начертание */
    font-family: Tahoma, Arial, sans-serif; /* имя шрифта */
    text-transform: uppercase; /* верхний регистр */
    text-decoration: none; /* убираем подчеркивание */
}
```

Осталось только вставить маркер. Причем не простой маркер, а фоновую картинку в виде маркера.

Для создания маркера достаточно добавить дополнительное правило `background: transparent url('../img/marker.png') no-repeat left center;`

селектору `ul#primary li`, после чего изменить значение свойства `padding` этого же селектора. Ведь слева от пункта меню у нас появилась картинка маркера, а значит, количество пустого места слева от пункта нужно увеличить, чтобы маркер не наезжал на текст. Присвоим свойству `padding` значение `0 5px 0 11px`.

В результате у нас получилось нечто напоминающее PSD-макет (рис. 2.61, *вверху*). Но есть одно бросающееся в глаза отличие — на макете перед первым пунктом меню маркера нет, а у нас есть.

Здесь нам поможет псевдокласс. Из всех рассмотренных нами в начале данной главы псевдоклассов лучше всего подойдет `first-child`. Именно он позволяет применить свойства CSS только к первому тегу в наборе тегов родительского элемента. А нам как раз и нужно изменить отображение только для первого тега `li` среди всех тегов `li` родительского элемента `ul`:

```
ul#primary li:first-child{

    background-image: none; /* удаляем маркер */

}
```

Кажется, работает (рис. 2.61, *внизу*). Но, к сожалению, только не в браузере Internet Explorer 6.

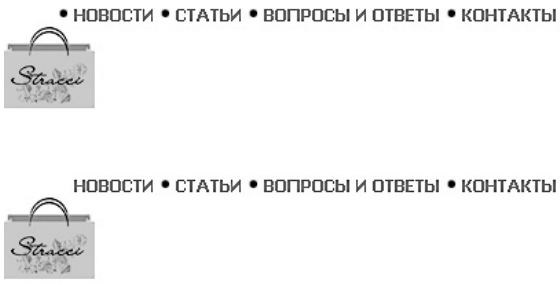


Рис. 2.61. Создаем дополнительное горизонтальное меню

Чтобы убрать маркер для первого тега `li` в браузере Internet Explorer 6, нужно воспользоваться следующей конструкцией:

```
* html ul#primary li:first-child{

    background-image: expression(this.previousSibling==null?'
none':'url(../img/marker.png)'); /* удаляем маркер */

}
```

Главное меню

Переходим к главному меню, которое является горизонтальным. Значит, для его создания нужно использовать тот же метод, что и для дополнительного горизонтального меню:

```
ul#firstmenu li{
    display: block; /* преобразуем в блочный элемент */
    float: left; /* преобразуем в плавающий элемент */
    list-style-type: none; /* удаляем маркер */
    list-style-image: none; /* удаляем маркер */
    padding: 0; /* расстояние между пунктами меню */
}
```

Кстати, главное меню имеет идентификатор `firstmenu`.

Все пункты главного меню имеют одинаковую ширину — 155 пикселов и высоту — 32 пиксела. Текст в них центрирован, а расстояние между пунктами меню — 14 пикселов. Таким образом, добавим селектору `ul#firstmenu li` следующие свойства:

```
    width: 155px; /* ширина пункта меню */
    height: 32px; /* высота без нижней границы */
    text-align: center; /* центрируем содержимое */
    margin: 0 0 0 14px; /* расстояние между пунктами */
```

Для первого в списке тега `li` следует удалить отступ:

```
ul#firstmenu li:first-child{
    margin: 0;
}
* html ul#firstmenu li:first-child{
     margin: expression(this.previousSibling==null?'0':'0 0 0
14px');
}
```

Теперь займемся фоном. Как видно на рис. 2.60, в качестве фона у нас выступает градиент, который внизу заканчивается сплошной одноцветной линией высотой 3 пиксела.

Одноцветную линию можно сделать границей. Соответственно, и высоту пункта меню нужно указать без высоты границы (так мы и сделали).

Градиент делаем картинкой шириной 1 пиксел.

Итак, добавим селектору `ul#firstmenu li` еще несколько свойств:

```
    background: #c9921e url('../img/bg_li.png') repeat-x left
bottom;

    border-bottom: 3px solid #887f6e;
```

Теперь можно переходить к тексту пунктов меню:

```
ul#firstmenu li a{

    color: #000; /* цвет пунктов меню */

    text-decoration: none; /* убираем подчеркивание */

    font-size: 9pt; /* размер шрифта */

}
```

Что же у нас получилось (рис. 2.62)?

Рис. 2.62. Главное меню

В принципе, неплохо. Только тегу `li` нужно добавить верхний `padding`, чтобы сместить текст пункта меню ниже:

```
ul#firstmenu li{

    padding: 3px 0 0 0; /* расстояние между пунктами меню */

    height: 29px; /* высота без нижней границы */

}
```

Необходимо также заменить шрифт главного меню нестандартным AvantGarde-GothicC:

```
Cufon.replace('ul#firstmenu li a', { fontFamily: 'AvantGardeGothicC',
fontSize: '13pt' });
```

Теперь главное меню готово.

Изучаем свойства: таблицы

В CSS есть специальные свойства, которые работают только для таблиц.

vertical-align

Основные сведения:

- значение по умолчанию: `baseline`;
- наследуется: нет;
- применяется: к встроенным элементам или ячейкам таблицы;
- версия CSS: 1;
- синтаксис: `vertical-align: baseline|bottom|middle|sub|super|text-bottom|text-top|top|inherit | значение | проценты`.

Содержимое колонок таблицы можно выравнивать не только по горизонтали, но и по вертикали. Это делается с помощью свойства `vertical-align`, которое может принимать следующие значения:

- `baseline` — выровнять базовую линию текущего элемента по базовой линии родителя;
- `bottom` — выровнять основание текущего элемента по нижней части элемента строки, расположенного ниже всех;
- `middle` — выровнять среднюю точку элемента по базовой линии родителя плюс половина высоты родительского элемента;
- `sub` — выровнять элемент так, словно он подстрочный (в виде нижнего индекса);
- `super` — выровнять элемент так, словно он надстрочный (в виде верхнего индекса);
- `text-bottom` — выровнять нижнюю границу элемента по самому нижнему краю текущей строки;
- `text-top` — выровнять границу элемента по самому высокому текстовому элементу текущей строки;
- `top` — выровнять верхний край элемента по верху самого высокого элемента строки.

border-collapse

Основные сведения:

- значение по умолчанию: `separate`;
- наследуется: нет;
- применяется: к тегу `<table>` или к элементам, у которых значение `display` установлено как `table` или `inline-table`;
- версия CSS: 2;
- синтаксис: `border-collapse: collapse | separate | inherit`.

Благодаря данному свойству, которое применяется к тегу `table`, можно убрать двойные границы вокруг колонок таблицы. Свойство поддерживает следующие значения:

- `collapse` — объединяет смежные границы колонок в одну (рис. 2.63, *слева*);
- `separate` — отображает смежные границы колонок отдельно друг от друга (рис. 2.63, *справа*).

Рис. 2.63. Свойство border-collapse

border-spacing

Основные сведения:

- значение по умолчанию: 0;
- наследуется: да;
- применяется: к таблицам;
- версия CSS: 2;
- синтаксис: `border-spacing: значение1 [значение2]`.

Если свойству `border-collapse` задано значение `separate`, с помощью свойства `border-spacing` можно указать расстояние между границами ячеек таблицы.

Если в свойстве `border-spacing` задано два значения через пробел, первое определяет расстояние по горизонтали, а второе — по вертикали.

caption-side

Основные сведения:

- значение по умолчанию: `top`;
- наследуется: да;
- применяется: к `<caption>` или ко всем элементам, у которых значение `display` установлено как `table-caption`;
- версия CSS: 2;
- синтаксис:
  - CSS2.1: `caption-side: top | bottom`;
  - CSS3: `caption-side: top | bottom | left | right`.

Данное свойство позволяет определить положение заголовка таблицы (содержимое тега `caption`) относительно самой таблицы. В качестве значений данного свойства могут выступать следующие ключевые слова:

- `top` — расположить заголовок по верхнему краю таблицы;
- `bottom` — разместить заголовок по нижнему краю таблицы.

empty-cells

Основные сведения:

- значение по умолчанию: `show`;
- наследуется: да;
- применяется: к `<td>`, `<th>` или к элементам, у которых `display: table-cell`;
- версия CSS: 2;
- синтаксис: `empty-cells: show | hide`.

С помощью данного свойства можно изменить способ отображения пустых ячеек таблицы. Оно принимает следующие значения:

- `show` — отобразить границу и фон пустой ячейки;
- `hide` — скрыть границу и фон пустой ячейки либо всю строку, если в ней все ячейки пустые.

Под пустыми ячейками таблицы понимаются следующие случаи:

- в ячейке вообще нет содержимого;
- в ячейке находятся только пробел, символ табуляции или перевод строки;
- свойству `visibility` ячейки присвоено значение `hidden`.

Данное свойство не поддерживается версиями 6 и 7 браузера Internet Explorer. В них пустые ячейки всегда отображаются так, словно для них установлено правило `empty-cells: hide;`.

table-layout

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к тегу `<table>` или к элементу, у которого значение `display` установлено как `table` или `inline-table`;
- версия CSS: 2;
- синтаксис: `table-layout: auto | fixed | inherit`.

Данное свойство задает способ вычисления ширины таблицы. Оно поддерживает следующие значения:

- `auto` — ширина колонок таблицы определяется браузером после анализа их содержимого;
- `fixed` — ширина колонок задается тегом `col` либо на основе содержимого первой строки таблицы. Если по каким-то причинам определить ширину колонок не удается, таблица делится на колонки равной ширины.

Изучаем свойства: печать

Среди свойств CSS можно найти и такие, действие которых заметно только после распечатки веб-страницы на принтере. В данном разделе кратко рассмотрены такие свойства.

orphans

Основные сведения:

- значение по умолчанию: 2;
- наследуется: да;
- применяется: к блочным элементам;
- версия CSS: 2;
- синтаксис: `orphans: число | inherit`.

С помощью данного свойства можно задать правило для так называемой висячей строки, то есть свойство определяет минимальное количество строк, которое может оставаться на предыдущей странице при печати документа. Данное свойство применяется только в том случае, если текст HTML-документа занимает две и более страницы.

widows

Основные сведения:

- значение по умолчанию: 2;
- наследуется: да;
- применяется: к блочным элементам;
- версия CSS: 2;
- синтаксис: `widows: число | inherit`.

Данное свойство определяет минимальное количество строк, которое может оставаться на следующей странице при печати документа. Данное свойство применяется только в том случае, если текст HTML-документа занимает две и более страницы.

В конфликтных ситуациях свойство `widows` имеет приоритет над свойством `orphans`.

page-break-inside

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;

- применяется: к блочным элементам;
- версия CSS: 2;
- синтаксис: `page-break-inside: auto | avoid | inherit`.

Позволяет запретить либо разрешить разрыв страницы внутри элемента при печати.

page-break-before

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к блочным элементам;
- версия CSS: 2;
- синтаксис: `page-break-before: always | auto | avoid | left | right | inherit`.

Задает разрыв страницы перед указанным элементом при печати. Возможные значения:

- `always` — всегда добавлять разрыв страницы;
- `auto` — вставлять разрыв страницы при необходимости;
- `avoid` — запретить вставку разрыва страницы;
- `left` — пропустить одну или две страницы, чтобы следующая страница при печати была четной;
- `right` — пропустить одну или две страницы, чтобы следующая страница при печати была нечетной;
- `inherit` — наследовать значение.

В 6-й и 7-й версиях браузера Internet Explorer значения `left`, `right` и `inherit` не поддерживаются.

page-break-after

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к блочным элементам;

- версия CSS: 2;
- синтаксис: `page-break-after: always | auto | avoid | left | right | inherit`.

Задает разрыв страницы после заданного элемента при печати. Значения данного свойства подобны тем, которые используются свойством `page-break-before`.

В 6-й и 7-й версиях браузера Internet Explorer значения `left`, `right` и `inherit` не поддерживаются.

Изучаем свойства: другое

Мы практически закончили изучать свойства CSS, доступные в 1-й и 2-й версиях данного языка. Осталось лишь узнать о последнем свойстве, которое вы сможете применять в своих проектах.

cursor

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: да;
- применяется: ко всем элементам;
- версия CSS: 2;
- синтаксис: `cursor: [url('путь к курсору'),] | [ auto | crosshair | default | e-resize | help | move | n-resize | ne-resize | nw-resize | pointer | progress | s-resize | se-resize | sw-resize | text | w-resize | wait | inherit ]`.

Данное свойство позволяет указать, какой формы станет указатель мыши, если вы расположите его в пределах тега, к которому применяется данное свойство. В качестве значения свойства `cursor` можно использовать следующие ключевые слова:

- `crosshair` — перекрестье;
- `default` — стрелочка;
- `e-resize` — стрелка вправо;
- `help` — вопросительный знак;
- `move` — стрелка перемещения объекта;

- `n-resize` — стрелка вверх;
- `ne-resize` — стрелка, направленная вправо и вверх;
- `nw-resize` — стрелка, направленная влево и вверх;
- `pointer` — стрелка в виде руки;
- `progress` — стрелка и песочные часы;
- `s-resize` — стрелка вниз;
- `se-resize` — стрелка направленная вправо и вниз;
- `sw-resize` — стрелка, направленная влево и вниз;
- `text` — каретка;
- `w-resize` — стрелка влево;
- `wait` — песочные часы.

Свойство `cursor` позволяет также задать путь к картинке, которая будет использоваться в качестве указателя. Однако браузер Opera не поддерживает такую возможность. В остальных браузерах форматы изображений, которые могут использоваться в качестве указателей, различаются:

- Internet Explorer — CUR и ANI;
- Mozilla Firefox, Google Chrome, Safari — CUR, PNG, GIF, JPG.

Дизайн прокрутки

Если содержимое страницы не помещается на экране, то отображается горизонтальная и/или вертикальная прокрутка, с помощью которой можно посмотреть те части страницы, которые не поместились на экране.

Дизайнеры довольно часто пытаются изменить стиль полосы прокрутки если не браузера, то определенного элемента внутри страницы (рис. 2.64). Сейчас мы попробуем удовлетворить всех дизайнеров, которые рисуют свою полосу прокрутки для определенного элемента.

Свойств CSS, которые меняли бы стиль полосы прокрутки для всех браузеров, не существует. Есть нестандартные свойства, которые работают в конкретном браузере и позволяют изменить цвет отдельной части полосы прокрутки. Но даже с их помощью нельзя изменить полосу прокрутки так, чтобы она была похожа на показанную на рис. 2.64.

В данном случае на помощь приходит JavaScript. Существует специальный сценарий, позволяющий создавать полосы прокрутки самого разнообразного вида — `jScrollPane.js`. Данный сценарий работает на основе jQuery, подробнее мы рассмотрим его в главе 3, посвященной JavaScript.

Рис. 2.64. Измененная полоса прокрутки[1]

Нововведения CSS3

До сих пор мы рассматривали возможности CSS1 и CSS2. Но уже существует новая версия CSS — CSS3. Правда, нововведения CSS3 поддерживаются только самыми последними версиями браузеров, да и то не все. Тем не менее коротко рассмотрим, какие дополнительные возможности открывает новая версия CSS.

Новые свойства

В CSS3 добавилось несколько десятков новых свойств. Рассмотрим некоторые из них.

opacity

Основные сведения:

- значение по умолчанию: 1;
- наследуется: нет;
- применяется: ко всем элементам;
- синтаксис: `opacity: значение`.

[1] Сайт http://moigeroi.com/ru/catalog/2.

Позволяет изменить уровень прозрачности тега в диапазоне от 0 до 1.

Браузер Internet Explorer для изменения прозрачности использует фильтры. Например, `filter: alpha(opacity=50)`, где параметр `opacity` может принимать значение от 0 до 100.

background-clip

Основные сведения:

- значение по умолчанию: `border-box`;
- наследуется: нет;
- применяется: ко всем элементам;
- синтаксис: `background-clip: [padding-box | border-box | content-box] [, [padding-box | border-box | content-box]]`.

Изменяет способ вывода фона или фоновой картинки элемента под границей тега (пунктирной или прозрачной). Возможные значения:

- `padding-box` — фон не отображается под границами;
- `border-box` — фон отображается под границами;
- `content-box` — фон отображается только внутри контента.

background-origin

Основные сведения:

- значение по умолчанию: `padding-box`;
- наследуется: нет;
- применяется: ко всем элементам;
- синтаксис: `background-origin: [padding-box | border-box | content-box] [, [padding-box | border-box | content-box]]*`.

Изменяет область позиционирования фонового рисунка, то есть позволяет определить начальную точку, относительно которой свойство `background` определяет координаты расположения фоновой картинки.

Возможные значения:

- `padding-box` — фон позиционируется относительно границы, но не заходит под саму границу;

- `border-box` — фон позиционируется относительно границы и может перекрываться ею;
- `content-box` — фон позиционируется относительно содержимого элемента.

background-size

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: ко всем элементам;
- синтаксис: `background-size: [<значение> | <проценты> | auto ]{1,2} | cover | contain`.

Позволяет масштабировать фоновое изображение. Помимо единиц измерений, фактически задающих размер картинки, свойство поддерживает следующие ключевые слова:

- `cover` — позволяет масштабировать изображение по меньшей стороне с сохранением пропорций (чтобы ширина/высота изображения равнялись ширине/высоте блока);
- `contain` — дает возможность масштабировать изображение по большей стороне с сохранением пропорций (чтобы ширина/высота изображения равнялись ширине/высоте блока и изображение полностью поместилось внутри блока).

border-image

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: нет;
- применяется: ко всем элементам, за исключением тех, у которых `border-collapse` задан как `collapse`;
- синтаксис: `border-image: none | [ <URL> [<число> | <проценты>]{1,4} [/ <толщина>{1,4}]? ] && [stretch | repeat | round]{0,2}`.

С помощью данного свойства можно создавать рамки из любого изображения. Достаточно указать ссылку на изображение в качестве значения данного свойства.

border-radius

Основные сведения:

- значение по умолчанию: 0;
- наследуется: нет;
- применяется: ко всем элементам, за исключением таблиц с `border-collapse: collapse`;
- синтаксис: `border-radius: <радиус>{1,4}`.

Мы уже встречались с этим свойством. Оно позволяет скруглить рамку вокруг элемента.

box-shadow

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: нет;
- применяется: ко всем элементам;
- синтаксис: `box-shadow: none | <тень> [,<тень>]*`, где `<тень>`:

```
inset <сдвиг по x> <сдвиг по y> <радиус размытия> <растяжение> <цвет>
```

С данным свойством мы также встречались. С его помощью можно создать тень под элементом.

box-sizing

Основные сведения:

- значение по умолчанию: `content-box`;
- наследуется: нет;
- применяется: ко всем элементам;
- синтаксис: `box-sizing: content-box | border-box | inherit`.

Позволяет изменить алгоритм расчета размеров элемента. Возможные значения:

- `border-box` — ширина элемента включает в себя ширину `padding` и `border`;
- `content-box` — значение по умолчанию, при котором ширина элемента не включает в себя ни `border`, ни `padding`, ни `margin`.

column-count

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к блочным элементам (кроме таблиц), ячейкам и элементам, у которых `display` установлен как `inline-block`;
- синтаксис: `column-count: <число> | auto`.

Определяет количество колонок в многоколоночном тексте.

column-gap

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: нет;
- применяется: к блочным элементам (кроме таблиц), ячейкам и элементам, у которых `display` установлен как `inline-block`;
- синтаксис: `column-gap: <значение> | normal`.

Позволяет задать расстояние между колонками в многоколоночном тексте.

column-rule

Основные сведения:

- значение по умолчанию: `medium none`;
- наследуется: нет;
- применяется: к блочным элементам (кроме таблиц), ячейкам и элементам, у которых `display` установлен как `inline-block`;
- синтаксис: `column-rule: <border-width> || <border-style> || <цвет>`.

Позволяет задать границу между колонками. Значение данного свойства полностью аналогично значению свойства `border`.

column-width

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;

- применяется: к блочным элементам (кроме таблиц), ячейкам и элементам, у которых `display` установлен как `inline-block`;
- синтаксис: `column-width: <значение> | auto`.

Определяет рекомендуемую ширину колонки (в зависимости от обстоятельств, ширина колонки может быть больше рекомендуемой).

columns

Основные сведения:

- значение по умолчанию: `auto`;
- наследуется: нет;
- применяется: к блочным элементам (кроме таблиц), ячейкам и элементам, у которых `display` установлен как `inline-block`;
- синтаксис: `columns: [column-width] || [column-count]`.

Универсальное свойство, заменяющее собой рассмотренные нами свойства `column-width` и `column-count`.

font-stretch

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;
- применяется: ко всем элементам;
- синтаксис: `font-stretch: inherit | ultra-condensed | extra-condensed | condensed | semi-condensed | normal | semi-expanded | expanded | extra-expanded | ultra-expanded`.

Позволяет изменить начертание шрифта с нормального на узкое или широкое. Значения `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed` в разной степени сужают символы шрифта. Значения `ultra-expanded`, `extra-expanded`, `expanded`, `semi-expanded` в различной степени расширяют символы шрифта.

resize

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: нет;

- применяется: к `<textarea>`;
- синтаксис: `resize: none | both | horizontal | vertical | inherit`.

Позволяет создавать текстовые поля с изменяемыми пользователем размерами, то есть посетитель сможет изменить ширину или высоту такого поля, просто потянув за его границу.

Возможные значения:

- `none` — отменить возможность изменения размеров пользователем;
- `both` — разрешить изменение размеров по горизонтали и вертикали;
- `horizontal` — разрешить изменение размеров только по горизонтали;
- `vertical` — разрешить изменение размеров только по вертикали.

tab-size

Основные сведения:

- значение по умолчанию: 8;
- наследуется: да;
- применяется: к блочным элементам;
- синтаксис: `tab-size: <число>`.

Позволяет изменить ширину символа табуляции.

text-align-last

Основные сведения:

- значение по умолчанию: `start`;
- наследуется: да;
- применяется: к блочным элементам;
- синтаксис: `text-align-last: start | end | left | right | center | justify`.

Задает выравнивание последней строки текста. Возможные значения выравнивания:

- `start` — по начальному краю блока;
- `end` — по конечному краю блока;
- `left` — по левому краю;
- `right` — по правому краю;

- `center` — по центру;
- `justify` — по ширине (если последняя строка абзаца состоит из одного слова, это слово выравнивается по левому краю).

text-overflow

Основные сведения:

- значение по умолчанию: `clip`;
- наследуется: нет;
- применяется: к блочным элементам;
- синтаксис: `text-overflow: clip | ellipsis`.

Определяет, что будет происходить с текстом, если он не помещается внутри тега. Возможные значения:

- `clip` — обрезать текст по размеру области;
- `ellipsis` — обрезать текст по размеру области и добавить многоточие в конец текста.

text-shadow

Основные сведения:

- значение по умолчанию: `none`;
- наследуется: да;
- применяется: ко всем элементам;
- синтаксис: `text-shadow : none | тень [,тень]*`, где `тень`:

```
<цвет> <сдвиг по x> <сдвиг по y> <радиус размытия>
```

Позволяет задать тень под текстом.

word-wrap

Основные сведения:

- значение по умолчанию: `normal`;
- наследуется: да;
- применяется: ко всем элементам;
- синтаксис: `word-wrap: normal | break-word | inherit`.

Определяет, нужно ли переносить длинные слова, которые не помещаются по ширине в заданную область.

writing-mode

Основные сведения:

- значение по умолчанию: нет;
- наследуется: да;
- применяется: ко всем элементам и генерируемому контенту;
- синтаксис: `writing-mode: lr-tb | rl-tb | tb-rl | bt-rl | tb-lr | bt-lr`.

Позволяет задать направление текста на странице. Возможные значения:

- `lr-tb` — слева направо;
- `rl-tb` — справа налево;
- `tb-rl` — вертикально и выровнять по верхнему и правому краям;
- `bt-rl` — вертикально и выровнять по нижнему и правому краям;
- `tb-lr` — вертикально и выровнять по верхнему и левому краям;
- `bt-lr` — вертикально и выровнять по нижнему и левому краям.

Новые значения

Изменились и некоторые существовавшие ранее свойства CSS. Они стали поддерживать дополнительные значения.

text-align

Список значений данного свойства дополнился значениями `start` и `end`, то есть значения свойства `text-align` аналогичны значениям рассмотренного ранее свойства `text-align-last`.

background

В CSS3 появилась возможность задавать сразу несколько фоновых картинок элементу. Для этого достаточно описать все фоновые картинки через запятую в этом свойстве.

Например, `background: url(images/onepict.png) repeat-y, #ccc url(images/twopict.png) repeat-y 100% 0;`.

background-attachment

Поскольку в CSS3 появилась возможность в свойстве `background` задавать сразу несколько фоновых картинок, свойство `background-attachment` также стало поддерживать возможность указания сразу нескольких значений через запятую — по одному на каждую фоновую картинку.

caption-side

В CSS3 появились дополнительные значения `left` и `right`. С их помощью заголовок можно разместить слева или справа от таблицы.

Глава 3

JavaScript, jQuery, Ajax

Создаем и подключаем сценарии

Ошибки в JavaScript

Синтаксис JavaScript

Работаем с DOM

Работа с jQuery

Готовые сценарии

В современном мире верстальщику, который не знает хотя бы основ JavaScript, будет очень сложно. Совершенно не обязательно быть гуру JavaScript, но основами данного языка владеть необходимо. Достаточно уметь подключать и использовать сторонние сценарии, а также иметь представление о jQuery и уметь работать с его базовыми возможностями. Этого хватит, чтобы выполнять серьезные заказы по верстке. И в данной главе мы рассмотрим эти вопросы.

Прежде всего нужно запомнить, что JavaScript является регистрозависимым языком. По этой причине следует запоминать не только названия свойств, методов или ключевых слов JavaScript, но и то, как они пишутся.

Создаем и подключаем сценарии

Код JavaScript может быть написан либо непосредственно в самом HTML-документе, либо в отдельном файле.

В HTML-документе

Для записи JavaScript-кода используется тег `script`. Желательно указывать его внутри тега `head`. Хотя и внутри тега `body` он будет прекрасно работать.

Синтаксис тега `script` следующий:

```
<script type="text/javascript">

    Код JavaScript

</script>
```

Как видите, ничего сложного пока нет.

В JS-файле

JavaScript-код можно также записывать в отдельном файле с расширением JS, после чего подключать его к HTML-документу примерно так, как мы делали это с CSS-файлами.

Файл с расширением JS является обычными текстовым файлом, как и другие уже известные нам файлы: CSS- и HTML-файлы. JS-файлы принято размещать в папке `js`. Создадим пустой файл `addons.js`, после чего подключим его к нашему шаблону.

Для подключения JS-файлов также используется тег `script`. Однако, в отличие от подключения CSS-документа, на этот раз следует использовать другой синтаксис:

```
    <script type="text/javascript" src="путь к JS-файлу"> </
script>
```

Теги `script`, которыми подключаются JS-файлы, могут находиться только внутри тега `head`. Внутри тега `body` их указывать нельзя.

События

В HTML существует возможность назначать код JavaScript каким-либо событиям. В этом случае JavaScript-код будет выполняться только при наступлении определенного события.

Событие в HTML — это некоторое действие (щелчок кнопкой мыши, перемещение указателя, нажатие клавиши и т.д.), производимое пользователем на веб-странице либо выполняемое элементами веб-страницы. События относятся только к тому тегу, которому они назначены. Например, если вы назначили событие щелчка кнопкой мыши тегу с идентификатором `header`, то оно наступит только в том случае, если посетитель щелкнет на теге с этим идентификатором.

Когда событие наступает, браузер вызывает обработчик события — объект JavaScript, выполняющий определенный код.

Существует четыре основных группы обработчиков событий:

- обработчики событий окна — выполняют какие-либо действия при открытии, закрытии, изменении размеров или перемещении окна браузера;
- обработчики событий мыши — производят какие-либо действия при обычном или двойном щелчке кнопкой мыши, а также при перемещениях указателя мыши;
- обработчики событий клавиатуры — выполняют какие-либо действия при нажатии клавиш клавиатуры;
- обработчики событий формы — производят какие-либо действия при изменении состояния элементов формы, а также при сбросе формы либо отправке ее на сервер.

Чтобы назначить JavaScript-код какому-либо событию (чтобы код выполнялся при наступлении данного события), достаточно воспользоваться специальными атри-

бутами тегов. Список атрибутов, которые можно использовать, зависит от тега. Но все такие атрибуты начинаются с префикса on. Например: <body onload="команды JavaScript">.

Чаще всего событиям назначают не сам JavaScript-код, а вызов функции, которая выполняла бы нужный JavaScript-код. Саму же функцию создают в отдельном JS-файле либо в теге script HTML-документа.

Обработчики событий окна

Как правило, обработчики событий окна применяются к тегу body, представляющему в HTML окно браузера, либо к тегу для создания фреймов (frameset).

Существуют следующие обработчики событий окна:

- onabort — если загрузка объекта прервана;
- onblur — если элемент потерял фокус;
- onerror — если возникла ошибка при загрузке веб-страницы или объекта;
- onfocus — если элемент получил фокус (стал активным в данный момент);
- onload — если загрузка объекта окончена;
- onmove — если окно или кадр перемещены;
- onresize — если размер окна/кадра изменился;
- onunload — если посетитель закрывает веб-страницу.

Обработчики событий для работы с фокусом (onblur и onfocus) дополнительно могут применяться к тегам a, abbr, acronym, address, applet, area, b, basefont, bdo bgsound, big, blockquote, body, br, button, caption, center, cite, code, col, colgroup, dd, del, dfn, dir, div, dl, dt, em, embed, fieldset, font, form, frame, h1, h2, h3, h4, h5, h6, hr, i, iframe, img, input, ins, isindex, kbd, label, legend, li, link, map, marquee, menu, nobr, object, ol, option, p, plaintext, pre, q, s, samp, select, small, span, strike, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, u, ul, var, wbr, xmp.

Обработчики событий мыши

Все обработчики событий мыши могут применяться к тегам a, abbr, acronym, address, applet, area, b, basefont, bdo, bgsound, big, blockquote, body, br, button, caption, center, cite, code, col, colgroup, dd, del, dfn, dir, div, dl, dt, em, embed, fieldset, font, form, frame, h1, h2, h3, h4, h5, h6, hr, i, iframe, img, input, ins, isindex, kbd, label, legend, li, link, map, marquee,

menu, nobr, object, ol, option, p, plaintext, pre, q, s, samp, select, small, span, strike, strong, sub, sup, table, tbody, td, textarea, tfoot, th, thead, tr, tt, u, ul, var, wbr, xmp.

Существуют следующие обработчики событий мыши:

- onclick — если на элементе щелкнули левой кнопкой мыши;
- ondblclick — если на элементе дважды щелкнули левой кнопкой мыши;
- ondragdrop — если объект был перемещен в область окна;
- onmousedown — если посетитель удерживает левую кнопку мыши над объектом;
- onmousemove — если посетитель перемещает указатель мыши над объектом;
- onmouseout — если указатель мыши вышел за пределы объекта;
- onmouseover — если указатель мыши вошел в пределы объекта;
- onmouseup — если посетитель отпустил левую кнопку мыши.

Интересна ситуация, возникающая, если вы назначаете обработчик onclick ссылке. В этом случае при нажатии ссылки сначала выполняется JS-код, назначенный событию onclick, а уже потом производится переход по ссылке. При этом вы вообще можете отменить переход по указанной ссылке. Для этого достаточно из события onclick (функции, вызываемой этим событием) вернуть значение false. Конечно, пока это вам ни о чем не говорит, но данный код отменяет переход по указанной ссылке: onclick="return false;".

Иногда возникает необходимость создать ссылку, которая вообще никуда не ведет, например, чтобы реализовать функционал такой ссылки только на JavaScript. Это можно сделать следующим образом:

```
<a href="javascript:void(0);" onclick="команды">название</a>
```

Обработчики событий клавиатуры

Все обработчики событий клавиатуры могут применяться к тем же тегам, что и обработчики событий мыши.

Существуют следующие обработчики событий клавиатуры:

- onkeydown — если посетитель нажал любую клавишу;
- onkeypress — если посетитель нажал и удерживает клавишу;
- onkeyup — если посетитель отпустил нажатую ранее клавишу.

Обработчики событий формы

Существуют следующие обработчики событий формы:

- `onsubmit` — если посетитель отправил форму (применяется к тегу `form`);
- `onselect` — если в форме выбрано поле (применяется к тегам `input`, `textarea`);
- `onreset` — если параметры формы были сброшены (применяется к тегу `form`);
- `onchange` — если в форме изменились данные (применяется к тегам `input`, `select`, `textarea`).

Ошибки в JavaScript

Как и любая деятельность человека, разработка сценариев JavaScript не обходится без возникновения ошибок. Поэтому не стоит бояться появления ошибок. Следует просто научиться определять и устранять их.

Не всегда легко на глаз выяснить наличие ошибок в сценарии. Не все браузеры явным образом сообщают, что в сценарии JavaScript есть ошибки. Собственно говоря, об этом сообщает только браузер Internet Explorer (рис. 3.1).

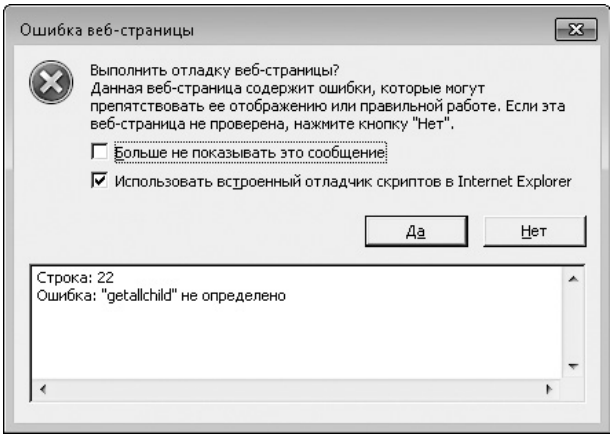


Рис. 3.1. Об ошибках в сценарии явным образом сообщает только браузер Internet Explorer

Если вы установили расширение Web Developer для браузера Mozilla Firefox, то о возникновении ошибок в сценариях будет также сообщать и он. Об этом сигнализирует наличие красного крестика в правом углу панели инструментов расширения Web Developer (рис. 3.2).

Рис. 3.2. Сигнализация об ошибках JavaScript в расширении Web Developer для браузера Mozilla Firefox

Но тем не менее именно браузеры являются нашими основными помощниками в поиске и устранении ошибок в JavaScript.

Любой современный браузер имеет инструменты для разработчика, с помощью которых можно в том числе увидеть количество ошибок в сценарии и кратко прочитать о возможных причинах возникновения этих ошибок.

- Mozilla Firefox. Пункт меню Инструменты ▸ Веб-разработка ▸ Консоль ошибок или сочетание клавиш Ctrl+Shift+J (рис. 3.3).

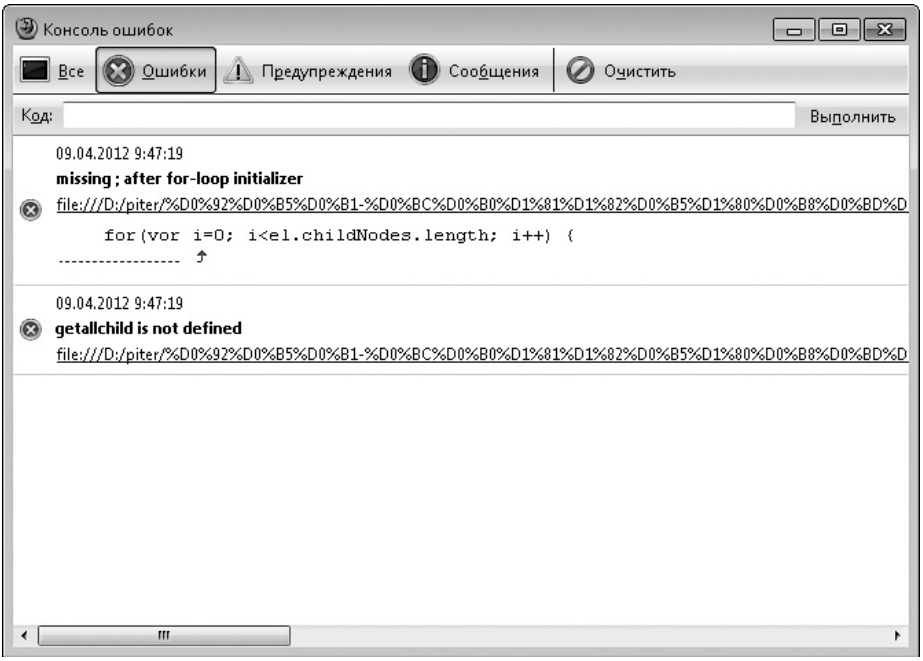


Рис. 3.3. Консоль ошибок в Mozilla Firefox

- Internet Explorer. В появившемся при открытии веб-страницы сообщении об ошибках нажмите кнопку Да (рис. 3.4).
- Opera. Пункт меню Страница ▸ Средства разработки ▸ Консоль ошибок или сочетание клавиш Ctrl+Shift+O (рис. 3.5).

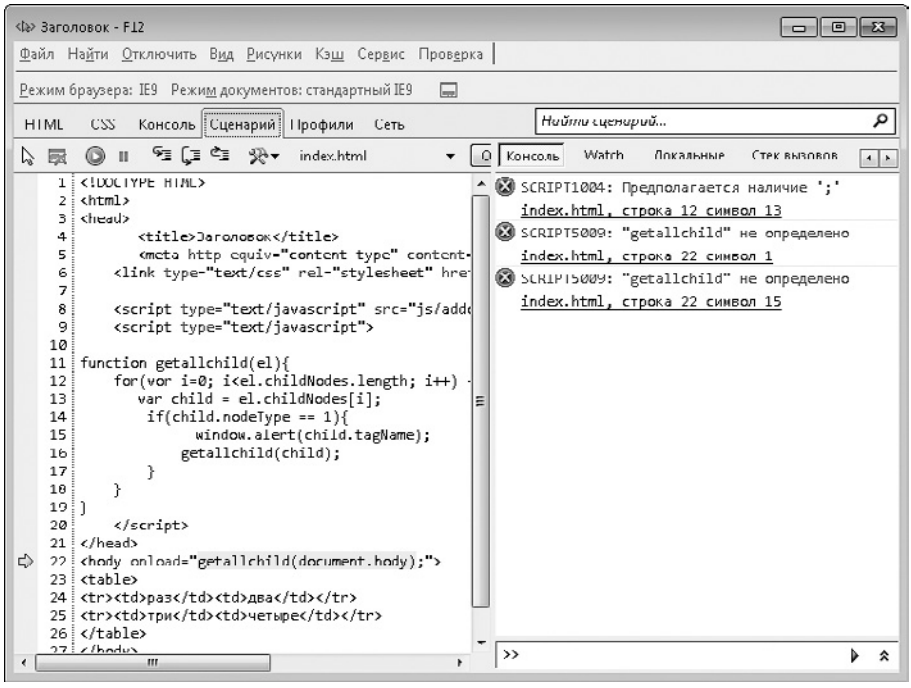


Рис. 3.4. Консоль ошибок в Internet Explorer

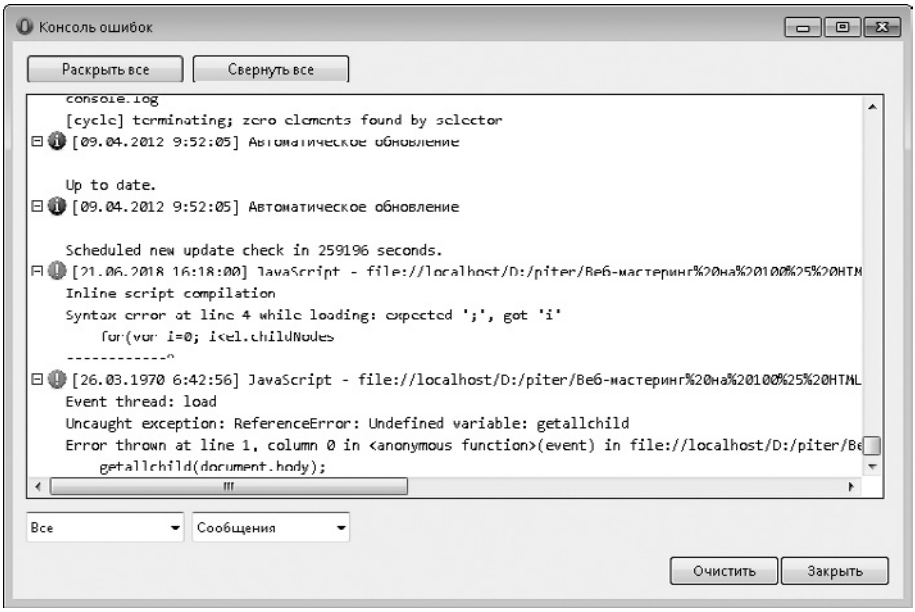


Рис. 3.5. Консоль ошибок в Opera

- Google Chrome. Пункт меню Инструменты ▸ Консоль JavaScript или сочетание клавиш Ctrl+Shift+J (рис. 3.6).

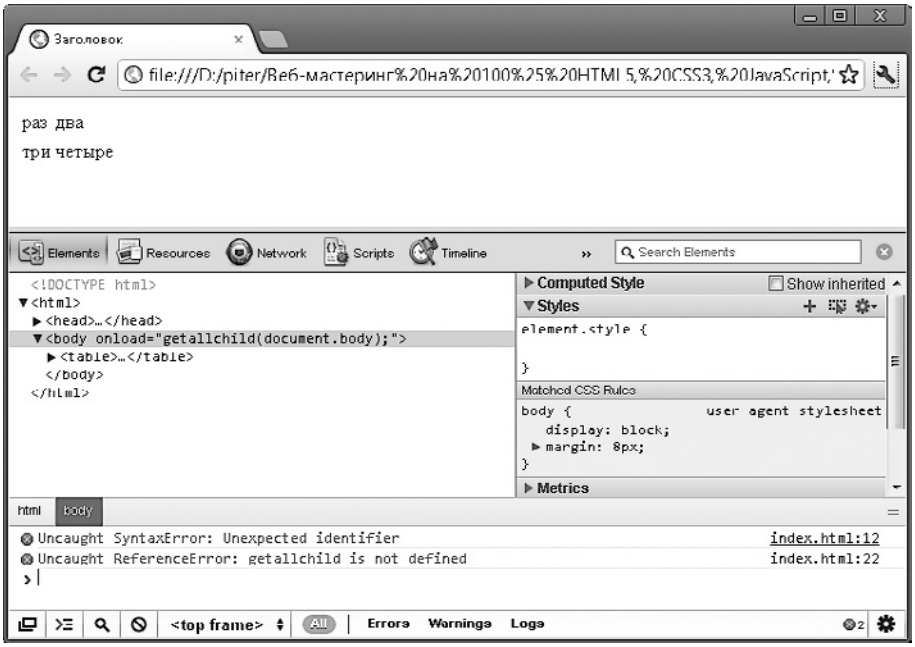


Рис. 3.6. Консоль ошибок в Google Chrome

В нашем примере ошибка заключалась лишь в строке:

```
for(vor i=0; i<el.childNodes.length; i++) {
```

Правильный вариант этой строки:

```
for(var i=0; i<el.childNodes.length; i++) {
```

Как обычно, браузер Internet Explorer показал на что угодно, но только не на истинную ошибку. По этой причине для отладки сценариев Internet Explorer лучше не использовать.

Синтаксис JavaScript

Подключать JS-код вы уже умеете. Осталось только узнать, как его создавать. Если вы уже изучали какой-нибудь язык программирования, то без труда освоите JavaScript.

Переменные

Переменной называется определенная ячейка в памяти компьютера, в которой хранятся какие-либо данные. У такой ячейки должно быть имя, иначе вы не сможете к ней обратиться, чтобы получить/изменить хранящиеся в ней данные.

Объявляем переменную

Перед тем как впервые использовать какую-либо переменную, ее желательно объявить, то есть задать имя этой переменной. Это делается следующим образом:

```
var имя;
```

В имени переменной могут использоваться символы латинского алфавита, а также числа. Однако начинаться имя переменной должно только с символа латинского алфавита.

Обратите также внимание на точку с запятой в конце строки JS-кода. Точка с запятой говорит о том, что команда закончилась. Дальше идет новая команда.

Если вы не укажете точку с запятой в конце какой-либо команды, JavaScript-код не будет работать.

И, конечно, не забывайте о регистрозависимости языка JavaScript. Объявления `var var1;` и `var Var1;` создают две разные переменные.

Объявляем переменную со значением

Представленным в предыдущем подразделе кодом мы просто объявили переменную без какого-либо значения, хранящегося в ней. Как правило, так не делается. Чаще всего переменную не только объявляют, но и сразу задают ей значение. Это делается в расширенной версии рассмотренной нами выше команды:

```
var имя = значение;
```

Значения переменных

В качестве значения переменной может выступать число, строка символов или какой-либо объект. Причем в одной и той же переменной может находиться как число, так и строка символов или объект. То есть переменные не разделяются по типу хранящейся в них информации.

Чтобы объявить переменную с численным значением, достаточно при ее создании указать нужное значение:

```
var myvar= 45;
```

Если значение должно быть строкой, необходимо взять его в кавычки:

```
var myvar= "45";

var myvar2= "testing...";
```

С одной стороны, это удобно. Не нужно заранее указывать тип переменной и следить, чтобы значение переменной всегда было строкой, числом или имело какой-либо другой тип. Но, с другой стороны, могут возникать некоторые сложности.

Например, действие оператора + отличается в зависимости от типа значений, к которым вы его применяете. Так, если данный оператор применяется к двум числам, то происходит сложение этих чисел. Но если оператор + применяется к числу и строке (или к двум строкам), то выполняется конкатенация строк, то есть их объединение. Например:

- `45+15=60 // число+число;`
- `45+"15"=4515 // число+строка.`

Комментарии

Комментарии есть даже в HTML и CSS, и было бы странно, если бы их не было в языке JavaScript. В JavaScript существует два способа задать комментарий.

Первый способ — использовать два слеша (`//`). С их помощью можно создать строку комментария. Все, что идет после `//` и до перехода на новую строку, будет считаться комментарием.

Например, комментарий в одну строку:

```
// это мой комментарий
```

Можно также задать комментарий в несколько строк:

```
// это мой

// очень-очень длинный

// комментарий
```

Как видите, таким способом неудобно создавать многострочные комментарии. Нужно начинать каждую строку с двух слешей. Гораздо удобнее воспользоваться другим способом создания комментариев.

Комментарием будет считаться все, что идет после /* и до */. Например:

```
/* Мой комментарий начался.

И продолжается он даже на новой строке.

Закончится он только тогда, когда я введу символы */
```

Присваиваем переменной значение

Значение, которое хранится в переменной, в любой момент можно заменить. Для этого используется запись вида:

```
имя_переменной=значение;
```

То есть все точно так же, как при объявлении переменной, только без var. Именно ключевое слово var в начале строки говорило о том, что мы объявляем новую переменную, а не изменяем значение существующей.

Символ = производит операцию присвоения, то есть представленная выше строка читается как «переменной имя_переменной присвоено значение значение».

На самом деле объявлять переменные вовсе не обязательно. Если в коде JavaScript будет встречена неизвестная переменная, она будет объявлена автоматически (то есть для нее будет выделен участок памяти под именем, которое задано неизвестной переменной в коде).

Типы переменных

Выше говорилось, что переменная может содержать число, строку или объект. На самом деле это не совсем так. От значения, которое хранится в переменной, зависит тип переменой. И типов переменных больше трех: Number, String, Boolean, Date, Math, Array, Object, Error.

При объявлении переменной можно указать, к какому типу она относится. Это делается следующим образом: var имя = new тип();.

Например:

```
var myvar = new Date(); // создать переменную типа Date
```

От типа переменной зависит не только формат ее значения, но и свойства и методы, которые она будет поддерживать.

Когда вы объявляете переменную без оператора new, тип переменной определяется автоматически, исходя из значения, которое в ней находится.

Типы `Number`, `String`, `Boolean` называются элементарными. При объявлении переменных данных типов лучше не использовать оператор `new`. Достаточно просто указать значение: `var myvar = 34;`.

Остальные типы переменных являются объектами. Вообще, в современном JavaScript все типы переменных являются объектами. Даже `Number`, `String` и `Boolean`.

Рассмотрим типы переменных подробнее.

Number

Переменными типа `Number` являются все переменные, в которых хранятся числовые значения: целые или десятичные.

На самом деле все числовые значения в JavaScript хранятся в формате `float64` (под них отводится 8 байт с плавающей точкой). При работе с десятичными значениями это может приводить к неточностям. Например, в JavaScript сложение чисел 0.1 и 0.2 приведет к результату 0.30000000000000004, а не к привычному нам 0.3.

В JavaScript можно не бояться таких математических ошибок, как деление на ноль. Результатом деления на ноль будет не ошибка и остановка выполнения сценария, а значения:

- `Number.POSITIVE_INFINITY` (плюс бесконечность) при делении на ноль положительного числа;
- `Number.NEGATIVE_INFINITY` (минус бесконечность) при делении на ноль отрицательного числа.

Различные ошибки могут также приводить к появлению значения `NaN` (результат не является числом). При этом выполнение сценария остановлено не будет.

String

Строковые переменные в JavaScript содержат в себе набор Unicode-символов (на один символ отводится 2 байта). Помимо обычных символов, внутри строки могут использоваться escape-последовательности.

Escape-последовательностью называется начинающийся со знака «\» набор символов, которые преобразуются в какой-либо нестандартный символ либо символ Unicode. Причем escape-последовательности, кодирующие символы Unicode, начинаются с «\u». Например:

```
var str1 = "это просто первая строка\nA это вторая";
```

В табл. 3.1 приведены основные escape-последовательности.

Таблица 3.1. Escape-последовательности

Последовательность	Описание
\0	Нуль
\b	Backspace (удалить символ слева от каретки)
\f	Переход на новую страницу документа
\n	Символ новой строки
\r	Возврат каретки
\t	Табуляция
\v	Вертикальная табуляция
\'	Апостроф или одинарная кавычка
\"	Двойная кавычка
\\	Обратный слеш (\)
\XXX	Символ в кодировке Latin-1 в диапазоне от 0 до 377. Например, \251 — это символ ©
\xXX	Символ в кодировке Latin-1 в диапазоне от 00 до FF (шестнадцатеричная система счисления). Например, \xA9 — это символ ©
\uXXXX	Символ в кодировке Latin-1, состоящий из четырех чисел в шестнадцатеричной системе счисления. Например, \x00A9 — это символ ©
\u0009	Табуляция
\u000B	Вертикальная табуляция
\u000C	Переход на новую страницу документа
\u0020	Пробел
\u00A0	Неразрывный пробел
\u000A	Переход на новую строку
\u000D	Возврат каретки
\u2028	Разделитель строк
\u2029	Разделитель абзацев
\u0000	Нуль
\u0008	Backspace
\u0009	Горизонтальная табуляция
\u0022	Двойная кавычка
\u0027	Одинарная кавычка
\u005C	Обратный слеш

Длина строки. Длину строки, которая находится в строковой переменной, можно узнать с помощью свойства `length`:

```
var str="привет";

var len=str.length; // len равно 6
```

Получение символа строки. В JavaScript существует несколько способов получить символ, который находится на указанной позиции в строке. Во всех этих способах позиция начинается с нуля (первый символ строки расположен на позиции 0).

Строку в JavaScript можно рассматривать как массив символов. И чтобы получить символ, расположенный на определенной позиции в строке, достаточно обратиться к данному элементу массива символов:

```
var str = "house";

var symb = str[1]; // symb равно "o"
```

Получить определенный символ из строки можно также с помощью метода `charAt(позиция)`, который поддерживают все строковые переменные в JavaScript:

```
var str = "house";

var symb = str.charAt(1); // symb равно "o"
```

Обратите внимание: перечисленными способами можно только получать символы, расположенные в строке. Изменять строку нельзя.

Boolean

Переменные типа `Boolean` (логические переменные) могут содержать в себе одно из двух значений:

- значение `true` означает «да» (истина);
- значение `false` означает «нет» (ложь).

Date

Как можно догадаться из названия, переменные типа `Date` хранят в себе дату. Не обязательно текущую дату. Дата может быть как уже прошедшей, так и еще не наступившей.

Дата может содержать в себе не только год, месяц и день, но и время. Вообще, существует множество вариантов объявления переменной типа `Date`:

```
var date1 = new Date(); /* текущая дата */

var date2 = new Date(число);

/* количество миллисекунд, прошедших от 1-Января-1970 00:00 */

var date2 = new Date(строка); /* строка, содержащая дату */
```

Строка, содержащая дату, должна быть записана в определенном формате. Причем этот формат может различаться для различных стран. Так, в Великобритании и многих других странах формат даты следующий: день, месяц, год. А в США формат даты: месяц, день, год.

Продолжим рассматривать варианты объявления переменной типа `Date`:

```
var date4 = new Date(число, число); /* год, месяц */

var date5 = new Date(число, число, число);

/* год, месяц, день */

var date6 = new Date(число, число, число, число);

/* год, месяц, день, час */

var date7 = new Date(число, число, число, число, число);

/* год, месяц, день, час, минута */

var date8 = new Date(число, число, число, число, число, число);
/* год, месяц, день, час, минута, секунда */
```

Месяц указывается начиная с нуля. То есть 0 — это январь.

Несколько примеров:

```
var date1 = new Date("21 May 2012 10:12");

var date2 = new Date(2012, 3, 1); /* 1 апреля 2012 года */
```

Math

Это весьма необычный объект. Прежде всего потому, что переменные типа `Math` создавать нельзя, то есть запись вида `var имя = new Math()` ошибочна.

В объекте класса `Math` хранятся различные математические функции, которые можно использовать для вычислений в своих сценариях. Это делается вызовом вида `Math.метод`. При этом никаких переменных типа `Math` не создается.

Например:

```
var var1 = 1.5;

var var2 = Math.round(var1); // var2 = 2
```

Метод `round` объекта `Math` позволяет округлить число по математическим правилам (если дробная часть больше 0,4, то аргумент округляется до большего).

Array

С помощью типа данных `Array` можно создавать массивы пронумерованных элементов. Подробнее об этом типе данных будет рассказано ниже.

Object

Объект без определенного класса.

Чаще всего объекты класса `Object` используются в JavaScript для создания ассоциативных массивов. Подробнее об этом будет рассказано в главе, посвященной массивам.

Error

Объекты класса `Error` создаются при различных ошибках выполнения сценария. Объект данного типа можно также объявить следующей записью:

```
new Error(сообщение об ошибке);
```

Конвертирование

В JavaScript значения одного типа можно приводить к значениям другого типа. Это бывает полезно, например, когда нужно сложить два числа, одно из которых действительно является числом, а другое представляет собой строку из цифр. Например, `34+"56"`.

Конвертирование в число

Number(). Чтобы конвертировать строку в число, достаточно воспользоваться записью `Number(строка)`. Например:

```
var tmp1 = 34+"56"; // tmp1 равно 3456

var tmp2 = 34+ Number("56"); // tmp2 равно 90
```

Конвертировать в число можно только строки, состоящие из чисел. Если вы попытаетесь конвертировать строку, которая состоит не только из чисел, то в качестве результата получите `NaN`.

parseFloat(). Существуют и другие способы преобразовать строку в число. Один из них — воспользоваться функцией `parseFloat(строка)`, позволяющей преобразовать строку в десятичное число.

Данная функция имеет существенное достоинство — она допускает наличие в строке нечисловых символов. Строка преобразуется в число до тех пор, пока в ней не будет обнаружен какой-либо символ, отличный от числа. При обнаружении такого символа остальная часть строки вместе с этим символом обрезается:

```
var var1=parseFloat("34"); // var1 равно 34

var var2=parseFloat("34.43"); // var2 равно 34.43

var var3=parseFloat("34.43sd"); // var3 равно 34.43

var var4=parseFloat("34.4sd3"); // var4 равно 34.4

var var5=parseFloat("ds34.43"); // var5 равно NaN
```

parseInt(). Существует также функция, позволяющая конвертировать строку в целое число в любой системе счисления: `parseInt(строка, система счисления)`. Чаще всего вы будете использовать десятичную систему счисления: `parseInt(строка, 10)`. Потому что именно в десятичной системе счисления выполняются все вычисления в окружающем вас мире.

Второй аргумент функции (система счисления) можно не указывать. В этом случае обработчик JavaScript попытается самостоятельно определить систему счисления. Например:

```
parseInt("0x10") = 16 // шестнадцатеричная система счисления

parseInt("010") = 8 // восьмеричная система счисления
```

Если строка начинается с `0x`, значит, число задано в шестнадцатеричной системе счисления. Если строка начинается с нуля, значит, это точно восьмеричная система счисления.

Как и функция `parseFloat`, функция `parseInt` допускает наличие нечисловых символов в строке:

```
var var1=parseInt("34", 10); // var1 равно 34

var var2=parseInt("34.43", 10); // var2 равно 34

var var3=parseInt("3443sd", 10); // var3 равно 3443

var var4=parseInt("344sd3", 10); // var4 равно 344

var var5=parseInt("ds34.43", 10); // var5 равно NaN
```

+. Существует еще один способ неявного преобразования строки в число — использование унарного оператора +. Здесь имеется в виду не операция сложения, а определение положительного числа. Например:

```
var tmp1 = 34+"56"; // tmp1 равно 3456

var tmp2 = 34+ (+"56"); // tmp2 равно 90
```

Собственно говоря, для конвертирования в число вместо + можно использовать и унарный оператор –. Но тогда число станет отрицательным:

```
var tmp1 = 34+"56"; // tmp1 равно 3456

var tmp2 = 34+ (-"56"); // tmp2 равно -22
```

Конвертирование в логический тип (Boolean)

Boolean(). Для конвертирования в тип `Boolean` достаточно воспользоваться записью `Boolean(значение)`.

Результатом конвертирования будет либо `true`, либо `false`.

Если в логический тип конвертируются значения `false`, `null`, `undefined`, `""` (пустая строка), `0`, `Number.NaN`, то результатом будет `false`.

Во всех остальных случаях результатом будет `true`.

!!. Еще одним способом конвертировать значение в логический тип является использование двойного отрицания: `!!значение`.

Свойства

В JavaScript переменные могут поддерживать различные свойства. Причем список поддерживаемых переменной свойств зависит от типа данной переменной.

Свойством можно назвать переменную внутри переменной. То есть свойство позволяет получить какое-либо значение, как правило, относящееся к переменной, для которой данное свойство задано. Проще всего разобраться в этом на примерах.

Ранее в книге мы уже встречались со свойствами. Например, свойство `length`, которое поддерживают все переменные типа `String`, позволяет получить длину строки, хранящейся в переменной:

```
var str = "свойство?";

var num = str.length; // num равно 9
```

Обратиться к определенному свойству можно следующим образом: `переменная.свойство`. Такое обращение называется точечной нотацией.

Помимо точечной нотации, существует еще один способ обратиться к свойству: `переменная["свойство"]`. Например:

```
var str = "свойство!";

var num = str["length"]; // num равно 9
```

Большинство свойств предназначено только для чтения, то есть вы можете лишь получить значение, хранящееся в данном свойстве. Изменить это значение нельзя.

Названия свойств JavaScript регистрозависимы. Например, строка `str.length` в коде работать будет, а вот строка `str.Length` не будет. Так что следует запоминать не только название свойства, но и его написание. Впрочем, большинство свойств в JavaScript полностью пишутся строчными буквами.

Некоторые свойства можно, а иногда и нужно вызывать без объявления конкретной переменной. Такие свойства называются статическими. В этом случае свойство указывается после названия класса, к объектам которого оно относится: `класс.свойство`. Например:

- `Math.PI` — возвращает значение числа «пи»;
- `Math.E` — возвращает значение числа Эйлера;
- `Number.MAX_VALUE` — возвращает максимальное число, которое можно записать в JavaScrit.

Далее мы кратко рассмотрим свойства, которые поддерживают переменные различных типов в JavaScript (табл. 3.2–3.6).

Таблица 3.2. Свойства переменных типа String

Свойство	Описание
length	Длина строки

Таблица 3.3. Свойства переменных типа Number

Свойство	Значение	Описание
Number.MAX_VALUE*	≈1.79E+308	Возвращает наибольшее возможное число в JavaScript
Number.MIN_VALUE*	≈5.00E-324	Возвращает наименьшее возможное число в JavaScript (наиболее близкое к нулю)

Продолжение ⇩

Таблица 3.3 *(продолжение)*

Свойство	Значение	Описание
Number.NaN*	–	Возвращает значение NaN
Number.NEGATIVE_ INFINITY*	–	Возвращает значение «отрицательная бесконечность»
Number.POSITIVE_INFINITY*	–	Возвращает значение «положительная бесконечность»
NaN	–	Возвращает значение NaN

* Статическое свойство.

Таблица 3.4. Свойства объекта класса Math

Свойство	Значение	Описание
Math.E	≈2.718	Константа Эйлера, основание натурального логарифма
Math.LN2	≈0.693	Натуральный логарифм 2
Math.LOG2E	≈1.442	Логарифм E по основанию 2
Math.LOG10E	≈0.434	Логарифм E по основанию 10
Math.PI	≈3.14159	Отношение длины окружности к ее диаметру
Math.SQRT12	≈0.707	Квадратный корень из 1/2; иначе говоря, 1, деленное на корень из 2
Math.SQRT2	≈1.414	Квадратный корень из 2

Все свойства класса `Math` являются статическими.

Таблица 3.5. Свойства переменных типа Array

Свойство	Описание
index	Для массива, созданного в результате сравнения на соответствие регулярному выражению, данное свойство содержит индекс найденного соответствия в исследуемой строке
input	Для массива, созданного в результате сравнения на соответствие регулярному выражению, данное свойство содержит строку, над которой производили сравнение
length	Длина массива

Таблица 3.6. Свойства переменных типа Error

Свойство	Описание
message	Краткое описание ошибки
name	Название типа ошибки

Методы

Помимо свойств, переменные могут поддерживать различные методы. Список доступных методов зависит от типа переменной.

Методом называется последовательность команд, в результате выполнения которых возвращается какое-либо значение. Как правило, метод каким-либо образом относится к переменной, для которой вы его вызываете.

Основной синтаксис вызова метода следующий: `переменная.метод()`.

Так вызываются методы, не требующие никаких дополнительных аргументов. Но для работы некоторых методов необходимо также указывать один или несколько аргументов (значений, которые будут использоваться в вычислениях во время выполнения метода). Аргументы указываются в круглых скобках, через запятую:

```
переменная.метод(аргумент);

переменная.метод(аргумент, аргумент);

переменная.метод(аргумент, аргумент, аргумент);
```

И так далее. Количество необходимых аргументов зависит от самого метода.

Поскольку любой метод возвращает какое-либо значение, это значение нужно получить и сохранить в переменной:

```
переменная = переменная.метод(аргумент);
```

Например:

```
var num = 12.3456;

var num2 = num.toFixed(); // num2 равно 12

var num3 = num.toFixed(2); // num3 равно 12.34
```

Как и свойства, названия методов в JavaScript регистрозависимы. Поэтому следует запоминать не только название метода, но и его написание.

Некоторые методы в JavaScript являются статическими. То есть их можно вызывать без объявления конкретной переменной. В этом случае метод указывается после названия класса, к объектам которого он относится: `класс.метод`.

В табл. 3.7–3.11 показаны методы различных переменных.

String

Таблица 3.7. Методы переменных типа String

Метод	Описание
split([separator][, limit])	Создает массив из строки, разбивая строку по разделителю на отдельные элементы массива. Например: arr = "a,b,c".split(',') // массив ["a", "b", "c"]
charCodeAt(index)	Возвращает Unicode-значение символа, расположенного по данному индексу
String.fromCharCode (num1, ..., numN)*	Преобразует Unicode-значение символа в сам символ и возвращает строку, состоящую из полученных символов
charAt(index)	Возвращает отдельный символ из строки, расположенный по данному индексу
concat(string2, string3[, ..., stringN])	Объединяет две строки (аналог операции +)
lastIndexOf(searchValue[, fromIndex])	Ищет подстроку в строке, начиная с конца строки
search(regexp)	Выполняет поиск по строке с помощью регулярных выражений
match(regexp)	Производит поиск по строке с помощью регулярных выражений
toLowerCase()	Преобразует символ в нижний регистр
toUpperCase()	Преобразует символ в верхний регистр
toLocaleLowerCase()	Преобразует символ в нижний регистр (от toLowerCase отличается только для некоторых языков, правила которых противоречат соглашениям Unicode)
toLocaleUpperCase()	Преобразует символ в верхний регистр (от toUpperCase отличается только для некоторых языков, правила которых противоречат соглашениям Unicode)
toString()	Возвращает элементарную строку
valueOf()	Возвращает элементарное значение для объекта String
substring(indexA, [indexB])	Возвращает подстроку, начиная с позиции index1 и до index2
slice(beginSlice[, endSlice])	Возвращает часть строки, начиная с позиции index1 и до index2. При этом вызывающая строка не изменяется
indexOf(searchValue[, fromIndex])	Ищет подстроку в строке, начиная с начала строки
substr(start[, length])	Возвращает часть строки
replace(regexp, newSubStr|function)	Возвращает строку, в которой была произведена замена

* Статический метод.

Number

Таблица 3.8. Методы переменных типа Number

Метод	Описание
toExponential ([fractionDigits])	Возвращает число в экспоненциальной записи (в виде строки с одной цифрой до десятичной точки, округленное до ractionDigits цифр после десятичной точки)
toString([radix])	Преобразует число в строку
toPrecision([precision])	Округляет число до общего количества цифр независимо — после запятой или нет
toFixed([fractionDigits])	Возвращает строковое представление числа без использования экспоненциальной записи и ровно с fractionDigits цифр после запятой (то есть отбрасывает все цифры в числе после определенной позиции)

Math

Таблица 3.9. Методы объектов класса Math

Метод	Описание
Math.abs(x)	Возвращает абсолютное значение числа (положительное значение числа)
Math.acos(x)	Возвращает результат выполнения операции acos
Math.asin(x)	Возвращает результат выполнения операции asin
Math.atan(x)	Возвращает результат выполнения операции atan
Math.atan2(y, x)	Возвращает числовое значение между –PI и PI, представляющее собой угол Theta для точки (x; y)
Math.cos(x)	Возвращает результат выполнения операции cos
Math.sin(x)	Возвращает результат выполнения операции sin
Math.exp(x)	Возвращает значение E (константа Эйлера) в степени x
Math.max([value1 [,value2[, ...]]])	Возвращает большее из заданных чисел
Math.min([value1 [,value2[, ...]]])	Возвращает меньшее из заданных чисел
Math.random()	Возвращает случайное число с плавающей точкой в промежутке от 0 до 1
Math.sqrt(x)	Возвращает квадратный корень из числа
Math.tan(x)	Возвращает результат выполнения операции tan
Math.log(x)	Возвращает натуральный (по основанию E) логарифм числа
Math.pow(base, exponent)	Возводит число в степень
Math.ceil(x)	Возвращает наименьшее целое, большее или равное аргументу (округляет число в большую сторону)
Math.floor(x)	Возвращает наибольшее целое, меньшее или равное аргументу (округляет число в меньшую сторону)
Math.round(x)	Округляет число по математическим правилам

Все методы класса `Math` являются статическими.

Date

UTC (Universal Coordinated Time, всемирное координированное время) — мировое время, которое было введено в 1964 году. Раньше UTC называлось «среднее время по Гринвичу» и обозначалось аббревиатурой GMT.

С мировым временем в JavaScript работают все методы, в названии которых есть подстрока `UTC`. Остальные методы работают с местным временем, то есть с временем, использующимся для города, в котором вы находитесь в данный момент (определяется по часовому поясу, установленному в настройках операционной системы).

Таблица 3.10. Методы переменных типа Date

Метод	Описание
Date.parse(dateVal)*	Получает строку с датой (напр. "Jan 03, 2000") и возвращает целое число, представляющее собой количество миллисекунд, которые истекли с полуночи 1 января 1970 года GMT+0 до этой даты
toLocaleString()	Возвращает строку, содержащую дату в длинном формате
getDate()	Возвращает целое число от 1 до 31 (день месяца)
getDay()	Возвращает целое число, обозначающее день недели: 0 — воскресенье, 1 — понедельник, 2 — вторник и т. д.
getFullYear()	Возвращает значение года, состоящее из четырех чисел
getMilliseconds()	Возвращает целое число от 0 до 999
getMinutes()	Возвращает целое число от 0 до 59
getSeconds()	Возвращает целое число от 0 до 59
getTimezoneOffset()	Возвращает смещение часового пояса (разницу между универсальным (UTC) и местным временем в минутах)
getUTCDate()	Возвращает целое число от 1 до 31
getUTCDay()	Возвращает целое число, обозначающее день недели в формате UTC: 0 — воскресенье, 1 — понедельник, 2 — вторник и т. д.
getUTCHours()	Возвращает целое число от 0 до 23
getUTCMilliseconds()	Возвращает целое число от 0 до 999 в формате UTC
getUTCMinutes()	Возвращает целое число от 0 до 59 в формате UTC
getUTCSeconds()	Возвращает целое число от 0 до 59 в формате UTC
setSeconds(secondsValue)	Изменяет количество секунд
setFullYear(yearValue [, monthValue[, dayValue]])	Устанавливает год
setMilliseconds(millisecondsValue)	Изменяет количество миллисекунд
setTime(timeValue)	Устанавливает время
setYear(yearValue)	Устанавливает год
setDate(dayValue)	Устанавливает дату
setUTCDate(dayValue)	Устанавливает дату в формате UTC

Метод	Описание
valueOf()	Возвращает количество миллисекунд, прошедших с полуночи 1 января 1970 года
setUTCHours(hoursValue [, minutesValue [, secondsValue[, msValue]]])	Устанавливает час в формате UTC
setHours(hoursValue [, minutesValue [, secondsValue [, msValue]]])	Устанавливает количество часов
setUTCMilliseconds(milli-secondsValue)	Устанавливает количество миллисекунд в формате UTC
setUTCMinutes(minutes-Value[, secondsValue [, msValue]])	Устанавливает количество минут в формате UTC
setMinutes(minutesValue [, secondsValue[, msValue]])	Устанавливает количество минут
setMonth (monthValue [, dayValue])	Устанавливает месяц
setUTCSeconds(secondsVa-lue)	Устанавливает количество секунд в формате UTC
setUTCFullYear(yearValue [, monthValue[, dayValue]])	Устанавливает год в формате UTC
setUTCMonth (monthValue[, dayValue])	Устанавливает месяц в формате UTC
toGMTString()	Возвращает строку, содержащую текстовое представление даты и времени по Гринвичу на английском языке в американском формате. Вместо этого метода следует использовать Date.toUTCString
toLocaleFormat(format-String)	Возвращает отформатированную по настройкам операционной системы информацию о дате и времени
toLocaleTimeString()	Возвращает отформатированную по настройкам операционной системы информацию о времени в виде строки
toLocaleDateString()	Возвращает отформатированную по настройкам операционной системы информацию о дате в виде строки
toString()	Возвращает строку, представляющую дату на английском языке в американском формате
toTimeString()	Возвращает информацию о времени в виде строки
toDateString()	Возвращает информацию о дате в виде строки на английском языке в американском формате
toUTCString()	Возвращает строку на английском языке в американском формате согласно универсальному часовому поясу (Гринвичу)
getUTCFullYear()	Возвращает полное значение года

Продолжение ⇩

Таблица 3.10 *(продолжение)*

Метод	Описание
getMonth ()	Возвращает целое число от 0 до 11; 0 соответствует январю, 1 — февралю и т. д.
Date.UTC(year, month, day[, hours[, minutes [, seconds[,ms]]]])*	Возвращает количество миллисекунд, истекших с полуночи 1 января 1970 года и указанной датой, используя время в формате UTC
getUTCMonth()	Возвращает целое число от 0 до 11; 0 соответствует январю, 1 — февралю и т. д.
getTime()	Возвращает количество миллисекунд, прошедших с полуночи 1 января 1970 года GMT
getYear()	Возвращает год

* Статический метод.

Array

Таблица 3.11. Методы переменных типа Array

Метод	Описание
push(elem1, elem2, ...)	Добавляет элементы в массив, начиная с текущей длины, и возвращает увеличенную длину массива
shift()	Удаляет из массива элемент с индексом 0 и сдвигает остальные элементы на один вниз. Возвращает удаленный элемент
unshift([elem1[, elem2[, ... [, elemN]]]])	Добавляет в массив аргументы и возвращает получившуюся длину
join([glue])	Преобразует массив в строку, соединяя в строку все элементы массива
concat(value1, value2, ..., valueN)	Создает новый массив, копируя в него текущий массив и перечисленные значения
splice(start, deleteCount, [elem1[, elem2[, ... [, elemN]]]])	Удаляет часть массива и добавляет новые элементы на место удаленных. Возвращает массив из удаленных элементов
pop()	Удаляет последний элемент массива и возвращает его
slice(start[, end])	Возвращает часть массива, не изменяя исходного массива (выполняет копирование части массива)
reverse()	Изменяет порядок элементов в массиве на обратный и возвращает ссылку на измененный массив
sort([sortFunction])	Сортирует массив

Массивы

Массивом называется набор переменных, к которым обращаются по одному и тому же имени, но с разным смещением (индексом, ключом). Массивы используют для

удобства, когда нужно хранить множество сходных по назначению переменных. Удобство также заключается в возможности изменять значения элементов массива в цикле.

Объявление массива

Массив создается конструкцией следующего вида:

```
var имя = new Array();
```

Так мы сообщаем, что в переменной `имя` будет находиться массив. Однако сам массив значениями не заполняем.

Существует также способ объявить массив и сразу же заполнить его значениями. Это делается так:

```
var имя = new Array(значение1, значение2, значение3);
```

То есть в скобках через запятую указывается столько значений массива, сколько вам нужно. При этом минимальное количество значений — два. Если вы зададите в скобках одно значение, то это будет совсем другая команда.

Всем значениям, которые вы укажете в скобках, будет присвоен свой индекс — число, по которому вы в любой момент сможете получить или изменить хранящееся в данной ячейке массива значение. Первому значению присваивается индекс 0, второму значению индекс 1, третьему — индекс 2 и т. д.

Итак, индексы элементов массива начинаются с нуля. И последовательно увеличиваются для каждого нового значения массива.

Существует еще один способ объявить массив со значениями:

```
var имя = [значение1, значение2, значение3];
```

Квадратные скобки и содержащееся в них значение называются инициализатором массива. Внутри квадратных скобок значений может быть как несколько, так и одно. Тогда как при использовании `new Array` можно указывать минимум два значения.

Кроме того, внутри квадратных скобок можно задавать неопределенные значения, то есть вообще пропускать какое либо значение:

```
var имя = [значение1,, значение3, значение4];
```

В этом случае элементу с индексом 1 (второму элементу) будет присвоено неопределенное значение.

Обращаемся к элементам массива

После того как массив создан, вы в любой момент можете обратиться к отдельным переменным, из которых этот массив состоит (к отдельным элементам массива). Конечно, если вы задали значения при объявлении массива.

Как вы уже знаете, каждому элементу массива присваивается свой индекс. Первому элементу — индекс 0, второму — индекс 1 и т. д. Чтобы обратиться к определенному элементу массива, нужно указать имя массива и индекс, к которому вы обращаетесь: `имя[индекс]`. Например:

```
var myArray = new Array(23, "hello", true);

var tmp = myArray[1]; // в tmp содержится значение "hello"

var tmp2 = myArray[0]; // в tmp содержится значение 23
```

Присваивание значений элементам массива

Зная имя массива и нужный индекс, вы в любой момент сможете не только получить хранящееся по данному индексу значение, но и изменить его. Это делается очень просто:

```
имя[индекс]=значение;
```

То есть это обычная операция присваивания, с помощью которой вы присваиваете элементу массива `имя` с индексом `индекс` значение `значение`.

Этой же операцией присваивания можно создавать новые элементы массива. Просто присвойте какое-либо значение элементу массива с индексом, которого еще не существует, — и элемент с указанным индексом будет автоматически создан. Например:

```
var myArray = new Array(23, "hello", true);

myArray[0] = 777; // было значение 23, теперь значение 777

myArray[23] = 11; // создали элемент массива с индексом 23
```

Длина массива

Длину массива можно узнать с помощью свойства `length`, которое есть у каждого массива (переменной, в которой хранится массив). Чтобы узнать длину массива, достаточно воспользоваться следующей записью: `имя.length`. Например:

```
var myArray = new Array(23, "hello", true);

var tmp = myArray.length; // tmp равно 3
```

```
myArray[23] = 77;

var tmp2 = myArray.length; // tmp2 равно 24
```

Обратите внимание на комментарии в примере, особенно на последний. Он не ошибочен. Длина массива действительно равна 24. Дело в том, что свойство `length` определяет длину массива не по количеству элементов в нем, а по наибольшему индексу, который есть в массиве.

Итак, длина массива равна наибольшему индексу массива плюс один. Поэтому элементы массива рекомендуется задавать с последовательными индексами, не перескакивая между ними.

Если вам нужно создать новый элемент массива, но вы не знаете, какой индекс сейчас у последнего элемента массива, вы можете воспользоваться свойством `length`:

```
var myArray = new Array(23, "hello", true);

myArray[23] = 77;

myArray[myArray.length] = 33; // myArray[24] равно 33
```

Интересен также тот факт, что свойство `length` доступно не только для чтения, но и для записи. Если вы присвоите данному свойству значение меньше текущей длины массива, то «урежете» массив (удалите все элементы массива, которые вышли за границы массива). Если же присвоить свойству значение больше длины массива, то массив будет расширен за счет создания пустых элементов.

Работа с началом и концом массива

Ситуации, когда нужно получить или поместить элемент в начало или в конец массива, возникают довольно часто. Для выполнения данных операций существуют специальные методы, которые поддерживаются любым массивом:

- `push(значение)` — создает элемент в конце массива и помещает в него указанное значение (то есть команда аналогична уже изученной нами команде `имя_массива[имя_массива.length]`);
- `pop()` — возвращает значение, которое находится в последнем элементе массива (в элементе с наибольшим индексом), при этом сам элемент с данным индексом удаляется из массива;
- `unshift(значение)` — создает элемент в начале массива и помещает в него указанное значение;
- `shift()` — возвращает значение, которое находится в первом элементе массива, при этом сам элемент удаляется из массива.

Например:

```
var myArray = new Array(23, 33, 43);

myArray.push(53); // Array(23, 33, 43, 53)

myArray.unshift(13); // Array(13, 23, 33, 43, 53)

myArray.pop(); // Array(13, 23, 33, 43)

myArray.shift(); // Array(23, 33, 43)
```

Многомерные массивы

Многомерным в JavaScript называется массив, значением которого является другой массив. В зависимости от количества вложенных массивов определяется уровень многомерности:

- если один массив вложить в другой, получится двухмерный массив;
- если один массив вложить в другой, а его, в свою очередь, в третий массив, то получится трехмерный массив и т. д.

Объявляем многомерный массив. Итак, чтобы создать многомерный массив, достаточно создать обычный массив, после чего одному из элементов массива присвоить другой массив:

```
var mass = new Array();

mass[0]= new Array();
```

Или можно так:

```
var mass = new Array(new Array(23, 34),new Array(11, 12));
```

Многомерные массивы можно также создавать с помощью квадратных скобок:

```
var mass = [[22, 34], [11, 12]];
```

Согласитесь, данная запись и короче и понятнее предыдущей, использующей для объявления массива оператор `new`.

Все рассмотренные выше способы создают двухмерный массив. В трехмерном массиве количество вложенных массивов достигает трех:

```
var mass = [[[22, 34], [77, 55]], [11, 12]];
```

Обращаемся к элементам массива. Чтобы получить или изменить значение элемента многомерного массива, достаточно в отдельных квадратных скобках последовательно указать индексы всех массивов, которые в него входят:

```
var mass = [23];
mass[1] = [33, 44]; // создаем двухмерный массив
mass[1][3] = [55, 66]; // создаем трехмерный массив
var var1 = mass[0]; // var1 = 23
var var2 = mass[1][0]; // var2 = 33
var var3 = mass[1][3][1]; // var3 = 66
```

Ассоциативные массивы

В обычных массивах индексом элемента массива является какое-либо число (`mas[0]`, `mas[4]` и т. д.). Ассоциативным же называется такой массив, в котором индексом элемента является строка (`mas["anna"]`, `mas["house"]` и т. д.).

В ассоциативных массивах вместо индекса используется понятие «ключ». В строке `mas["house"]` ключом элемента ассоциативного массива `mas` является строка `house`.

Объявляем ассоциативный массив. Ассоциативные массивы создаются с помощью объектов класса `Object`. Чаще всего они объявляются следующим образом:

```
имя = {
   ключ1: значение1,
   ключ2: значение2,
   ...
}
```

В данном случае фигурные скобки и их содержимое называются инициализатором объекта.

Обратите внимание: ключ ассоциативного массива не берется в кавычки. Тогда как значение, если оно является строкой, берется. Например:

```
mas = {
   house: 23,
   street: "Тополево, 8",
   city: "Москва",
}
```

Обращаемся к элементам массива. Обращение к элементам ассоциативного массива выполняется по ключу. Собственно, все точно так же, как и с обычными массивами. Например, попробуем обратиться к элементам описанного выше ассоциативного массива:

```
var var1 = mas["house"]; // var1 равно 23

var var2 = mas["street"]; // var2 равно "Тополево, 8"

var var3 = mas["city"]; // var3 равно "Москва"
```

Арифметические операции

Все мы еще со школы знаем такие арифметические операции, как сложение, вычитание и умножение. Все эти операции поддерживаются в JavaScript. В табл. 3.12 перечислены возможные арифметические операции и способы их выполнения в JavaScript.

Таблица 3.12. Арифметические операции в JavaScript

Название	Символ	Пример
Сложение чисел	+	var res = var1+var2;
Конкатенация строк	+	var res = var1+var2;
Вычитание	-	var res = var1-var2;
Умножение	*	var res = var1*var2;
Обычное деление	/	var res = var/var2;
Деление по модулю	%	var res = var%var2;
Единичный инкремент	++	res++ или ++res
Единичный декремент	--	res-- или --res

Сложение и конкатенация. Как можно заметить из таблицы, один и тот же оператор используется и для сложения, и для конкатенации. Если действие выполняется над числами, то производится сложение. Если же хотя бы один из операндов является строкой, то происходит конкатенация, то есть объединение строк:

```
var res1 = 12 + 23; // res1 содержит число 35

var res2 = 12 + "23"; // res2 содержит строку 1235

var res3 = "12" + "23"; // res3 содержит строку 1235
```

Инкремент и декремент. Обратите также внимание на варианты таких арифметических операций, как единичный инкремент и декремент. Эти операции могут записываться двумя способами:

- ++res или res++;
- --res или res--.

Между этими способами записи есть кардинальное различие, с которым вы столкнетесь при использовании инкремента/декремента в вычислениях. Проще всего понять это различие на примерах:

```
var var1 = 6;

var var2 = 6;


var1++; // var1 равно 7

++var2; // var2 равно 7

// Разницы нет.


var sum1 = var1++; // sum1 равно 7, var1 равно 8

var sum2 = ++var2; // sum2 равно 8, var2 равно 8

// Есть разница!


sum1 = sum1 + var1++; // sum1 равно 15, var1 равно 9

sum2 = sum2 + ++var2; // sum2 равно 17, var2 равно 9

// Есть разница!
```

Итак:

- ++переменная — сначала выполняется увеличение значения переменной на единицу, после чего выполняются другие операции над этой переменной;
- переменная++ — сначала выполняются операции над переменной, а потом значение переменной увеличивается на единицу.

Типы операндов. Как правило, арифметические операции выполняются над числами. Однако так бывает не всегда:

- Number и Number — если оба операнда числовые, выполняется арифметическая операция;
- Number и Boolean — если один из операндов логический, он приводится к числовому значению (значение true становится 1, а значение false — 0);

- `Boolean` и `Boolean` — если оба операнда логические, то они оба преобразуются в числовые (значение `true` становится 1, а значение `false` — 0);
- `Boolean` и `String` — логический операнд преобразуется в строку;
- `Number` и `String` — число преобразуется в строку.

Операторы присваивания. Пока вы знаете только об одном операторе присваивания — операторе =. Но в JavaScript существуют и другие разновидности данного оператора. В частности, поддерживаются операторы присваивания, которые перед присваиванием выполняют различные арифметические операции над присваиваемым элементом и элементом, которому присваивают:

- `var1*= var2` — данная запись аналогична `var1 = var1*var2`;
- `var1/= var2` — данная запись аналогична `var1 = var1/var2`;
- `var1%= var2` — данная запись аналогична `var1 = var1%var2`;
- `var1+= var2` — данная запись аналогична `var1 = var1+var2`;
- `var1-= var2` — данная запись аналогична `var1 = var1-var2`.

Условные операторы

В сценариях условные операторы используются довольно часто. Они позволяют выполнить набор команд только в том случае, если истинно заданное условие.

If

Наиболее простой синтаксис условного оператора:

```
if(условие1){
   команда1;
   команда2;
}
```

Если условие истинно, то будут выполнены все команды, расположенные внутри фигурных скобок. Иначе команды, размещенные в фигурных скобках, будут пропущены.

Если команда только одна, фигурные скобки можно опустить: `if(условие1) команда1;`. Однако практика показывает, что лучше этого не делать. Во время программирования довольно часто одна команда превращается в несколько. И все равно приходится добавлять фигурные скобки.

Существует и более сложный синтаксис данного условного оператора:

```
if(условие1){
   команда1;
   команда2;
}else{
   команда3;
   команда4;
}
```

В этом случае если условие истинно, то будут выполнены все команды, расположенные внутри первых фигурных скобок. Если же условие ложно, то будут выполнены команды, размещенные внутри вторых фигурных скобок (после `else`).

Но это еще не самый сложный синтаксис оператора `if`. Как вам такой синтаксис:

```
if(условие1){
   команда1;
   команда2;
}else if(условие2){
   команда3;
   команда4;
}else{
   команда5;
   команда6;
}
```

В данном случае если `условие1` истинно, то будут выполнены все команды, расположенные внутри первых фигурных скобок. В противном случае проверяется на истинность `условие2`. И если оно истинно, то будут выполнены все команды, расположенные внутри вторых фигурных скобок. Если же и первое и второе условия ложны, то выполняются команды, размещенные внутри третьих фигурных скобок (после `else`).

Количество условий в приведенной конструкции неограниченно. Например:

```
var num = 12;
var k;
if(num == 1){
    k = 101;
}else if(num == 2){
    k = 202;
}else if(num < 10){
    k = 1010;
}else if(num > 20){
    k = 2020;
}else{
    k = num*100;
}
```

Условия. В качестве условия используется любая конструкция сравнения одного значения с другим. Если результат сравнения равен `true`, то условие выполняется.

При сравнении строк важен регистр каждого отдельного символа строки. Если хотя бы один символ в строке отличается, результатом сравнения будет `false`.

В условном операторе `if` возможны следующие варианты сравнения (операторы отношения):

- `значение1 == значение2` — «если `значение1` равно `значению2`, то»;
- `значение1 != значение2` — «если `значение1` не равно `значению2`, то»;
- `значение1 === значение2` — «если `значение1` строго равно `значению2` (совпадает не только значение, но и тип значения), то»;
- `значение1 !== значение2` — «если `значение1` строго не равно `значению2` (операнды не равны и/или не совпадает их тип), то»;
- `значение1 > значение2` — «если `значение1` больше `значения2`, то»;
- `значение1 >= значение2` — «если `значение1` больше или равно `значению2`, то»;

- `значение1 < значение2` — «если `значение1` меньше `значения2`, то»;
- `значение1 <= значение2` — «если `значение1` меньше или равно `значению2`, то»;
- `!значение1` — «если `значение1` равно `false` (то есть не существует либо равно `false`, `null`, `undefined`, `""` (пустая строка), `0` или `Number.NaN`), то»;
- `значение1` — «если `значение1` равно `true` (то есть существует и не равно `false`, `null`, `undefined`, `""` (пустая строка), `0` или `Number.NaN`), то».

Значениями в условиях могут выступать как переменные, так и числа, логические значения, либо математические выражения. Например:

- `if(true == false){}` — условие никогда не выполнится;
- `if(true)` — условие всегда выполняется;
- `if(var1 === false){}` — условие выполнится, только если `var1` имеет тип `Boolean` и содержит значение `false`;
- `if(var1-var2 == 0){}` — условие выполнится, только если `var1` и `var2` содержат одинаковые значения.

В условных операторах можно задавать сразу несколько условий. В этом случае все условия указываются внутри одних круглых скобок оператора `if` и между условиями задаются правила подчинения. Проще говоря, условия объединяются одним из логических операторов:

- `&&` — логическое И, то есть если и первое и второе условия истинны;
- `||` — логическое ИЛИ, то есть если истинно хотя бы одно из условий.

Чтобы не запутаться, лучше брать каждое отдельное условие в круглые скобки.

Например:

- `if((var1 === false) && (var2 > 23)){}` — условие выполнится, только если `var1` имеет тип `Boolean`, содержит значение `false` и при этом значение переменной `var2` больше 23;
- `if((var1 == 10) || (var1 == 20)){}` — условие выполнится, если `var1` равно 10 или 20;
- `if(((var1 == 10) || (var1 == 20)) && (var2 > 23)){}` — условие выполнится, если `var1` равно 10 или 20 и при этом `var2` больше 23.

?:

Нет, это не проблемы с кодировкой символов. `?:` — это упрощенный синтаксис условного оператора, который можно использовать вместо оператора `if`.

Основной синтаксис данного условного оператора следующий:

```
переменная=(условие)?(значение1):(значение2);
```

В представленном синтаксисе переменной присваивается значение1, если условие истинно. В противном случае переменной присваивается значение2.

Например:

```
var num = 34;

var str = (num>50)?("очень много"):("слишком мало");

// str равно "слишком мало"
```

Если переписать представленный выше пример с помощью оператора if, то получится:

```
var num = 34;

var str = "";

If(num>50){

   str="очень много";

}else{

   str="слишком мало";

}
```

Switch

При разработке сайта может возникнуть ситуация, когда одну и ту же переменную нужно сравнивать сразу с множеством различных значений и в зависимости от значения этой переменной выполнять то или иное действие.

Если решать данную задачу уже известными нам методами, то на помощь нам придет условный оператор if. А точнее, его полная запись:

```
if(var1 == 1){

   команды1

}else if(var1 == 2){

   команды2

}else if(var1 == 3){
```

```
   команды3
}else if(var1 == 4){
   команды4
}else if(var1 == 5){
   команды5
}else{
   команды6
}
```

Количество операторов `if` можно увеличить. Но давайте остановимся на этом. Ведь уже сейчас видно, что данный способ крайне громоздок и неудобен. В JavaScript есть более простой способ выполнения разных команд в зависимости от значения переменной — оператор `switch`.

Оператор `switch` имеет следующий синтаксис:

```
switch(переменная){
case значение1:
   команды1;
   break;
case значение2:
   команды2;
   break;
default:
   команды3;
}
```

В представленном примере, если значение `переменной` равно `значение1`, то будут выполнены `команды1`. Если значение `переменной` равно `значение2`, то будут выполнены `команды2`. Если ни один из операторов `case` не указывает на текущее значение переменной, то будут выполнены команды из секции `default` — `команды3`.

Если переписать представленный выше пример с условным оператором `if`, то получится следующее:

```
switch(var1){
case 1:
   команды1
   break;
case 2:
   команды2
   break;
case 3:
   команды3
   break;
case 4:
   команды4
   break;
case 5:
   команды5
   break;
default:
   команды6
}
```

Оператор `break` используется не только внутри `switch`, но и внутри циклов (о них вы прочтете в следующей главе). Он предназначен для того, чтобы незамедлительно завершить выполнение цикла либо оператора `switch`.

Внутри `switch` он используется, чтобы отделить команды одного значения переменной, от команд другого значения. То есть если не указать оператор `break`, то будут выполнены и команды из следующих вариантов значения переменной. Например:

```
switch(var1){
case 1:
   команды1
   break;
case 2:
   команды2
case 3:
   команды3
   break;
default:
   команды4
}
```

В данном примере:

- если `var1` равно 1, то будут выполнены `команды1`;
- если `var1` равно 2, то будут выполнены `команды2` И (обратите внимание!) `команды3`;
- если `var1` равно 3, то будут выполнены `команды3`;
- если `var1` не равно 1, 2 и 3, то будут выполнены `команды4`.

Иногда разработчики намеренно не задают оператор `break`, чтобы реализовать нужное им поведение. Но чаще всего это делается по забывчивости и приводит к ошибкам в работе сценария.

Циклы

Циклом называется повторение выполнения одной и той же команды (либо набора команд) множество раз. Повторение цикла может происходить:

- либо до тех пор, пока не станет истинным какое-либо условие;
- либо пока команда (набор команд) не будет выполнена определенное количество раз.

For

Цикл `for` позволяет выполнить набор команд указанное количество раз. Основной синтаксис данного цикла следующий:

```
for(инициализация; условие; шаг){

   команды

}
```

В первом аргументе цикла `for` мы задаем переменную, с помощью которой будем определять, сколько еще раз нужно повторить выполнение команд (инициализация переменной-счетчика). Как правило, это делается следующим образом:

```
for(var i=0; условие; шаг){

   команды

}
```

То есть мы объявляем новую переменную и сразу же задаем ей начальное значение. Переменная-счетчик может называться по-разному. Но за долгую практику программирования сложилась традиция давать переменным-счетчикам имена `i`, `j` или `k`.

Точно так же и с начальным значением данной переменной. Начальное значение может быть любым. Но принято начинать отсчет с нуля. Это особенно удобно при работе с массивами.

Переменная-счетчик не является локальной переменной цикла, то есть обратиться к данной переменной можно и за пределами цикла. В частности, поэтому переменной при инициализации присваивается значение. Ведь вполне возможно, что данная переменная уже использовалась ранее и уже содержит какое-либо значение.

Перейдем ко второму аргументу цикла `for` — к условию. Условие определяет, когда цикл будет завершен. Как правило, условие заключается в сравнении заданной в первом аргументе переменной с каким-либо конечным значением. Цикл выполняется до тех пор, пока условие истинно:

```
for(var i=0; i<10; шаг){

   команды

}
```

В данном примере цикл будет выполняться до тех пор, пока значение переменной `i` меньше 10. А поскольку изначально переменной `i` присвоено значение 0, без

третьего аргумента цикл будет выполняться бесконечно. Ведь внутри цикла нельзя изменять значение переменной, заданной в первом аргументе цикла `for`.

Итак, третий аргумент цикла `for` задает правило, по которому будет изменяться значение заданной в первом аргументе переменной при завершении очередного шага цикла. Как правило, третий аргумент указывается следующим образом:

```
for(var i=0; i<10; i++){

   команды

}
```

То есть при каждом окончании цикла значение переменной `i` будет увеличиваться на 1. После чего, если значение переменной `i` меньше 10, цикл будет повторяться заново.

Выйти из цикла можно либо при выполнении условия, заданного во втором аргументе цикла, либо с помощью ключевого слова `break`, которое мы уже рассматривали выше в пункте «Switch» подраздела «Условные операторы» данного раздела. Например:

```
for(var i=0; i<10; i++){

   if(условие){ break; } // выйти из цикла при истинном

                         // условии команды

}
```

При необходимости циклу `for` можно вообще не задавать никаких аргументов и выходить из такого бесконечного цикла с помощью ключевого слова `break`:

```
for(;;){

   if(условие){ break; } // выйти из цикла

                         // при истинном условии команды

}
```

Помимо `break`, внутри любых циклов можно использовать ключевое слово `continue`. Оно позволяет перейти к следующему шагу цикла, пропуская выполнение команд, расположенных ниже вызова `continue`. Например:

```
for(var i=0; i<10; i++){

   команды1
```

```
    continue;
    команды2
}
```

В данном примере команды2 не выполнятся никогда. Поэтому `continue` используют в сочетании с условным оператором. Если условие истинно/ложно, то выполняется команда `continue`.

В качестве примера рассмотрим, как цикл используется для прохода по массиву:

```
var arr = ["красный", "оранжевый", "желтый", "зеленый"]
for(var i=0; i<arr.length; i++){
    switch(arr[i]){
    case "оранжевый":
        команды1
        break;
    default:
        команды2
    }
}
```

В данном примере цикл завершится только при достижении конца массива (`i<arr.length`). И при каждой итерации цикла нам для вычислений доступен следующий элемент массива (`arr[i]`).

For in

Рассмотренный в конце предыдущего пункта цикл прохода по элементам массива идеально подходит для массивов, в которых индексы заданы последовательно. Но что делать, если в массиве некоторые индексы пропущены? Например, есть элементы с индексами 0, 13, 26, а остальных нет:

```
var mass = ["красный"];
mass[13] = "оранжевый";
mass[26] = "желтый";
```

В принципе, здесь также можно использовать рассмотренный выше цикл `for`. Но в данном случае он будет неэффективен. Ведь циклу придется совершить 27 операций только для того, чтобы проверить три элемента массива.

Куда эффективнее использовать специальную разновидность цикла `for`, предназначенную именно для прохода по массивам, объектам и другим последовательностям данных, — цикл `for in`.

Синтаксис цикла `for in` следующий:

```
for(var переменная in массив) {

   команды

}
```

Данный цикл проходит по всем элементам массива, помещая каждый из них в переменную `переменная`. При первом проходе массива в `переменной` находится первый элемент массива (не обязательно расположенный по индексу 0), при втором проходе — второй и т. д.

Например:

```
for(var item in mass) {

   window.alert(item);

}
```

В данном варианте цикла также можно использовать ключевые слова `break` и `continue`.

While

Помимо цикла `for`, существует еще одна часто используемая разновидность цикла — цикл `while`. Цикл `while` позволяет повторять выполнение набора команд до тех пор, пока не будет выполнено какое-либо условие.

Цикл `while` используется в том случае, если заранее невозможно определить, сколько проходов цикла потребуется. Цикл `while` имеет следующий синтаксис:

```
while(условие){

   команды

}
```

Цикл будет выполняться до тех пор, пока условие истинно, конечно, если вы не завершите цикл с помощью `break`. Довольно часто цикл `while` используется именно для того, чтобы задать цикл с выходом только с помощью ключевого слова `break`:

```
while(1==1){

    if(условие) break;

    команды

}
```

Следует также обратить внимание на то, что цикл `while` сначала проверяет условие и, только если оно истинно, выполняет команды. Поэтому команды из цикла могут быть выполнены определенное количество раз либо вообще ни разу. Это основное отличие цикла `while` от цикла `do while`.

Do while

Разновидностью цикла `while` является цикл `do while`. Он также позволяет повторять выполнение набора команд до тех пор, пока не будет выполнено какое-либо условие. Однако в случае использования цикла `do while` команды из цикла будут выполнены минимум один раз. Потому что цикл сначала выполняет содержащиеся в нем команды, а потом уже проверяет истинность условия.

Синтаксис цикла `do while`:

```
do{

    команды

}while(условие)
```

Функции

Функцией называется набор команд, которым присвоено какое-либо имя. Указав в коде имя функции, вы можете запустить выполнение набора команд с этим именем. Это позволяет упростить и сделать нагляднее процесс программирования.

То есть, если у вас есть некая последовательность команд, которую нужно выполнить несколько раз в различных участках кода, вы можете поступить разумно или неразумно:

- неразумно — написать всю последовательность команд в одном месте кода, потом написать ту же последовательность команд в другом месте, в третьем и т. д.;

- разумно — создать функцию, которая выполняла бы данную последовательность команд, после чего вызывать выполнение данной функции во всех местах вашего кода, где это необходимо.

В чем же неразумность первого способа? Во-первых, увеличивается объем кода. Соответственно, возрастает вероятность появления ошибок и ухудшается наглядность кода. Во-вторых, если вы захотите модифицировать повторяющийся блок команд, вам придется изменить его во всех местах кода, где он встречается. А если бы вы использовали возможности функций, модифицировать блок команд нужно было бы только один раз — внутри функции.

Создаем функцию. Итак, функции создаются следующим образом:

```
function имя(){

    команды

}
```

Функции имеют локальную область видимости, то есть все изменения, которые происходят внутри функции, не влияют на переменные снаружи функции. А кроме того, все переменные, объявленные внутри функции, недоступны снаружи и уничтожаются, как только завершается выполнение функции.

ПРИМЕЧАНИЕ

Область видимости — часть кода, в пределах которой можно получить доступ к данной переменной.

Глобальная область видимости — область видимости переменной в пределах всего сценария, то есть переменная видна в любом участке сценария, а также внутри любых функций, созданных в данном сценарии. Глобальную область видимости имеют все переменные, созданные в сценарии за пределами каких-либо функций.

По этой причине приведенный выше вариант создания функции малопригоден. Ведь результаты выполнения команд, записанных внутри функции, нигде не сохраняются.

Как правило, функции используются для того, чтобы произвести какое-либо вычисление, после чего вернуть результат этого вычисления. Чтобы вернуть из функции какое-либо значение, используется ключевое слово `return`:

```
function имя(){

    команды

    return значение;

}
```

Из функции можно вернуть только одно значение. При этом все команды, записанные после ключевого слова `return`, выполнены не будут. Как только из функции возвращается значение, выполнение функции завершается.

Ключевое слово `return` также используется для того, чтобы раньше времени завершить выполнение функции:

```
function имя(){
   команды1
   if(условие) return;
   команды2
}
```

Если условие истинно, выполнение функции будет завершено, и `команды2` выполнены не будут. При этом функция не вернет никакого значения, ибо само значение, которое нужно вернуть, не указано в ключевом слове `return` (но если нужно что-то вернуть, вы можете указать возвращаемое значение).

Поскольку функция имеет локальную область видимости, внутри нее недоступны переменные, которые были объявлены за пределами функции (на самом деле доступны, но прямо пользоваться ими крайне не рекомендуется). Поэтому перед тем, как что-то выполнить и вернуть какой-либо результат, в функцию нужно передать исходные данные.

ПРИМЕЧАНИЕ

Однако функция, созданная внутри другой функции, видит переменные внешней функции. Причем видит она их даже тогда, когда внешняя функция завершила свое исполнение. Это называется замыканием.

Набор значений, которые понадобятся для вычислений внутри функции, передается в функцию с помощью аргументов:

```
function имя(аргумент1, аргумент2, аргумент3){
   команды1
   return значение;
}
```

Аргументы, которые необходимо передать в функцию для ее выполнения, перечисляются через запятую внутри круглых скобок. Каждый аргумент представляет собой переменную с определенным именем — под данным именем переданное значение будет доступно внутри функции. Например:

```
var num = 34;

function myfunc(arg1){

   if(arg1==34) return true;

   return false;

}
```

Если в функцию `myfunc` будет передано значение 34, то функция вернет логическое значение `true`. В противном случае функция вернет `false`.

Вызываем функцию. Допустим, мы создали функцию. Как же нам теперь ее вызвать?

Если функция не имеет аргументов и не возвращает никакого значения, то вызвать ее проще всего. Достаточно воспользоваться командой `имя_функции()`. Например:

```
myfunc();


function myfunc(){

   window.alert("hello");

}
```

В данном примере можно заметить одну особенность использования функций. Функцию можно вызвать до того, как она будет создана. Чаще всего так и делают: создают функции в конце сценария, чтобы они не мешали чтению кода.

Если для работы функции необходимы какие-либо аргументы, функцию вызывают следующим образом: `имя_функции(значение_для_аргумента1, значение_для_аргумента2)`. Например:

```
var num = 34;

myfunc(num);


function myfunc(arg1){

   window.alert("Вы передали в функцию значение "+arg1);

}
```

Сколько аргументов было указано при создании функции, столько и нужно передать в нее при вызове. Не больше и не меньше. Иначе возникнет ошибка.

Если функция возвращает какое-либо значение, возвращаемое значение нужно присвоить какой-либо переменной:

```
var num = 34;

var res = myfunc(num, 23);


function myfunc(arg1, arg2){

   if(arg1>arg2) return true;

   return false;

}
```

Создаем функцию оператором new. Помимо рассмотренного выше способа создания функции, в JavaScript существует еще один. Функция в JavaScript является одним из типов переменных (объектом класса `Function`), а значит, ее можно создать таким же образом, как и переменную любого другого типа:

```
var имя_функции = new Function (команда);
```

Команда является строкой команд. Например:

```
var myfunc = new Function("window.alert('111');");

myfunc();
```

Так мы создали функцию без каких-либо аргументов. Чтобы создать функцию с аргументами, их нужно перечислить через запятую перед командой создаваемой функции:

```
var имя_функции = new Function (аргумент1, аргумент2, команда);
```

При этом имя каждого аргумента также является строкой:

```
var myfunc = new Function("arg1", "arg2", "if(arg1>arg2) return
true; return false;");


var num = 34;

var res = myfunc(num, 23);
```

Данный способ создания функции имеет одно отличие от рассмотренного ранее. При объявлении функции оператором `new` функцию сначала нужно объявить (создать) и только потом использовать. Иначе возникнет ошибка:

```
var num = 34;

var res = myfunc(num, 23); // ОШИБКА, такой функции ЕЩЕ НЕТ


var myfunc = new Function("arg1", "arg2", "if(arg1>arg2) return
true; return false;");
```

Функции с переменным количеством аргументов. В жизни каждого разработчика рано или поздно возникает необходимость в функции, которая могла бы принимать различное количество аргументов, чтобы можно было при необходимости вызвать функцию с одним аргументом, двумя, тремя и любым другим количеством аргументов.

Создавать для этого множество отдельных функций, каждая из которых принимает различное количество аргументов, было бы не самым разумным решением. Ведь в JavaScript существует и более простой способ решить поставленную задачу.

Прежде всего следует заметить, что в JavaScript совершенно не обязательно вызывать функцию с тем количеством аргументов, которые были указаны при ее объявлении. Вы можете создать функцию с тремя аргументами, а при вызове указать только два. И ошибки не будет. Но, конечно, не факт, что функция будет работать так, как вам нужно.

Точно так же вы можете вообще не объявить аргументов при создании функции. И тем не менее при вызове функции указывать для нее аргументы:

```
var summ = test(34, 56);

// ошибки несоответствия количества аргументов не возникнет

function test(){

   команды

}
```

Другое дело, что мы пока не знаем способов, как можно обратиться к аргументам функции, которые не были объявлены при создании этой функции. Например, как внутри функции, приведенной в примере выше, обратиться к первому аргументу функции `test`?

В JavaScript внутри любой функции можно обратиться к массиву `arguments`. Данный массив не нужно создавать. Он уже создан при вызове любой функции. И он содержит аргументы, которые были указаны при текущем вызове функции. Каждый аргумент является отдельным элементом данного массива. Если аргументов передано не было, массив `arguments` будет пустым.

Как вы уже знаете, получить длину массива можно с помощью свойства `length`: `arguments.length`. А обратиться ко всем элементам массива можно из цикла. Так что же нам мешает создать функцию с переменным количеством аргументов:

```
var s1 = test(); // s1 равно 0

var s2 = test(10); // s2 равно 10

var s3 = test(10, 20); // s3 равно 30

var s4 = test(10, 20, 50); // s4 равно 80


function test(){

    var summ=0;

    if(arguments.length){

        for(var i=0; i<arguments.length; i++){

            summ+=Number(arguments[i]);

        }

    }

    return summ;

}
```

Работаем с DOM

До сих пор мы не нашли ни одного способа, как из JavaScript можно влиять на отдельные элементы веб-страницы. А ведь способы должны быть. Иначе зачем вообще внедрять в HTML-документ код JavaScript?

Способы действительно есть. И все они собраны в одно понятие — объектная модель DOM (Document Object Model). Методы и свойства данной модели как раз и предназначены для того, чтобы производить какие-либо действия над веб-страницей.

Модель DOM представляет веб-документ в виде дерева тегов. Тег в этом дереве называется узлом-элементом. А содержимое тега (текст) — текстовым узлом. Атрибуты тегов также считаются узлами дерева.

Узлы-элементы и текстовые узлы — это типы элементов. Вообще, в модели DOM существует 12 различных типов элементов. Но вы будете встречаться именно с этими двумя, так как остальные в JavaScript не используются.

Как обычно, в разных браузерах дерево модели DOM может различаться, как и свойства с методами, которые поддерживают определенные элементы модели DOM. С этим, как всегда, придется мириться.

К каждому тегу в дереве можно обратиться, чтобы получить или изменить его содержимое либо выполнить какую-либо другую операцию. Все это делается с помощью свойств и методов DOM. Но перед тем, как использовать свойства и методы, нам следует узнать об объектах, которые поддерживают эти свойства и методы. Ведь, как мы уже знаем, свойства и методы пишутся с помощью точечной нотации после объекта, к которому они относятся.

Объекты DOM

Модель DOM состоит из множества объектов, которые объединены в дерево объектов. На рис. 3.7 представлена часто используемая часть этого дерева.

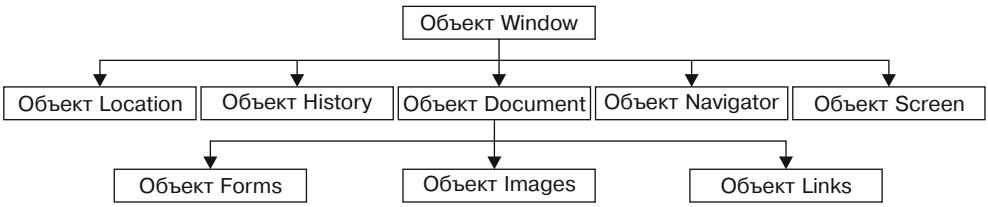


Рис. 3.7. Часто используемая часть дерева объектов DOM

ПРИМЕЧАНИЕ

На самом деле к DOM (объектная модель документа) относится только один объект — document. Все остальные объекты относятся к BOM (Browser Object Model, объектная модель браузера). Но в формате данной книги это несущественно.

Что же в практическом плане означает эта иерархия между объектами DOM? Прежде всего то, что объекты, расположенные ниже в дереве объектов DOM, нельзя вызвать без указания объектов, которые расположены выше. То есть, чтобы обратиться к объекту `forms`, нужно сначала обратиться к объекту `window`, потом через точечную нотацию — к объекту `document`, и только потом — к `forms`: `window.document.forms`.

Поскольку объект `window` в JavaScript подразумевается изначально (глобальный объект), при желании обращение к нему можно опустить. Тогда для доступа к свойствам и методам объекта `forms` достаточно будет указать строку `document.forms`.

Каждый объект в DOM отвечает за свою «часть» веб-страницы: один за формы на веб-странице, другой за картинки на ней, третий за саму веб-страницу, четвертый за экран браузера и т. д. Они содержат свойства и методы, предназначенные для управления этой «частью» веб-страницы.

Window

Строка обращения: `window`.

Данный объект находится на самой вершине иерархии объектов DOM. Он отвечает за само окно браузера и все, что с ним связано.

Поскольку объект `window` является глобальным, для доступа к его свойствам и методам можно опустить точечную нотацию: `alert("строка")` вместо `window.alert("строка")`.

В табл. 3.13, 3.14 приведены свойства и методы объекта `Window`.

Таблица 3.13. Свойства объекта Window

Свойство	Описание
defaultStatus	Отображаемое по умолчанию в строке состояния окна браузера сообщение
status	Текущее сообщение, показываемое в строке состояния окна браузера
frames	Массив всех фреймов окна
length	Количество фреймов в родительском окне
name	Имя окна, указанное при его открытии методом open
parent	В HTML-документах, состоящих из набора фреймов, указывает на родительское окно
self и window	Синонимы имени окна
top	В HTML-документах, состоящих из набора фреймов, указывает на окно верхнего уровня

Таблица 3.14. Методы объекта Window

Метод	Описание
alert	Отображает диалоговое сообщение с кнопкой OK
open	Открывает окно
close	Закрывает окно

Метод	Описание
confirm	Отображает диалоговое сообщение с кнопками OK и Отмена
prompt	Отображает диалоговое сообщение с полем ввода
setTimeout	Устанавливает таймер
clearTimeout	Отменяет таймер

Location

Строка обращения: `window.location`.

Данный объект хранит информацию о текущей веб-странице: адрес сайта, адрес страницы и т. д.

В табл. 3.15, 3.16 приведены свойства и методы объекта `Location`.

Таблица 3.15. Свойства объекта Location

Свойство	Описание
href	Полный URL-адрес текущей веб-страницы
hash	Имя «якоря» (закладки) в URL-адресе веб-страницы
host	Часть URL-адреса страницы, содержащая имя сервера и порт
hostname	Имя сервера, с которого загружена текущая веб-страница
pathname	Путь к файлу на сервере (без имени сервера и порта)
port	Номер порта, через который происходит обращение к веб-странице
protocol	Протокол передачи данных (HTTP, FTP)
search	Строка GET-параметров, которая идет после знака “?”

Таблица 3.16. Методы объекта Location

Метод	Описание
assign	Открывает указанную страницу вместо текущей
reload	Обновляет текущую веб-страницу
replace	Открывает указанную страницу вместо текущей. При этом в истории браузера адрес текущего документа заменяется адресом нового

History

Строка обращения: `window.history`.

Содержит информацию о страницах сайта, которые пользователь посетил до этого. Список этих страниц называется стеком истории браузера.

В табл. 3.17, 3.18 приведены свойство и методы объекта `History`.

Таблица 3.17. Свойство объекта History

Свойство	Описание
length	Определяет количество посещенных пользователем веб-страниц (количество URL в текущей истории браузера)

Таблица 3.18. Методы объекта History

Метод	Описание
go	Загружает веб-страницу с указанным номером. Номера страниц отсчитываются от текущей: текущая страница имеет индекс 0, предыдущая — –1 и т. д.
back	Загружает предыдущую веб-страницу (эквивалентно go(-1))
forward	Загружает следующую веб-страницу из списка истории, если таковая имеется (эквивалентно go(1))

Navigator

Строка обращения: `window.navigator`.

Содержит информацию о браузере и операционной системе пользователя, который в данный момент открыл веб-страницу.

В табл. 3.19 приведены свойства объекта `Navigator`.

Таблица 3.19. Свойства объекта Navigator

Свойство	Описание
appCodeName	Содержит кодовое имя (платформу, «движок») браузера
appName	Содержит название браузера (Netscape, Microsoft Internet Explorer)
appVersion	Отображает версию браузера, платформу, выпуск браузера и наименование операционной системы пользователя
userAgent	Возвращает строку, идентифицирующую браузер пользователя (комбинация значений свойств appCodeName и appVersion)
cookieEnabled	Определяет, разрешено или нет использование cookies (если разрешено, возвращает true)
browserLanguage	Содержит текущий язык браузера
systemLanguage	Отображает код языка операционной системы пользователя по умолчанию
userLanguage	Содержит пользовательские настройки языка операционной системы
platform	Указывает платформу операционной системы пользователя (например, Win32)
cpuClass	Содержит тип процессора на компьютере пользователя (например, x86 или Alpha)

Screen

Строка обращения: `window.screen`.

Содержит информацию о мониторе пользователя, который в данный момент открыл веб-страницу. Чаще всего разработчиками используется информация о разрешении пользователя (ширине и высоте экрана пользователя в пикселах).

В табл. 3.20 приведены свойства объекта `Screen`.

Таблица 3.20. Свойства объекта Screen

Свойство	Описание
width	Содержит ширину экрана монитора в пикселах
height	Отображает высоту экрана монитора в пикселах
availHeight	Указывает высоту полезной области экрана монитора (без Панели задач и других элементов графического интерфейса ОС)
availWidth	Содержит ширину полезной области экрана монитора (без Панели задач и других элементов графического интерфейса ОС)
colorDepth	Отображает установленную глубину цвета (для 256 цветов возвращается 8, для 16,7 млн цветов (режим High Color) — возвращается 32)
updateInterval	Указывает интервал времени (в миллисекундах) между обновлениями экрана

Document

Строка обращения: `window.document`.

Содержит информацию о самой веб-странице, которая в данный момент открыта в окне браузера.

В табл. 3.21, 3.22 приведены свойства и методы объекта `Document`.

Таблица 3.21. Свойства объекта Document

Свойство	Описание
alinkColor	Определяет цвет ссылок, выбранных пользователем
linkColor	Указывает цвет еще не посещенных ссылок, размещенных в HTML-документе
vlinkColor	Определяет цвет уже посещенных ссылок, размещенных в HTML-документе
bgColor	Задает цвет фона тега body в шестнадцатеричном виде либо с помощью ключевых слов
fgColor	Задает цвет текста в документе в шестнадцатеричном виде либо с помощью ключевых слов
lastModified	Содержит дату последнего изменения HTML-документа

Продолжение ⇩

Таблица 3.21 *(продолжение)*

Свойство	Описание
location	Содержит полный URL-адрес текущей веб-страницы
referrer	Указывает полный URL-адрес страницы, с которой была открыта данная веб-страница
title	Содержит заголовок документа (содержимое тега title)
URL	Определяет полный URL-адрес HTML-документа

Таблица 3.22. Методы объекта Document

Метод	Описание
getElementsByName ({Имя_элемента})	Возвращает массив тегов, для которых атрибут name имеет указанное значение
getElementById ({ID})	Возвращает тег с указанным идентификатором
getElementsByTag-Name({Имя тега})	Возвращает массив тегов веб-страницы, имя которых передано в качестве аргумента метода
write	Записывает текст или HTML-код в текущее место документа

Node

Свойства и методы данного типа выполняются над конкретным тегом (узлом) выб-страницы, то есть над тем, что возвращается методами `getElementsByName`, `getElementById`, `getElementsByTagName` объекта `Document`.

В табл. 3.23, 3.24 приведены свойства и методы объекта `Node`.

Таблица 3.23. Свойства Node

Свойство	Описание
nodeName	Имя текущего узла (только для чтения)
nodeValue	Значение текущего узла (только для чтения)
nodeType	Тип объекта текущего узла (только для чтения)
parentNode	Возвращает родительский узел текущего узла
childNodes	Возвращает список узлов, содержащий всех потомков текущего узла
firstChild	Возвращает первый дочерний узел текущего узла
lastChild	Возвращает последний дочерний узел текущего узла
previousSibling	Возвращает предшествующий узел
nextSibling	Возвращает последующий узел
attributes	Возвращает коллекцию узлов, содержащую все атрибуты данного узла

Свойство	Описание
ownerDocument	Объект Document, связанный с текущим узлом
namespaceURI	URI пространства имен данного узла (только для чтения)
prefix	Префикс пространства имен данного узла
localName	Локальная часть квалифицированного имени текущего узла (только для чтения)
baseURI	Возвращает адрес документа (только для чтения)
textContent	Возвращает текстовое содержимое текущего узла

Таблица 3.24. Методы Node

Метод	Описание
insertBefore (newChild, refChild)	Вставляет новый узел newChild перед существующим дочерним узлом refChild
replaceChild (newChild, oldChild)	Заменяет и возвращает существующий дочерний узел oldChild новым узлом newChild
removeChild (oldChild)	Удаляет и возвращает дочерний узел oldChild из списка потомков
appendChild (newChild)	Добавляет в конец списка потомков текущего узла новый узел newChild
hasChildNodes()	Возвращает true, если текущий узел имеет потомков
cloneNode(deep)	Возвращает копию данного узла. Если аргумент метода равен true, то копируются и текущий узел, и его потомки
normalize()	Нормализует текущий узел (перемещает текстовые узлы вглубь поддерева текущего узла)
isSupported(feature, version)	Возвращает true, если текущий узел поддерживает возможность feature версии version
hasAttributes()	Возвращает true, если текущий узел имеет атрибуты

Forms

Строка обращения: `window.document.forms`.

Является коллекцией и содержит массив форм, которые есть на веб-странице, открытой в данный момент в окне браузера.

ПРИМЕЧАНИЕ

Коллекцией называется массив объектов, проиндексированный как по числовым номерам элементов, так и по их именам.

В табл. 3.25, 3.26 приведены свойства и методы объекта `Forms`.

Таблица 3.25. Свойства объекта Forms

Свойство	Описание
action	Задает URL-адрес сценария на сервере, которому отправляется форма
enctype (encoding)	Указывает тип MIME-кодирования формы
elements[]	Массив, содержащий все элементы формы
length	Определяет количество элементов формы
method	Определяет способ отправки формы серверу
name	Содержит значение атрибута name формы
target	Определяет окно назначения браузера, в котором должен отображаться результат отправки формы (имя фрейма, либо ключевые слова _self, _blank, _top и _parent)

Таблица 3.26. Методы объекта Forms

Метод	Описание
reset()	Сбрасывает значения всех элементов формы в исходные состояния
submit()	Программно отправляет форму на сервер
onReset	Событие, возникающее при щелчке на кнопке Сбросить или при выполнении метода reset()
onSubmit	Событие, возникающее при щелчке на кнопке Отправить или при вызове метода submit()

Images

Строка обращения: `window.document.images`.

Является коллекцией и содержит массив изображений, которые есть на веб-странице, открытой в данный момент в окне браузера.

Links

Строка обращения: `window.document.links`.

Является коллекцией и содержит массив ссылок, которые есть на веб-странице, открытой в данный момент в окне браузера.

Примеры

Простое перечисление свойств и методов бывает полезным, если вы и так прекрасно знаете, как все это работает. В противном случае нам никак не обойтись без множества примеров. Однако в формате данной книги модели DOM будет уделено гораздо меньше внимания, чем возможностям jQuery, о которых мы поговорим далее. Так что примеров будет не множество, а лишь несколько.

Доступ к элементам веб-страницы

Чтобы работать с конкретным тегом на веб-странице (изменять его атрибуты, добавлять свойства CSS, изменять содержимое тега и т. д.), вам сначала необходимо получить к нему доступ, то есть каким-либо образом обратиться к нужному тегу на веб-странице.

Дерево элементов веб-страницы доступно из объекта `document`.

Корневым элементом любого дерева правильно сверстанной веб-страницы является тег `html`. Получить доступ к вершине дерева (то есть к тегу `html`) можно с помощью вызова `document.documentElement`. Однако вместо вершины дерева вы можете обратиться сразу к тегу `body`. Это делается с помощью вызова `document.body`.

Дальнейшая навигация по веб-странице может выполняться различными способами.

Массив childNodes. К любому ребенку текущего элемента веб-страницы можно обратиться с помощью массива `childNodes`, который как раз и содержит в себе всех детей данного элемента (но не детей этих детей). Это делается следующим образом: `document.body.childNodes[индекс]`.

Как и любой массив, `childNodes` поддерживает свойство `length`, позволяющее определить текущее количество элементов в массиве. Так что для перебора всех элементов массива мы можем воспользоваться следующим кодом:

```
for(var i=0; i<document.body.childNodes.length; i++) {

   var child = document.body.childNodes[i]

   window.alert(child.tagName)

}
```

ПРИМЕЧАНИЕ

Данный цикл будет работать только в том случае, если веб-страница полностью загружена. Поэтому в рабочих проектах этот цикл нужно создавать в функции, после чего назначать выполнение этой функции событию onload тега body либо какому-нибудь другому событию, наступающему только после полной загрузки страницы.

Как мы уже знаем, внутри дерева DOM могут находиться элементы 12 различных типов. Хотя из JavaScript можно получить доступ только к двум из них: тегу и тексту внутри тега.

Чтобы определить тип текущего элемента модели DOM, нужно воспользоваться свойством `nodeType`. Если оно возвращает значение `1`, значит, текущий элемент является тегом. Если же оно возвращает значение 3, значит, перед нами текст.

Таким образом, чтобы перебрать не только дочерние элементы тега `body`, но и дочерние элементы этих дочерних элементов, нужно расширить наш код следующим образом:

```
    for(var i=0; i<document.body.childNodes.length; i++) {
      var child = document.body.childNodes[i];
        if(child.nodeType == 1){
            window.alert(child.tagName);
            for(var j=0; j<child.childNodes.length; j++) {
                  var child2 = child.childNodes[j];
                if(child2.nodeType == 1){
                  window.alert(child2.tagName);
                }
            }
        }
    }
```

Если еще немножко модифицировать данный код, то в конце концов мы сможем обратиться ко всем элементам на веб-странице:

```
function getallchild(el){
    for(var i=0; i<el.childNodes.length; i++) {
      var child = el.childNodes[i];
        if(child.nodeType == 1){
            window.alert(child.tagName);
            getallchild(child);
        }
    }
}
```

То есть мы создали функцию, которая вызывает сама себя до тех пор, пока не будут перебраны все элементы на веб-странице. Функции, вызывающие сами себя, называются рекурсивными. При их создании самое главное — не превратить вызов в цикл. Иначе функция будет вызывать сама себя до конца света. Но в нашем примере этого, кажется, не случилось.

Нам осталось лишь запустить данную функцию, указав в качестве ее аргумента тег, начиная с которого нужно выполнить перебор. Например:

```
<body onload="getallchild(document.body);">
```

firstChild. Свойство `firstChild` позволяет получить доступ к первому дочернему элементу текущего элемента. Если детей у элемента нет, свойство `firstChild` равно `null`.

lastChild. Свойство `lastChild` дает возможность получить доступ к последнему дочернему элементу текущего элемента. Если детей у элемента нет, свойство `lastChild` равно `null`.

Например, `document.body.lastChild.nodeType`.

previousSibling. Данное свойство позволяет обратиться к элементу дерева, который расположен слева от текущего элемента (расположен над данным тегом на веб-странице). То есть с помощью данного свойства можно обратиться к предыдущему брату текущего тега среди всех дочерних тегов родительского тега.

nextSibling. Данное свойство дает возможность обратиться к элементу дерева, который расположен справа от текущего элемента (расположен под данным тегом на веб-странице). То есть с помощью данного свойства можно обратиться к следующему брату текущего тега среди всех тегов родительского тега.

parentNode. Данное свойство позволяет обратиться к родительскому тегу текущего элемента веб-страницы, то есть вернуться на один шаг назад по дереву DOM.

Чтобы лучше понять действие перечисленных свойств, посмотрите на рис. 3.8.

getElementById. Перечисленные выше свойства позволяют последовательно получить доступ к любому дочернему тегу внутри веб-страницы. Но последовательный перебор вложенных тегов довольно утомителен для разработчика. Куда проще было бы просто указать идентификатор нужного тега. Что и можно сделать с помощью данного метода.

С помощью метода `getElementById` можно сразу обратиться к любому тегу на веб-странице, для которого установлен атрибут `id`. Для этого достаточно воспользоваться вызовом: `document.getElementById("идентификатор")`.

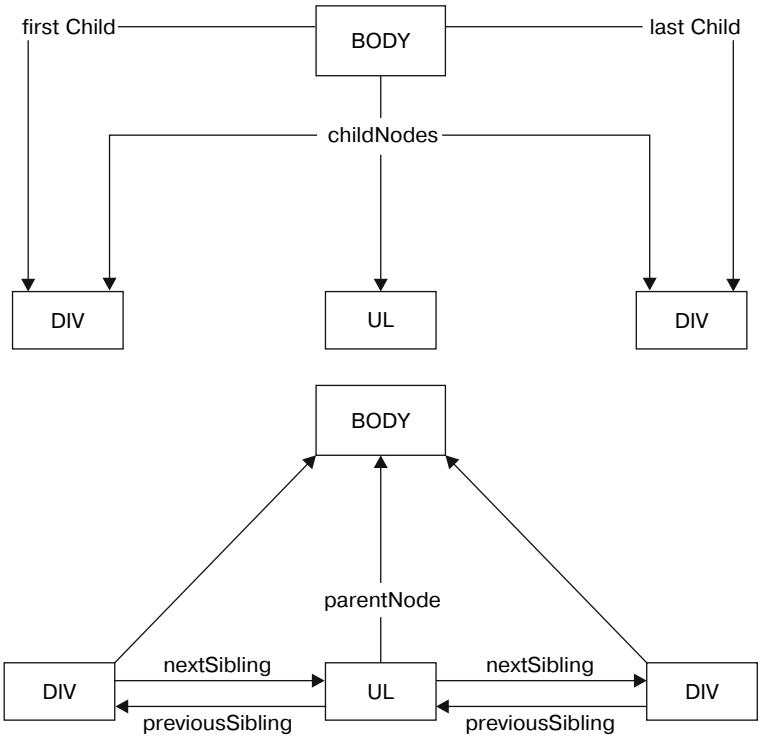


Рис. 3.8. Действие свойств для доступа к элементам

getElementsByTagName. С помощью модели DOM можно также последовательно обратиться ко всем тегам с заданным именем на веб-странице.

Например, чтобы перебрать в сценарии все теги `li`, которые есть на веб-странице, достаточно воспользоваться записью вида:

```
document.getElementsByTagName('li')
```

Если же вы хотите получить все теги `li`, которые есть внутри тега с определенным идентификатором, просто воспользуйтесь записью:

```
document.getElementById("идентификатор").getElementsByTagName
('li')
```

В любом случае метод `getElementsByTagName` возвращает массив, содержащий ссылки на все найденные теги. И работать с результатом выполнения данного метода нужно точно так же, как с массивами. В частности, чтобы обратиться к первому найденному тегу `li` на веб-странице, нужно сделать следующее:

```
document.getElementsByTagName('li')[0]
```

С помощью метода `getElementsByTagName` можно также получить всех потомков определенного тега, то есть все теги, которые находятся внутри него. Для этого достаточно в качестве аргумента метода `getElementsByTagName` указать значение `*`:

```
document.getElementsByTagName('*').
```

getElementsByName. Как вы знаете из главы 1, посвященной HTML, всем элементам формы присваивается атрибут `name`, значение которого должно быть уникальным в пределах текущего тега `form`.

Возможности JavaScript позволяют получить массив тегов, которым присвоен атрибут `name` с указанным значением. Это делается с помощью метода `getElementsByName`. Однако следует учитывать, что данный метод будет работать только с теми тегами, для которых в спецификации HTML явно указана поддержка атрибута `name`: `form`, `input`, `a`, `select`, `textarea` и с некоторыми другими.

Собственно говоря, работа с данным методом происходит точно так же, как и с рассмотренным ранее методом `getElementsByTagName`:

```
document.getElementsByName('mylogin')

document.getElementById("идентификатор").getElementsByName('my
login')

document.getElementsByName('mylogin')[0]
```

this в обработчиках событий. При работе с событиями довольно часто возникает необходимость получить доступ к тегу, для которого было назначено данное событие. Это можно сделать с помощью рассмотренных ранее свойств модели DOM. Но куда проще воспользоваться специальным оператором `this`. Данный оператор как раз указывает на объект, вызвавший текущее событие.

Работаем со свойствами CSS

Еще одной полезной возможностью JavaScript является возможность устанавливать новые CSS-свойства либо изменять значения существующих свойств для любого тега на веб-странице.

ПРИМЕЧАНИЕ

Значение свойства, которое в CSS было объявлено с указанием ключевого слова !important, изменить нельзя.

Чтобы установить CSS-свойство, нужно воспользоваться DOM-свойством `style`, которое поддерживается всеми элементами веб-страницы. То есть нам сначала

требуется получить доступ к нужному элементу веб-страницы и только потом вызвать свойство `style` для него. Например, так:

```
document.getElementById("идентификатор").style
```

Применить какое-либо CSS-свойство к тегу с данным `идентификатором`.

Или так:

```
document.body.firstChild.style
```

В любом случае после DOM-свойства `style` через точечную нотацию нужно указать CSS-свойство, значение которого мы хотим изменить:

```
document.getElementById("идентификатор").style.width
```

При этом, если в названии CSS-свойства есть дефисы, они удаляются, а первая буква слов, написанных после дефиса, делается прописной:

```
document.getElementById("идентификатор").style.zIndex
```

```
document.body.firstChild.style.backgroundImage
```

Так мы просто обратились к CSS-свойству. Чтобы изменить значение CSS-свойства, достаточно выполнить операцию присваивания:

```
document.getElementById("идентификатор").style.width="100px";
```

```
document.getElementById("идентификатор").style.zIndex=300;
```

```
document.body.firstChild.style.backgroundImage="url('../img/
mypict.png')";
```

Если вы хотите получить текущее значение CSS-свойства, достаточно поставить вызов свойства `style` справа от оператора присваивания:

```
var w = document.getElementById("идентификатор").style.width;
```

Например, напишем простой сценарий, случайным образом изменяющий цвет фона при щелчке кнопкой мыши на теге `div`:

```
<head>
    <title>Заголовок</title>
      <meta http-equiv="content-type" content="text/html;
charset=UTF-8" />
     <script type="text/javascript" src="js/addons.js"> </
script>
```

```
    <script type="text/javascript">

    function changecolor(obj){

obj.style.backgroundColor="rgb("+parseInt(Math.random()*254)+
","+parseInt(Math.random()*254)+","+parseInt(Math.
random()*254)+")";

    }

    </script>

</head>

<body>

<div onclick="changecolor(this);" style="width: 300px; height:
300px; cursor: pointer;">Нажми меня!</div>

</body>
```

Изменяем содержимое тега

Возможности JavaScript позволяют не только назначать тегам новые свойства CSS. С помощью JavaScript можно также изменять содержимое любого тега. Для этого используется свойство `innerHTML`, которое поддерживается всеми элементами веб-страницы.

innerHTML. Свойство `innerHTML` доступно как для чтения, так и для записи. То есть вы можете не только изменять содержимое тега, присваивая данному свойству новое значение, но и получать текущее содержимое тега, присваивая свойство `innerHTML` какой-либо переменной.

Стоит обратить внимание еще на одну особенность свойства `innerHTML`. Оно понимает не только обычный текст, но и любые HTML-теги. То есть с помощью данного свойства вы можете добавлять в HTML-документ не просто обычный текст, но и любые блоки тегов.

Итак, чтобы получить текущее содержимое тега, нужно использовать запись вида:

```
var w = document.getElementById("идентификатор").innerHTML;
```

Если же вы хотите изменить содержимое тега, достаточно воспользоваться записью:

```
document.getElementById("идентификатор").innerHTML="текст";
```

Дополнить содержимое тега новым значением можно с помощью операции конкатенации:

```
document.getElementById("идентификатор").innerHTML+="текст";
```

Например, создадим сценарий, выводящий на экран все, что пользователь пишет в текстовом поле (рис. 3.9):

```
<body>

<input type="text" size="77" value="" onchange="document.getE
lementById('thistext').innerHTML+=' '+this.value;" />

<br /><br /><div id="thistext"></div>

</body>
```

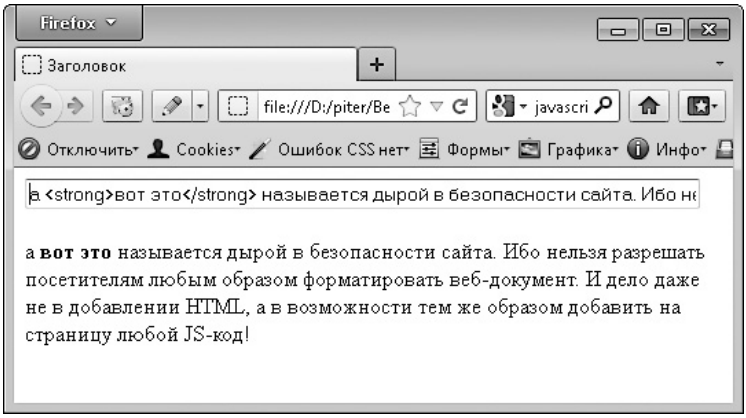


Рис. 3.9. Изменяем содержимое тега

outerHTML. Свойство `innerHTML` помещает текст внутрь данного тега. Тогда как свойство `outerHTML`, которое также поддерживается JavaScript, позволяет поместить текст вместо данного тега.

Что это значит в практическом плане, лучше понять на примерах. Предположим, у нас есть следующий кусок HTML-кода — `<div id="mtext">test</div>`. Тогда:

```
document.getElementById("mtext").innerHTML="текст";

// <div id="mtext">текст</div>

document.getElementById("mtext").outerHTML="текст";

// текст (тег с идентификатором mtext удален с веб-страницы)
```

```
document.getElementById("mtext").innerHTML+="текст";

//<div id="mtext">testтекст</div>

document.getElementById("mtext").outerHTML+="текст";

//<div id="mtext">test</div>текст
```

В остальном свойство `outerHTML` аналогично свойству `innerHTML`.

Изменяем атрибуты тега

С помощью JavaScript можно добавлять, удалять и изменять атрибуты тега. Для этого используются либо специальные методы `setAttribute`, `getAttribute`, `hasAttribute` и `removeAttribute`, либо прямое обращение к атрибуту.

setAttribute. Для установки атрибута служит метод `setAttribute(имя, значение)`, который позволяет присвоить значение `значение` атрибуту `имя`. Например:

```
document.getElementById("myimg").setAttribute("alt","лого");
```

getAttribute. Если вы хотите получить текущее значение атрибута, достаточно воспользоваться методом `getAttribute(имя)`. Например:

```
var myattr = document.getElementById("myimg").
getAttribute("alt");
```

hasAttribute. Однако перед тем, как использовать метод `getAttribute`, желательно проверить, существует ли вообще нужный атрибут у данного тега. Это делается с помощью метода `hasAttribute(имя)`:

```
if(document.getElementById("myimg").hasAttribute("alt")){

   var myattr = document.getElementById("myimg").
getAttribute("alt");

}else{

   document.getElementById("myimg").setAttribute("alt","да");

}
```

removeAttribute. И последний метод для работы с атрибутами — `removeAttribute(имя)` — позволяет удалить атрибут у данного тега. Перед тем как удалять атрибут, удостоверьтесь, что он существует:

```
if(document.getElementById("myimg").hasAttribute("alt")){

   document.getElementById("myimg").removeAttribute("alt");

}
```

Прямое обращение к атрибуту. В модели DOM для доступа к основным стандартным атрибутам можно использовать не только перечисленные выше методы, но и прямое обращение к атрибуту, то есть:

```
document.getElementById("идентификатор").атрибут
```

Например:

```
document.body.id = "mytheme";
```

То есть, обратившись к объекту в модели DOM, вы просто указываете с помощью точечной нотации имя атрибута, с которым хотите работать. Однако из этого правила есть исключения:

- атрибут `class` — для доступа к данному атрибуту используется не свойство `class`, а свойство `className`;
- атрибут `for` — в Internet Explorer для доступа к данному атрибуту применяется свойство `forHtml`.

Следует также обратить внимание на то, что имена атрибутов при указании их в качестве свойств DOM являются регистрозависимыми, то есть:

```
document.body.id = "mytheme1";

document.body.ID = "mytheme2";
```

В результате выполнения данного кода атрибут `id` тега `body` будет иметь значение `mytheme1`. А не `mytheme2`, как можно было предположить.

Создание и удаление тегов

Вы уже научились изменять содержимое тегов, их атрибуты и назначенные им CSS-свойства. Осталось только научиться создавать и удалять теги и можно вообще забросить HTML и CSS. Зачем они нужны, если все можно сделать с помощью JavaScript...

Это, конечно, шутка. Любую технологию нужно использовать по своему прямому назначению. Язык HTML — для создания каркаса веб-страницы. Язык CSS — для визуализации. А JavaScript — для добавления интерактивности.

Создание. Для создания новых тегов предназначен метод `createElement` объекта `document`. Просто укажите в качестве аргумента этого метода тег, который нужно создать. И не забудьте сохранить возвращенное значение в какой-либо переменной:

```
var newDiv = document.createElement('div');
```

Тег создан и находится в указанной переменной `newDiv`. Теперь мы можем добавить тегу нужные атрибуты:

```
newDiv.className = 'myclass';

newDiv.id = 'myid';
```

Мы даже умеем присваивать тегу свойства CSS:

```
newDiv.style.backgroundColor = 'white';
```

И, конечно, совсем недавно мы научились добавлять тегу содержимое:

```
newDiv.innerHTML = 'Наш новый тег';
```

Так что теперь мы знаем все, что может понадобиться для создания новых тегов. Осталось только научиться выводить созданный тег на веб-страницу.

Публикация. После того как тег создан, его нужно добавить на веб-страницу. Для этого сначала нужно определить, куда вы хотите его добавить: то есть после или перед каким из уже существующих на веб-странице тегов вы хотите добавить созданный тег.

И, в конце концов, нам останется лишь воспользоваться одним из следующих методов:

- `appendChild(новый тег)` — добавить тег в конец данного элемента (последним в списке детей этого элемента);
- `insertBefore(новый тег, перед каким тегом вставить)` — добавить новый тег перед дочерним тегом, который был указан в качестве второго аргумента метода.

Например:

```
var newDiv = document.createElement('div');

document.getElementById('oldDiv').appendChild(newDiv);
```

Или:

```
var newLi = document.createElement('li');
```

```
document.getElementById('listUL').insertBefore(newLi, document.
getElementById('listUL').getElementsByTagName('li')[0]);
```

Удаление. Для удаления тега с веб-страницы используется метод `removeChild(элемент)`. Вызывать данный метод нужно у родителя тега, который вы хотите удалить: `родитель.removeChild(ребенок)`. Например:

```
document.getElementById('listUL').removeChild(document.getElement-
ById('listUL').getElementsByTagName('li')[0]);
```

Но если ссылки на родителя тега нет, можно воспользоваться услугами промежуточного свойства `parentNode`, которое возвращает родителя элемента:

```
document.getElementById('oldDiv').parentNode.removeChild(docu-
ment.getElementById('oldDiv'));
```

Работа с таймерами

В JavaScript существует еще одна интересная возможность, связанная с функциями. Вы можете запустить функцию через указанное время либо вообще назначить многократный запуск функции через указанный интервал времени.

Все это реализуется с помощью таймеров. Таймер бывает:

- одноразовый — выполняет запуск функции лишь один раз;
- многоразовый — выполняет запуск функции через определенный интервал времени.

Таймеры работают в фоновом режиме, то есть после установки таймера работа веб-страницы и всех JavaScript сценариев на ней не приостанавливается (как и кода, который указан после метода `setTimeout`).

Одноразовый таймер. Чтобы создать одноразовый таймер, достаточно воспользоваться методом:

```
window.setTimeout(имя функции, интервал);
```

Имя функции указывается в виде строки. Собственно говоря, вместо имени функции можно указать сам код, который должен быть выполнен. Но для объемных наборов команд это не совсем удобно.

Интервал задается в миллисекундах (1000 миллисекунд равно 1 секунде). Он определяет, через какой промежуток времени после вызова метода `setTimeout` будет выполнена указанная в методе функция (или набор команд).

Например:

```
setTimeout("alert('привет, человек')",3000);
```

Назначенный, но еще не выполненный таймер можно отменить с помощью метода `clearTimeout`. Однако для этого данному методу нужно передать идентификатор таймера, который следует отменить. Получить же идентификатор таймера можно при объявлении метода `setTimeout` — он возвращается данным методом. Это делается следующим образом:

```
var mytimer = window.setTimeout(имя функции, интервал);

window.clearTimeout(mytimer);
```

Многоразовый таймер. Для создания многоразовых таймеров используется метод `setInterval`, принимающий те же аргументы, что и метод `setTimeout`:

```
window.setInterval(имя функции, интервал);
```

Возвращает метод `setInterval` идентификатор созданного таймера.

Многоразовый таймер непрерывно запускает функцию с указанным интервалом времени до тех пор, пока посетитель не закроет страницу. Либо пока многоразовый таймер не будет отменен с помощью метода `clearInterval(идентификатор таймера)`.

Методу `clearInterval` необходимо передать идентификатор таймера, который возвращается методом `setInterval`.

В качестве примера установим таймер, который бы запускал функцию `myFunction` каждые пять секунд:

```
var myTimer = setInterval("myFunction()",5000);
```

В дальнейшем, если нам понадобится прекратить запуск функции `myFunction`, мы всегда сможем воспользоваться командой `window.clearInterval(myTimer)`.

Стандартные диалоговые окна

Для общения с посетителем в JavaScript используются различные диалоговые окна (рис. 3.10).

Существует три метода для вызова диалоговых окон:

- `alert(сообщение)` — отображает информационное диалоговое окно с указанным сообщением и одной кнопкой для закрытия окна;

- `confirm(сообщение)` — выводит подтверждающее диалоговое окно с указанным сообщением и двумя кнопками для закрытия окна;
- `prompt(сообщение, по умолчанию, заголовок)` — отображает диалоговое окно с указанным сообщением, двумя кнопками для закрытия окна, а также полем для ввода данных.

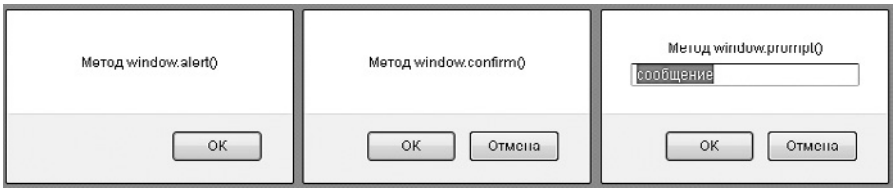


Рис. 3.10. Стандартные диалоговые окна JavaScript

Все эти методы принадлежат объекту `window`. Поэтому вызывать их можно либо с указанием объекта (`window.alert`), либо без оного (`alert`). Рассмотрим данные методы подробнее.

alert. Метод `window.alert` предназначен лишь для информирования пользователя. Диалоговое окно, которое открывается с его помощью, содержит только одну кнопку, которая предназначена для того, чтобы закрывать диалоговое окно.

confirm. Более сложный метод `window.confirm` не только информирует посетителя о каком-либо событии, но и предоставляет ему выбор. Посетитель может нажать либо кнопку `OK`, либо кнопку `Отмена`. Обе эти кнопки закрывают диалоговое окно. Но при этом от нажатой кнопки зависит, какое значение вернет метод `confirm`. Если нажата кнопка `OK`, вернется значение `true`. Если же была нажата кнопка `Отмена` — значение `false`:

```
var res = window.confirm("Метод window.confirm()");

if(res==true){

    window.alert('Нажата кнопка OK');

}else{

    window.alert('Нажата кнопка Отмена');

}
```

prompt. Еще больше возможностей предоставляет метод `window.prompt`. Он не только выводит указанное сообщение, но и позволяет пользователю ответить на ваше сообщение.

Метод `prompt` возвращает ответ пользователя. Причем возвращаемый ответ зависит от кнопки, которую посетитель нажал для закрытия диалогового окна:

- кнопка ОК — возвращает текст, который пользователь написал в диалоговом окне;
- кнопка Отмена — возвращает `null`, даже если в диалоговом окне написан какой-либо текст.

В отличие от перечисленных ранее методов, метод `prompt` требует указания трех аргументов:

- первый — содержит сообщение, которое будет выводиться в диалоговом окне;
- второй — включает в себя значение по умолчанию, которое будет показываться в поле для ввода ответа;
- третий — содержит заголовок для выводимого диалогового окна.

Например:

```
while(1==1){

   var res = window.prompt('Сколько вам лет?','16','Возрастные
ограничения');

    if(res===null){

        window.alert('Пожалуйста, ответьте на вопрос');

        continue;

    }

    if(parseInt(res)>=18){

        window.alert('Добро пожаловать на наш сайт');

    }else{

       window.alert('К сожалению, вы должны покинуть наш сайт.
Увидимся через '+(18-parseInt(res))+' лет');

    }

    break;

}
```

Работа с jQuery

Объектная модель DOM позволяет сделать с веб-страницей абсолютно все, что угодно. Но, как показали годы, работать с ней сложно. Это весьма трудоемкий процесс. Приходится писать много кода для обращения к конкретному элементу на веб-странице, а также для выполнения других задач. В том числе и по этим причинам все больше и больше веб-разработчиков отдают свое предпочтение jQuery. После чего навсегда забывают об объектной модели DOM.

Что же такое jQuery? Если модель DOM входит в состав самого языка JavaScript, то jQuery является сторонней библиотекой, которую необходимо специально подключать к HTML-документу для того, чтобы пользоваться ее возможностями. Пока это единственный недостаток jQuery — размер HTML-документа увеличивается ровно на 93 Кбайт. Именно столько весит файл последней версии библиотеки jQuery (на момент написания книги — версии 1.7.2).

Вопрос, подключать jQuery или нет, стоит перед теми, кто разрабатывает сайты с нуля. Если вы создаете сайты на различных CMS (о них мы поговорим далее в книге), вопрос подключения jQuery, скорее всего, стоять не будет. В большинстве CMS библиотека jQuery подключена к HTML-документу по умолчанию и отключить ее нельзя. А если библиотека jQuery и так подключена, то не пользоваться ее возможностями — это верх глупости.

Еще несколько слов хотелось бы сказать о версиях jQuery. С каждой новой версией в библиотеку jQuery входят все новые возможности: дополнительные функции, которые можно использовать в своих разработках. Тем не менее, если есть такая возможность, вместо новых функций лучше обходиться возможностями предыдущих версий jQuery. Хотя бы потому, что не все CMS поддерживают последние версии jQuery.

Подключение jQuery

Если вы создаете сайты на какой-либо CMS, то самостоятельно подключать jQuery, скорее всего, не нужно. В противном случае необходимо будет позаботиться о подключении jQuery к HTML-документу. К счастью, это совершенно несложно.

Перед тем как подключать jQuery, эту библиотеку нужно скачать. Для этого откройте страницу `http://docs.jquery.com/Downloading_jQuery` и скачайте сжатую (minified) версию библиотеки jQuery. Собственно, скачать библиотеку у вас не получится — перейдя по нужной ссылке, вы просто откроете в браузере JS-файл библиотеки jQuery. После этого сохраните данный JS-файл в какой-либо папке разрабатываемого сайта.

После того как JS-файл библиотеки jQuery получен, его нужно подключить к HTML-документу. Это делается точно так же, как и подключение любых других JS-файлов:

```
<head>

    <title>Заголовок</title>

      <meta http-equiv="content-type" content="text/html;
charset=UTF-8" />

   <link type="text/css" rel="stylesheet" href="css/style.css"
media="screen" />



   <script type="text/javascript" src="js/jquery-1.7.2.min.js">
</script>

</head>
```

Вот, собственно говоря, и все подключение.

Первые шаги

Для получения доступа к любой возможности jQuery используется уже известная вам точечная нотация. При этом любая команда начинается со слова `jQuery` — это начальный объект, к которому нужно обратиться для доступа к возможностям jQuery.

Например:

```
window.alert(jQuery('.test').html());

if(jQuery.browser.mozilla===true){}
```

Большинство возможностей jQuery реализовано в виде функций (методов). Свойства, которые jQuery предоставляет, можно пересчитать по пальцам.

Поскольку jQuery предназначен для работы с элементами веб-страницы, перед использованием возможностей jQuery HTML-документ должен быть полностью загружен. То есть возможности jQuery можно использовать только после наступления события `onload`. Однако добавлять вызов функций непосредственно к этому событию тега `body` нет никакой необходимости. Вместо этого, как правило, используют возможности самого jQuery.

Чтобы определенный JS-код выполнился сразу после полной загрузки HTML-документа, достаточно написать его внутри следующей конструкции:

```
jQuery(document).ready(function($){

   // JS-код, который необходимо выполнить сразу после

   // полной загрузки веб-страницы

});
```

Правильное использование jQuery:

```
   <script type="text/javascript" src="js/jquery-1.7.2.min.js">
</script>

    <script type="text/javascript">

    jQuery(document).ready(function($){

        window.alert(jQuery.browser.mozilla);

    });

    </script>
```

Неправильное использование jQuery, приводящее к ошибкам:

```
   <script type="text/javascript" src="js/jquery-1.7.2.min.js">
</script>

    <script type="text/javascript">

    window.alert(jQuery.browser.mozilla);

    </script>
```

Доступ к элементам веб-страницы

В jQuery доступ к элементам веб-страницы выполняется точно так же, как и в CSS, — путем указания селектора. Селектор указывается в круглых скобках после объекта jQuery:

```
jQuery('селектор');
```

В качестве селектора могут выступать:

- тег — `jQuery('div');`
- класс — `jQuery('.pmore');`

- идентификатор — `jQuery(#header');`
- все элементы — `jQuery('*');`
- комбинации тегов, классов и идентификаторов — `jQuery(div#header')`, `jQuery(span.older')`.

Пробелы. Как и в CSS, в jQuery вы можете указывать элементы селектора через пробел, чтобы создать последовательность вложений. Например, чтобы получить доступ к тегу `div`, расположенному внутри тега с идентификатором `header`:

```
jQuery("#header div");
```

Запятые. В одном вызове jQuery можно использовать несколько селекторов, перечисленных через запятую: `jQuery("span.older, #header div");`. В этом случае вы сможете получить доступ сразу ко всем тегам, соответствующим перечисленным селекторам.

Символы >, +, ~. В селекторах jQuery также действуют символы подчинения >, +,~:

- `jQuery("#header > div")` — доступ к тегу `div`, который является прямым потомком тега с идентификатором `header`;
- `jQuery("#header + div")` — доступ к тегу `div`, который следует непосредственно за тегом с идентификатором `header`;
- `jQuery("#header ~ div")` — доступ к тегам `div`, которые следуют за тегом с идентификатором `header`.

Атрибуты. Подробнее об этих вариантах селектора читайте в главе 2, посвященной CSS. Заодно можете перечитать разделы главы 2, посвященные селекторам по атрибутам, поскольку jQuery поддерживает и такие селекторы:

- `jQuery("тег[атрибут]")` — доступ к тегам `тег`, у которых имеется атрибут `атрибут`;
- `jQuery("тег[атрибут=значение]")` — доступ к тегам `тег`, у которых имеется атрибут `атрибут`, для которого установлено значение `значение`;
- `jQuery("тег[атрибут!=значение]")` — доступ к тегам `тег`, у которых атрибут `атрибут` не имеет значения `значение`;
- `jQuery("тег[атрибут^=значение]")` — доступ к тегам `тег`, у которых атрибут `атрибут` имеет значение, начинающееся со `значение`;
- `jQuery("тег[атрибут$=значение]")` — доступ к тегам `тег`, у которых атрибут `атрибут` имеет значение, заканчивающееся на `значение`;
- `jQuery("тег[атрибут*=значение]")` — доступ к тегам `тег`, у которых атрибут `атрибут` имеет значение, содержащее в себе подстроку `значение`;

- `jQuery("тег[атрибут~=значение]")` — доступ к тегам `тег`, у которых атрибут `атрибут` имеет значение, содержащее в себе слово `значение`;
- `jQuery("тег[атрибут|=значение]")` — доступ к тегам `тег`, у которых атрибут `атрибут` имеет значение, содержащее префикс `значение`.

В одном селекторе можно объединять несколько правил по атрибутам. Например, `jQuery("div[class=myclass][id=myid]")`.

Псевдоклассы. В состав селекторов jQuery могут также входить псевдоклассы. Однако в jQuery они называются фильтрами. И их список несколько отличается по сравнению с CSS:

- `:focus` — элемент в фокусе;
- `:first` — первый найденный элемент;
- `:last` — последний найденный элемент;
- `:eq(номер)` — элемент под заданным номером среди возвращенных селектором элементов;
- `:not(селектор)` — все найденные элементы, кроме перечисленных в скобках;
- `:even` — только элементы с четными номерами позиций среди возвращенных селектором элементов;
- `:odd` — только элементы с нечетными номерами позиций среди возвращенных селектором элементов;
- `:gt(n)` — элементы с индексом, превышающим `n` среди возвращенных селектором элементов;
- `:lt(n)` — элементы с индексом меньше `n` среди возвращенных селектором элементов;
- `:header` — только элементы, являющиеся заголовками (с тегами `h1`, `h2` и т. д.);
- `:animated` — элементы, которые в данный момент задействованы в анимации;
- `:hidden` — только невидимые элементы;
- `:visible` — только видимые элементы;
- `:contains(текст)` — элементы, содержащие заданный текст;
- `:empty` — пустые элементы, не содержащие в себе ни текста, ни других элементов;
- `:has(селектор)` — элементы, которые содержат хотя бы один элемент из перечисленных в скобках;
- `:parent` — непустые элементы;

- `:first-child` — элементы, расположенные первыми в своих родительских элементах;
- `:last-child` — элементы, размещенные последними в своих родительских элементах;
- `:nth-child( )` — элементы, расположенные указанным образом в родительских элементах (четные, нечетные, идущие под заданным номером);
- `:only-child` — элементы, которые являются единственными потомками в своих родительских элементах;
- `:button` — элементы с тегом или типом `button`;
- `:radio` — элементы, которые являются переключателями;
- `:checkbox` — элементы, которые являются флажками;
- `:text` — элементы, которые являются текстовыми полями;
- `:password` — элементы, которые являются полями ввода пароля;
- `:file` — элементы, которые являются полями загрузки файлов;
- `:submit` — элементы, которые являются кнопками отправки формы;
- `:reset` — элементы, которые являются кнопками очистки формы;
- `:image` — элементы `input` типа `image`;
- `:input` — элементы форм (теги `input`, `textarea`, `button`);
- `:selected` — элементы со статусом `selected` (например, выбранными могут быть теги `option`);
- `:checked` — элементы со статусом `checked` (например, выбранными могут быть теги `checkbox`, `radio`);
- `:enabled` — элементы со статусом `enabled`;
- `:disabled` — элементы со статусом `disabled`.

Например, `jQuery("#header div:focus");`.

Поиск дочерних элементов. Помимо указания дочерних элементов в селекторе через пробел, jQuery поддерживает функцию `find`, возвращающую указанные элементы, которые расположены внутри текущего элемента.

Например, команду `jQuery("#header div")` с помощью функции `find` можно переписать следующим образом: `jQuery("#header").find("div")`.

Функция `find` иногда бывает полезной при разработке сценариев.

Поиск предка, относительно которого выполняется позиционирование. Из главы 2, посвященной CSS, вы знаете, что значение `absolute` свойства `position` позиционирует элемент относительно первого из родительских тегов, для которых

свойству `position` было присвоено значение `relative` или `absolute`. В jQuery есть специальная функция, которая возвращает такой родительский элемент. Это функция `offsetParent()`:

```
jQuery("#colorbox").offsetParent();
```

Работа с набором элементов

Поскольку jQuery работает с селекторами, ситуация, когда команда доступа к элементу возвращает сразу несколько найденных элементов, возникает довольно часто. Если вам нужен только один из возвращенных элементов, то можно ни о чем не задумываться, так как по умолчанию jQuery возвращает только самый первый найденный элемент, а остальные отбрасывает.

Но что делать, если вам нужен не самый первый из найденных элементов, а второй, третий... последний? В этом случае проще всего воспользоваться фокусом — `:eq(индекс)`. Например:

- `jQuery("#header div:eq(1)")` — возвращает второй тег `div`, найденный внутри тега с идентификатором `header` (отсчет начинается от нуля);
- `jQuery("#header div:eq(2)")` — возвращает третий тег `div`, найденный внутри тега с идентификатором `header`, и т. д.

Таким образом можно получить доступ к одному из возвращенных значений. Но как получить доступ сразу ко всем возвращенным значениям? В jQuery это делается с помощью функции `each`:

```
jQuery(селектор).each(

   function(){

      // здесь доступ к элементу выполняется

      // с помощью команды jQuery(this)

   }

);
```

Лучше просто запомнить формат использования функции `each`, не задумываясь о том, почему он именно такой. Только обратите внимание, что внутри объявляемой функции доступ к текущему элементу выполняется командой `jQuery(this)`. Например:

```
jQuery('.test, .test2').each(

   function(){
```

```
        window.alert(jQuery(this).html());
    }
);
```

Существование элемента

Перед тем как работать с элементом, крайне желательно всегда проверять, существует ли данный элемент на веб-странице. Для этого используется свойство `length`:

```
if(jQuery('селектор').length){
    // выполняем какие-либо действия с данным селектором
}
```

Работа со свойствами CSS

Для изменения и задания значений свойствам CSS в jQuery используется метод `css("селектор", "значение")`:

```
jQuery('.test, .test2').css("display", "none");
```

Обратите внимание, что показанной в примере командой мы изменяем значение свойства CSS только у первого найденного тега. Чтобы применить изменения ко всем найденным тегам, нужно воспользоваться следующим синтаксисом:

```
    jQuery('.test, .test2').each(
        function(){
            jQuery(this).css('display','none');
        }
    );
```

Если не указывать в функции `css` нового значения, то функция вернет текущее значение данного свойства CSS:

```
if(jQuery('.test, .test2').css("display")=='none'){}
```

Не стоит также забывать и о проверке на существование элемента:

```
if(jQuery('.test, .test2').length){
    if(jQuery('.test, .test2').css("display")=='none'){
```

```
        // выполняем какие-либо действия
    }
}
```

В одной функции `css` можно задавать сразу множество CSS-свойств. В этом случае функция `css` имеет другой синтаксис: `css("свойство":"значение", "свойство":"значение")`. Например:

```
jQuery('.test, .test2').each(
    function(){
        jQuery(this).css('display':'none','padding':'7px');
    }
);
```

Если вы перебираете элементы функцией `each`, то проверять на существование элемента не нужно, так как, если подходящих элементов на веб-странице не найдено, содержимое функции `each` просто не выполнится ни разу.

Высота элемента. Отдельные функции jQuery позволяют узнать либо изменить текущую высоту элемента. Это следующие функции:

- `height()` — возвращает высоту элемента (число в пикселах, возвращается без единицы измерения);
- `height(значение)` — изменяет высоту элемента (`значение` должно быть числом в пикселах, без указания единицы измерения);
- `innerHeight()` — возвращает сумму высоты элемента и значений `padding-top` и `padding-bottom` (высота плюс внутренние отступы);
- `outerHeight()` — возвращает сумму высоты элемента и значений свойств `padding-top`, `padding-bottom`, `border-top`, `border-bottom` (высота плюс внутренние отступы и граница);
- `outerHeight(true)` — возвращает сумму высоты элемента и значений свойств `padding-top`, `padding-bottom`, `border-top`, `border-bottom`, `margin-top`, `margin-bottom` (высота плюс внутренний и внешний отступы, а также граница).

Например, `jQuery('#header').height(444);`.

Ширина элемента. Существуют также специальные функции jQuery для доступа к ширине элемента:

- `width()` — возвращает ширину элемента в виде числа в пикселах, без единицы измерения;
- `width(значение)` — изменяет ширину элемента (`значение` должно быть числом в пикселах, без указания единицы измерения);
- `innerWidth()` — возвращает сумму CSS-свойств `width`, `padding-left`, `padding-right`;
- `outerWidth()` — возвращает сумму CSS-свойств `width`, `padding-left`, `padding-right`, `border-left`, `border-right`;
- `outerWidth(true)` — возвращает сумму CSS-свойств `width`, `padding-left`, `padding-right`, `border-left`, `border-right`, `margin-left`, `margin-right`.

Например:

```
if(jQuery('#header').length && jQuery('#header').width()<77){
   jQuery('#header').width(444);
}
```

Позиция элемента. Специальные функции jQuery позволяют получить или изменить текущие координаты расположения элемента относительно родительского тега или самой веб-страницы:

- `position()` — возвращает координаты `left` и `top` расположения тега относительно родительского тега:

  ```
  jQuery('#header').position().left;
  ```

- `offset()` — возвращает координаты `left` и `top` расположения тега относительно веб-страницы:

  ```
  jQuery('#header').offset().top;
  ```

- `offset({top:значение, left:значение})` — устанавливает координаты `left` и `top` расположения тега относительно веб-страницы:

  ```
  jQuery('#header').offset({top:30, left:100});
  ```

Величина прокрутки (скроллинга). Функции `scrollTop()` и `scrollLeft()` позволяют получить или изменить величину вертикальной/горизонтальной прокрутки для элемента.

Чтобы данные функции вернули текущую величину прокрутки, их нужно вызвать без указания аргументов.

Для задания величины прокрутки достаточно указать нужную величину в качестве первого аргумента функции.

Изменение CSS-свойств при наведении указателя. Более сложной является задача изменения значений каких-либо свойств CSS при наведении указателя мыши на элемент. Для решения этой задачи селектором вида `тег:hover` не обойтись. Вместо него необходимо использовать следующую конструкцию:

```
$(селектор).hover(
   function(){
      // команды, выполняемые при наведении
      // указателя на элемент
   },
   function(){
      // команды, выполняемые, когда указатель
      // вышел за пределы элемента
   }
);
```

Например:

```
$("ul.secmenu li a").hover(
   function(){
      jQuery(this).css('color','#fff');
   },
   function(){
      jQuery(this).css('color','#000');
   }
);
```

Изменяем содержимое тега

Благодаря jQuery любое изменение содержимого веб-страницы не представляет никакой сложности. Вы можете не только добавлять содержимое в определенный тег, но и выполнять следующее:

- добавлять содержимое перед тегом;
- добавлять содержимое после тега;
- оборачивать любой тег в другие теги.

Рассматривать функции изменения содержимого мы будем на следующем HTML-примере:

```
<div id="header">Пример</div>
```

html. Добавить любой HTML-текст внутрь определенного тега можно с помощью функции `html`. Аргументом данной функции должен быть текст, который вы хотите добавить в текущий элемент:

```
jQuery("#header").html("<strong>Ваш текст</strong>");
```

Результат выполнения данного кода:

```
<div id="header"><strong>Ваш текст</strong></div>
```

Если аргумент не указывать, то функция вернет текущее содержимое выбранного тега:

```
if(!jQuery("#header").html()){

   // какие-либо действия

}
```

text. Функция `text` является аналогом функции `html`. Однако она не поддерживает теги HTML — весь добавляемый ею текст будет считаться обычным текстом. В остальном работа с функцией `text` аналогична работе с функцией `html`.

append. В jQuery отдельная функция существует даже для того, чтобы добавить содержимое в конец текущего содержимого тега (то есть эквивалент команды `jQuery("#header").html(jQuery("#header").html()+"содержимое")`). Это функция `append`:

```
jQuery("#header").append("<div>обертка</div>");
```

Результат выполнения данного кода:

```
<div id="header">Пример<div>обертка</div></div>
```

prepend. Если вы хотите добавить содержимое перед текущим содержимым тега, достаточно воспользоваться функцией `prepend`:

```
jQuery("#header").prepend("<div>обертка</div>");
```

Результат выполнения данного кода:

```
<div id="header"><div>обертка</div>Пример</div>
```

after. В некоторых случаях весьма полезной является функция, позволяющая добавлять любой текст (в том числе и HTML) после определенного тега. Это функция `after` с аргументом в виде текста, который нужно вставить:

```
jQuery("#header").after("<strong>Ваш текст</strong>");
```

Результат выполнения данного кода:

```
<div id="header">Пример</div><strong>Ваш текст</strong>
```

before. «Братом» функции `after` является функция `before`. Она позволяет вставить текст перед тегом:

```
jQuery("#header").before("<strong>Ваш текст</strong>");
```

Результат выполнения данного кода:

```
<strong>Ваш текст</strong><div id="header">Пример</div>
```

wrap. Еще одной полезной функцией является функция `wrap`. Она позволяет обернуть тег другим тегом или набором тегов. Например:

```
jQuery("#header").wrap("<strong>");
```

Результат выполнения данного кода:

```
<strong><div id="header">Пример</div></strong>
```

А вот пример более сложной обертки:

```
jQuery("#header").wrap("<div>обертка</div>");
```

Результат выполнения данного кода:

```
<div>обертка<div id="header">Пример</div></div>
```

wrapInner. Разновидностью функции `wrap` является функция `wrapInner`. С ее помощью можно обернуть содержимое тега, а не сам тег.

Сравните:

```
jQuery("#header").wrapInner("<strong>");
```

Результат выполнения данного кода:

```
<div id="header"><strong>Пример</strong></div>
```

Более сложный пример:

```
jQuery("#header").wrapInner("<div>обертка</div>");
```

Результат выполнения данного кода:

```
<div id="header"><div>обертkаПример</div></div>
```

empty. И наконец, если вам нужно удалить все содержимое тега, достаточно воспользоваться функцией `empty`. Она не принимает никаких аргументов:

```
jQuery("#header").empty();
```

Изменяем атрибуты тега

Для работы с атрибутами тега в jQuery применяется функция `attr`. Действие данной функции зависит от количества указанных в ней аргументов.

Первым аргументом функции является название атрибута. Если указан только первый аргумент, то функция вернет текущее значение данного атрибута:

```
jQuery("p").each(
   function(){
       if(jQuery(this).find("img").attr("alt")=="пример картинки"){
          // какой-либо код
       }
   }
);
```

Вторым аргументом функции, если вы решите его указать, должно быть значение, которое будет присвоено выбранному атрибуту:

```
jQuery("img").each(
   function(){
      jQuery(this).attr("alt","пример картинки");
   }
);
```

Удаление тегов

Для удаления тегов достаточно воспользоваться функцией `remove("удаляемые теги")`. При вызове без аргументов функция удаляет выбранный в данный момент тег и все его содержимое:

```
jQuery("#header").remove(); // удалить тег с id равным header
```

Аргументом данной функции может быть селектор (точно такой же селектор, как и тот, что мы указываем в команде `jQuery("селектор")`), фильтрующий удаляемые теги по дополнительным параметрам.

Управление тегом select

Отдельно стоит рассмотреть возможности, которые предоставляет jQuery по работе с тегом `select`. Это весьма интересно не только с точки зрения веб-программирования. На примере тега `select` можно увидеть всю мощь доступа к элементам через селекторы.

Итак, у нас есть тег `select`:

```
<select id="mselect" name="mselect">
   <option value="1">one</option>
   <option value="2">two</option>
   <option value="3">three</option>
</select>
```

Что же мы можем с ним сделать?

Получить значение выбранного элемента:

```
jQuery("#mselect option:selected").val();
```

Или

```
jQuery("#mselect :selected").val();
```

Или

```
jQuery("select#mselect").val();
```

Получить текст выбранного элемента:

```
jQuery("#mselect :selected").html();
```

Или:

```
jQuery("#mselect :selected").text();
```

Сделать тег select недоступным:

```
jQuery("#mselect").attr("disabled","disabled");
```

Разблокировать тег select:

```
jQuery("#mselect").attr("disabled","");
```

Удалить выбранный элемент:

```
jQuery("#mselect :selected").remove();
```

Удалить первый элемент:

```
jQuery("#mselect :first").remove();
```

Удалить последний элемент:

```
jQuery("#mselect :last").remove();
```

Удалить элемент, у которого атрибут value имеет значение 2:

```
jQuery("#mselect option[value='2']").remove();
```

Или:

```
jQuery("#mselect [value='2']").remove();
```

Очистить содержимое тега select:

```
jQuery("#mselect").empty();
```

Перебрать все теги option тега select:

```
jQuery('#mselect option').each(function(){
   // команда
});
```

Сделать выбранным последний элемент тега select:

```
jQuery("#mselect :last").attr("selected", "selected");
```

Сделать выбранным второй элемент тега select:

```
jQuery("#mselect :nth-child(2)").attr("selected", "selected");
```

Сделать выбранным элемент тега select, содержащий текст 'two':

```
jQuery("#mselect :contains('two')").attr("selected",
"selected");
```

Или:

```
jQuery("#mselect").find("option:contains('two')").
attr("selected", "selected");
```

Только первое вхождение:

```
jQuery("#mselect :contains('two')").first().attr("selected",
"selected");
```

Или:

```
jQuery("#mselect").find("option:contains('two')").first().
attr("selected", "selected");
```

Сделать выбранным элемент тега select, у которого атрибут value имеет значение 2:

```
jQuery("#mselect [value='2']").attr("selected", "selected");
```

Добавить тег option в начало списка тега select:

```
jQuery("#mselect").prepend(jQuery('<option value="0">zero</
option>'));
```

Добавить тег option в конец списка тега select:

```
jQuery("#mselect").append(jQuery('<option value="4">four</
option>'));
```

Добавить тег option после второго элемента тега select:

```
jQuery("#mselect option:nth-child(2)").after(jQuery('<option
value="22">twotwo</option>'));
```

Определить количество элементов option в списке тега select:

```
jQuery("select[id=mselect] option").size();
```

Если в списке тега select есть выбранный элемент, сделать следующее:

```
if(jQuery("#mselect").val()){}
```

Сделать все элементы в списке select невыбранными:

```
jQuery('#mselect option:selected').each(function(){
   this.selected=false;
});
```

Анимация

Уникальной возможностью jQuery является возможность создания различного рода анимаций. Вы легко можете создавать плавное скрытие и отображение элементов либо другие анимации, связанные с последовательным изменением значений каких-либо CSS-свойств.

show и hide. Специальные функции позволяют отобразить (`show`) или скрыть (`hide`) элемент путем изменения значения CSS-свойства `display`. В вызове данных функций нет ничего необычного:

```
jQuery("селектор").hide();
```

Как правило, функции анимации назначаются на какое-либо событие:

```
<script type="text/javascript">
    function anim(){
        jQuery("#header").hide();
    }
</script>
<a href="javascript:void(0);" onclick="anim();">нажми</a>
<div id="header"><strong>Пример</strong></div>
```

Пока мы вызывали функции `show` и `hide` без каких-либо аргументов. Такой синтаксис позволяет немедленно показать или немедленно скрыть элемент. Если же вам нужна анимация, достаточно вызвать одну из этих функций, указав в качестве ее аргумента продолжительность отображения/скрытия в миллисекундах:

```
jQuery("#header").hide(777);
```

При таком вызове функции произойдет плавное изменение высоты элемента:

- для функции `hide` — от текущего размера элемента до нулевой высоты;
- для функции `show` — от нулевой высоты до истинного размера элемента.

toggle. У рассмотренных выше функций есть один кардинальный недостаток — их две. А значит, при их использовании нужно дополнительно реализовывать функционал, который бы определял, свернут элемент в данный момент или нет. И уже исходя из текущего состояния элемента вызывать ту или иную функцию.

Функция `toggle` лишена указанного недостатка. Как можно легко заметить, она одна. И, несмотря на это, с ее помощью можно и скрыть, и отобразить элемент. Если элемент скрыт, функция его отображает. В противном случае элемент скрывается.

Синтаксис данной функции подобен синтаксису функций `show` и `hide`.

Если функцию `toggle` вызвать без аргументов, то элемент будет немедленно скрыт/отображен: `jQuery("селектор").toggle();`.

Если в качестве аргумента функции `toggle` указать время в миллисекундах — `jQuery("селектор").toggle();` — то элемент будет скрыт/отображен плавно, за указанное количество времени. При этом будет выполнено сразу две анимации: изменение прозрачности элемента и изменение его высоты.

animate. И последней функцией для анимации, которую мы рассмотрим, будет функция `animate`. С ее помощью можно создавать пользовательскую анимацию, то есть анимировать плавное изменение любых свойств CSS. Точнее, не любых, а лишь тех, которые связаны с размерами элемента: размер рамки и самого элемента, отступы, размер шрифта и т. д. С остальными свойствами данная функция работать не умеет.

ПРИМЕЧАНИЕ

Существует сценарий jquery.color.js, позволяющий расширить возможности функции animate. После его подключения с помощью функции animate также можно будет анимировать изменение цвета элемента.

Наиболее простой синтаксис данной функции:

```
jQuery("селектор").animate( {

    свойство1:"значение1",

    свойство2:"значение2",

} );
```

Название CSS-свойства указывается по правилам JavaScript (Camel Case), то есть дефисы из свойства удаляются, а слова, которые идут после дефисов, пишутся с большой буквы: `backgroundImage` вместо `background-image`, `borderTop` вместо `border-top` и т. д.

Значением свойства является конечное значение данного CSS-свойства, которое будет присвоено элементу в конце анимации. Записывать значение свойства можно разными способами:

- абсолютное значение — указывается привычным для нас способом и определяет конечное значение для данного CSS-свойства (`"45px"`, `"40%"` и т. д.);
- относительное значение — задается в формате `"+=значение"` или `"-=значение"`, и позволяет указать, насколько должно измениться текущее значение CSS-свойства после окончания анимации (`"+=45px"`, `"-=40%"` и т. д.).

Например:

```
jQuery("#header").animate( {

    height:"-=1px",

} );
```

Относительные значения при повторном вызове функции аккумулируются, то есть данная анимация при первом вызове функции уменьшит высоту элемента на 1 пиксел. При повторном вызове высота последовательно уменьшится на 2 пиксела. При третьем вызове — на 3 пиксела и т. д.

Работа с событиями

Еще одной полезной возможностью jQuery является возможность назначать функции определенным событиям. Ранее в книге мы делали это только с помощью атрибутов тегов, начинающихся с префикса `on` (`onclick`, `onchange` и т. д.). Сейчас давайте попробуем сделать это программно.

Для назначения функций событиям проще всего воспользоваться синтаксисом функции `bind`:

```
jQuery('селектор').bind('событие', function(){

    // выполняемые при наступлении события команды

});
```

Для указания на событие используются следующие ключевые слова: `blur`, `focus`, `load`, `resize`, `scroll`, `unload`, `beforeunload`, `click`, `dblclick`, `mousedown`, `mouseup`, `mousemove`, `mouseover`, `mouseout`, `mouseenter`, `mouseleave`, `change`, `select`, `submit`, `keydown`, `keypress`, `keyup`, `error`. Как видите, в большинстве своем это уже известные нам атрибуты тегов, у которых опущен префикс `on`.

Например, перепишем код, который мы использовали ранее для назначения анимации:

```
...
<script type="text/javascript">
    jQuery(document).ready(function($){
        jQuery('#clickme').bind('click', function(){
            jQuery("#header").toggle(777);
        });
    });
</script>
</head>
<body>
    <a id="clickme" href="javascript:void(0);">нажми</a>
    <div id="header"><strong>Пример</strong></div>
</body>
```

Используем Ajax

Технология Ajax позволяет изменять содержимое страницы без ее полной перезагрузки, то есть с помощью Ajax можно получить какие-либо данные от сервера и отобразить их на веб-странице. При этом сама веб-страница перезагружена не будет. Изменится только отдельная часть страницы, в которую вы поместите полученные с помощью Ajax данные.

Благодаря Ajax можно снизить нагрузку на сервер, уменьшить размер сайта (объем данных, которые нужно получить посетителю, чтобы просмотреть новую страницу), а также ускорить открытие страниц сайта.

Чаще всего Ajax используется для получения и отображения небольшого объема данных либо по запросу посетителя, либо автоматически, через равный интервал времени. Без Ajax для этих целей пришлось бы занового открывать всю веб-страницу.

Рассмотрим некоторые функции jQuery, которые предназначены для выполнения Ajax-запросов.

ПРИМЕЧАНИЕ

Базовые возможности JavaScript также позволяют выполнять Ajax-запросы. Но благодаря jQuery выполнение Ajax-запросов существенно упрощается.

Одной из функций для работы с Ajax, которые поддерживает jQuery, является функция `load`. С ее помощью можно поместить данные, полученные после вызова указанного в первом аргументе функции сценария, в текущий элемент:

```
$("селектор").load("файл");
```

Например:

```
$("#curmarks").load('http://dev.biz.ua/updatebookmarks.php');
```

Если бы в файле `updatebookmarks.php` содержалась строка «Привет, мир», то после выполнения данной команды строка «Привет, мир» отобразилась бы в теге с идентификатором `curmarks`.

В файл сценария, вызываемый функцией `load`, можно передать любые данные. При выполнении сценария переданными данными можно будет воспользоваться с помощью PHP.

Проще всего передавать данные `GET`-запросом:

```
$("селектор").load("файл?переменная1=значение1&переменная2=
значение2&переменная3=значение3");
```

Поскольку в строке URL запрещено использовать некоторые символы, передаваемые `GET`-запросом значения нужно предварительно кодировать, чтобы заменить запрещенные символы их разрешенными аналогами. Это можно сделать с помощью функции `encodeURIComponent(значение)`. Например:

```
jQuery("#mycart").load(Drupal.settings.basePath + 'mycart.php
?op=add&nid='+encodeURIComponent(num)+'&oid='+encodeURICompon
ent(oid));
```

Справочник jQuery

Мы рассмотрели лишь наиболее востребованные возможности jQuery. Для дальнейшего самостоятельного изучения вы можете воспользоваться табл. 3.27–3.40, в которых перечислены все функции данной библиотеки. Помимо названия функции и ее описания, в таблицах приводятся следующие сведения:

- колонка «Версия» — версия jQuery, начиная с которой данная возможность стала поддерживаться;
- колонка «Возвращает» — тип объекта, который возвращается данной функцией.

Таблица 3.27. Свойства jQuery

Версия	Свойство	Описание
1.0	$.browser	Содержит версию и тип используемого браузера. Для получения точной информации вместо данного свойства рекомендуется использовать $.support
1.3	$.support	Объект с информацией об особенностях текущего браузера
1.3	jQuery.fx.off	Отменяет выполнение всех анимаций
1.4.3	$.fx.interval	Содержит временной промежуток (в миллисекундах) между кадрами любой анимации. По умолчанию равно 13
1.3	.context	Содержит текущий контекст
1.0	.length	Содержит количество выбранных элементов
1.3	.selector	Содержит селектор, заданный в функции $()

Таблица 3.28. Базовые функции jQuery

Версия	Функция	Возвращает	Описание
1.0, 1.4	$(condition)	jQuery	Возвращает группу элементов, найденных на странице по заданным условиям
1.0, 1.4	$(htmlText)	jQuery	Возвращает элементы, заданные HTML-текстом
1.0	$(callback)	jQuery	Вызывает функцию callback, когда веб-страница полностью загружена. Аналог $(document).ready()
1.6	jQuery.holdReady(hold)	boolean	Позволяет отложить выполнение события ready
1.0	jQuery.noConflict([removeAll])	object	Освобождает переменную $, чтобы избежать конфликтов имен
1.5	jQuery.sub()	Jquery	Создает копию глобального объекта jQuery
1.5	jQuery.when(obj1,obj2,..)	Deferred	На основе нескольких заданных объектов (обычно deferred) создает новый deferred-объект, следящий за состоянием всех заданных

Таблица 3.29. Функции для работы с атрибутами

Версия	Функция	Возвращает	Описание
1.0	.attr(attrName)	string	Возвращает значение атрибута attrName для выбранного элемента
1.0	.attr(attrName, value)	jQuery	Атрибуту attrName присваивает значение value

Версия	Функция	Возвращает	Описание
1.0	.attr({attrName1: value1, attrName2: value2, ...})	jQuery	Группе атрибутов attrName1, attrName2... присваивает значения value1, value2...
1.1	.attr(attrName, function(index, value)	jQuery	Атрибуту attrName присваивает возвращенное пользовательской функцией значение
1.0	.removeAttr (attrName)	jQuery	Удаляет атрибут attrName
1.0	.addClass ("clName1 clName2 ...")	jQuery	Добавляет классы clName1, clName2
1.4	.addClass(function (index, class))	jQuery	Добавляет классы, список которых возвращает заданная пользователем функция (в виде строки, в которой классы перечислены через пробел)
1.0	.removeClass()	jQuery	Удаляет все классы
1.0	.removeClass("class1 class2 ...")	jQuery	Удаляет классы class1, class2...
1.4	.removeClass(function(index, class))	jQuery	Удаляет классы, список которых возвращает заданная пользователем функция (в виде строки, в которой классы перечислены через пробел)
1.0	.toggleClass("class1 class2 ...")	jQuery	Добавляет указанные классы, если элемент еще не содержит этих классов. Если же классы у элемента имеются, функция удаляет их
1.3	.toggleClass("class1 class2 ...", switch)	jQuery	В зависимости от параметра switch добавляет (если true) или удаляет (если false) классы
1.4	.toggleClass(function(index, class) ,[switch])	jQuery	Добавляет/удаляет классы, список которых возвращает заданная пользователем функция (в виде строки, в которой классы перечислены через пробел)
1.2	.hasClass (className)	boolean	Проверяет наличие класса className у выбранных элементов
1.0	.val()	string, array	Возвращает значение атрибута value
1.0	.val(newVal)	jQuery	Атрибуту value будет присвоено значение newVal
1.4	.val(function(index, newVal))	jQuery	Атрибуту value будет присвоено значение, возвращенное пользовательской функцией
1.6	.prop(propName)	string	Возвращает значение атрибута propName
1.6	.prop(propName, value)	jQuery	Устанавливает свойству propName значение value

Продолжение ⇩

Таблица 3.29 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.6	.prop({propName1: value1, propName2: value2, ...})	jQuery	Задает свойствам propName1, propName2... значения value1, value2... соответственно
1.6	.prop(propName, function(index, value))	jQuery	Устанавливает свойству propName возвращенное пользовательской функцией значение
1.6	.removeProp (propName)	jQuery	Удаляет свойство propName

Таблица 3.30. Функции для работы с CSS

Версия	Функция	Возвращает	Описание
1.0	css(styleName)	string	Возвращает значение CSS-свойства styleName
1.0	css(styleName, value)	jQuery	Присваивает значение value CSS-свойству styleName
1.0	css({styleName1: value1, styleName2: value2, ...})	jQuery	Группе CSS-свойств styleName1, styleName2... присваивает значения value1, value2....
1.4	css(styleName, function(index, value))	jQuery	CSS-свойству styleName присваивает значение, возвращенное пользовательской функцией
1.0	.height()	integer	Возвращает высоту элемента без учета отступов и толщины рамки
1.0	.innerHeight()	integer	Возвращает высоту элемента с учетом размера внутренних отступов (padding)
1.0	.outerHeight([includeMargin])	integer	Возвращает высоту элемента с учетом размера внутренних отступов, рамки (border-width) и при необходимости внешних отступов (margin)
1.0	.height(value)	jQuery	Устанавливает высоту элемента равной value
1.4	.height(function (index, value))	jQuery	Устанавливает высоту элемента равной значению, которое вернет пользовательская функция
1.0	.width()	integer	Возвращает ширину элемента без учета отступов и толщины рамки
1.0	.innerWidth()	integer	Возвращает ширину элемента с учетом размера внутренних отступов (padding)
1.0	.outerWidth([includeMargin])	integer	Возвращает ширину элемента с учетом внутренних отступов, рамки (border-width) и при необходимости внешних отступов (margin)
1.0	.width(value)	jQuery	Устанавливает ширину элемента равной value

Версия	Функция	Возвращает	Описание
1.4	.width(function (index, value))	jQuery	Устанавливает ширину элемента равной значению, которое вернет пользовательская функция
1.2	.offset()	integer	Возвращает координаты выбранного элемента в виде объекта с полями top и left. Координаты возвращаются относительно начала страницы
1.2	.position()	integer	Возвращает координаты выбранного элемента в виде объекта с полями top и left. Координаты возвращаются относительно ближайшего родителя, у которого CSS-свойство position равно relative, absolute или fixed
1.4	.offset(value)	jQuery	Изменяет координаты элемента на value (объект с двумя полями в виде: {top:newTop, left:newLeft})
1.4	.offset(function (index, value))	jQuery	Изменяет координаты элемента на те, которые вернет пользовательская функция
1.2.6	.offsetParent()	jQuery	Возвращает ближайшего предка элемента, у которого CSS-свойство position равно relative, absolute или fixed
1.2.6	.scrollTop()	integer	Возвращает величину вертикальной прокрутки элемента
1.2.6	.scrollLeft()	integer	Возвращает величину горизонтальной прокрутки элемента
1.2.6	.scrollTop(value)	jQuery	Изменяет величину вертикальной прокрутки на value
1.2.6	.scrollLeft(value)	jQuery	Изменяет величину горизонтальной прокрутки на value

Таблица 3.31. Функции для работы с DOM

Версия	Функция	Возвращает	Описание
1.0	.html()	string	Возвращает HTML-содержимое выбранного элемента
1.0	.html(newHTML)	jQuery	Заменяет содержимое элемента на newHTML
1.6	.html(elements)	jQuery	Перемещает элементы elements (DOM-элементы или объект jQuery) внутрь выбранных элементов
1.4	.html(function (index, value))	jQuery	Заменяет содержимое выбранных элементов на возвращенное пользовательской функцией
1.0	.text()	string	Возвращает содержащийся в элементе текст
1.0	.text(newText)	jQuery	Заменяет содержимое элемента на текст newText

Продолжение ⇩

Таблица 3.31 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.4	.text(function (index, value))	jQuery	Заменяет содержимое выбранных элементов на возвращенное пользовательской функцией
1.0	elements. append(content), content. appendTo(elements)	jQuery	Добавляет content (HTML-текст, объект jQuery или DOM-объект) в конец элементов elements
1.4	.append(function (index, value))	jQuery	Добавляет в конец элемента HTML-текст, возвращенный пользовательской функцией
1.0	elements. prepend(content), content.prependTo (elements)	jQuery	Добавляет content (HTML-текст, объект jQuery или DOM-объект) в начало элементов elements
1.4	.prepend(function (index, value))	jQuery	Добавляет в начало элемента HTML-текст, возвращенный пользовательской функцией
1.0	elements. after(content), content.insertAfter (elements)	jQuery	Добавляет content (HTML-текст, объект jQuery или DOM-объект) после элементов elements
1.4	.after(function (index))	jQuery	Добавляет после элемента возвращенный пользовательской функцией HTML-текст
1.0	elements. before(content), content.insertBefore (elements)	jQuery	Добавляет content (HTML-текст, объект jQuery или DOM-объект) перед элементами elements
1.4	.before(function (index))	jQuery	Добавляет перед элементами возвращенный пользовательской функцией HTML-текст
1.0	.wrap(content)	jQuery	Обертывает элементы содержимым content (HTML-текст, объект jQuery или DOM объект)
1.4	.wrap(function())	jQuery	Обертывает элементы возвращенным пользовательской функцией содержимым (в виде HTML-текста)
1.2	.wrapInner(content)	jQuery	Обертывает изнутри элементы содержимым content (HTML-текст, объект jQuery или DOM объект)
1.4	.wrapInner (function())	jQuery	Обертывает изнутри элементы возвращенным пользовательской функцией содержимым (в виде HTML-текста)
1.0	.remove([selector])	jQuery	Удаляет элементы со страницы
1.4	.detach([selector])	jQuery	Удаляет элементы со страницы
1.0	.empty()	jQuery	Удаляет содержимое элемента
1.4	.unwrap()	jQuery	Удаляет родительские элементы у выбранных элементов

Версия	Функция	Возвращает	Описание
1.2	elements.replaceWith(content), content.replaceAll(elements)	jQuery	Заменяет элементы elements на содержимое content (HTML-текст, объект jQuery или DOM объект)
1.4	.replaceWith(function)	jQuery	Заменяет элементы на возвращенное пользовательской функцией содержимое (в виде HTML-текста)
1.0	.clone([withDataAndEvents])	jQuery	Дублирует элементы
1.5	.clone([withDataAndEvents],[deepWithDataAndEvents])	jQuery	Дублирует элементы; при этом указывается глубина вложения, в пределах которой будут копироваться обработчики событий

Таблица 3.32. Функции для работы с набором элементов

Версия	Функция	Возвращает	Описание
1.0	.children([selector])	jQuery	Возвращает все дочерние элементы
1.4	.closest(selector, [context])	jQuery	Ищет ближайший подходящий элемент из следующих: выбранный элемент, его родитель, его прародитель и т. д.
1.6	.closest(jQuery object)	jQuery	Ищет ближайший подходящий элемент из следующих: выбранный элемент, его родитель, его прародитель и т. д. Отличается от предыдущего типом принимаемого значения
1.6	.closest(element)	jQuery	Ищет ближайший подходящий элемент из следующих: выбранный элемент, его родитель, его прародитель и т. д. Отличается от предыдущего типом принимаемого значения
1.0	.find(selector)	jQuery	Ищет элементы, соответствующие заданному селектору
1.6	.find(jQuery object)	jQuery	Ищет элементы внутри выбранных элементов, оставляя те, которые содержатся в заданном объекте jQuery
1.6	.find(element)	jQuery	Осуществляет поиск элемента element (DOM-элемент) внутри выбранных элементов
1.0	.next([selector])	jQuery	Возвращает элемент, лежащий непосредственно после заданного элемента
1.2	.nextAll([selector])	jQuery	Возвращает все элементы, лежащие после заданного элемента

Продолжение ⇩

Таблица 3.32 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.4	.next([selector])	jQuery	Возвращает все элементы, начиная от одних заданных элементов и заканчивая другими
1.2.6	.offsetParent()	jQuery	Возвращает ближайшего предка, у которого CSS-свойство position равно relative, absolute или fixed
1.0	.parent([selector])	jQuery	Возвращает родительский элемент
1.0	.parents([selector])	jQuery	Возвращает всех родителей выбранного элемента
1.4	.parentsUntil([selector])	jQuery	Возвращает всех родителей выбранного элемента, прекращая поиск перед элементами, соответствующими заданному селектору selector
1.0	.prev([selector])	jQuery	Возвращает элементы, которые находятся непосредственно перед каждым из выбранных элементов
1.4	.prevUntil ([selector])	jQuery	Возвращает элементы, которые находятся перед выбранными элементами, но не дальше первого элемента, соответствующего заданному селектору
1.0	.siblings([selector])	jQuery	Возвращает элементы, которые являются соседними для выбранных элементов
1.0	.eq(index)	jQuery	Возвращает элемент под заданным номером в наборе выбранных элементов
1.0	.filter(selector)	jQuery	Фильтрует набор элементов, оставляя только те, которые удовлетворяют селектору selector
1.0	.filter(function (index))	jQuery	Фильтрует набор элементов с помощью заданной функции (если функция возвращает false, то элемент удаляется)
1.4	.first()	jQuery	Возвращает первый элемент из выбранных
1.4	.has(selector)	jQuery	Фильтрует набор элементов, оставляя те, которые имеют соответствующих заданному селектору потомков
1.4	.has(DOMelmt)	jQuery	Фильтрует набор элементов, оставляя тот, который содержит заданный DOM-элемент
1.0	.is(selector)	boolean	Проверяет, удовлетворяет ли заданному селектору хотя бы один из выбранных элементов
1.6	.is(jQuery object)	boolean	Проверяет, есть ли среди выбранных элементов хотя бы один из элементов заданного объекта jQuery
1.6	.is(elem)	boolean	Проверяет, есть ли среди выбранных элементов элемент elem (DOM-элемент)

Версия	Функция	Возвращает	Описание
1.6	.is(function(index))	boolean	Вызывает функцию для каждого элемента
1.4	.last()	jQuery	Возвращает последний элемент из выбранных
1.0	.not(selector)	jQuery	Исключает из набора элементы, соответствующие селектору selector
1.0	.not(DOMelmts)	jQuery	Исключает из набора заданные объектом DOM (или массивом DOM-объектов) элементы
1.4	.not(function (index))	jQuery	Фильтрует набор элементов с помощью заданной функции
1.1.4	.slice(start,[end])	jQuery	Возвращает элементы с индексами от start до end
1.0	.each(callback (index, domElement))	jQuery	Выполняет функцию callback для каждого элемента
1.2	.map(callback(index, domElement))	jQuery	Выполняет функцию callback для каждого элемента
1.0	.add(selector)	jQuery	Добавляет элементы, найденные на странице с помощью заданного селектора selector
1.0	.add(elements)	jQuery	Добавляет элементы, найденные на странице с помощью заданного DOM-элемента
1.0	.add(html)	jQuery	Добавляет элементы, найденные на странице с помощью заданного HTML-текста
1.4	.add(selector, context)	jQuery	Добавляет элементы, найденные на странице с помощью заданного селектора selector внутри области, заданной параметром context (DOM-элемент, jQuery-объект или объект документа)
1.2	.andSelf()	jQuery	Добавляет элементы из предыдущего набора к текущему
1.2	.contents()	jQuery	Возвращает все дочерние элементы и текстовое содержимое выбранных элементов
1.0	.end()	jQuery	Возвращает предыдущий набор элементов

Таблица 3.33. Функции для работы с элементами формы

Версия	Функция	Возвращает	Описание
1.0	.focus(handler(event-Object))	jQuery	Устанавливает функцию handler в качестве обработчика события focus
1.4.3	.focus(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focus. При этом в обработчик передаются дополнительные данные

Продолжение ⇩

Таблица 3.33 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.0	.focus()	jQuery	Вызывает событие focus
1.0	.blur(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события blur
1.4.3	.blur(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события blur. При этом в обработчик передаются дополнительные данные
1.0	.blur()	jQuery	Вызывает событие blur
1.0	.focusin(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focusin
1.4.3	.focusin(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focusin. При этом в обработчик передаются дополнительные данные
1.0	.focusout(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focusout
1.4.3	.focusout (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focusout. При этом в обработчик передаются дополнительные данные
1.0	.select(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события select
1.4.3	.select(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события select. При этом в обработчик передаются дополнительные данные
1.0	.select()	jQuery	Вызывает событие select
1.0	.submit(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события submit
1.4.3	.submit(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события submit. При этом в обработчик передаются дополнительные данные
1.0	.submit()	jQuery	Вызывает событие submit
1.2	jQuery.param(obj)	string	Преобразует объект, массив или массив объектов в строку, закодированную для передачи через URL
1.4	jQuery.param (obj,[traditional])	string	Преобразует объект, массив или массив объектов в строку, закодированную для передачи через URL. При этом вы можете выбрать тип преобразования, которое будет выполнено (глубокое или поверхностное)

Версия	Функция	Возвращает	Описание
1.0	.serialize()	string	Возвращает строку (в виде параметров URL-запроса) с именами и значениями выбранных элементов формы
1.2	.serializeArray()	Array	Возвращает массив объектов, содержащий данные элементов формы
1.0	.val()	string, array	Возвращает значение атрибута value
1.0	.val(newVal)	jQuery	Устанавливает значение newVal атрибуту value
1.4	.val(function(index, newVal))	jQuery	Устанавливает атрибуту value значение, возвращенное пользовательской функцией

Таблица 3.34. Функции для работы с событиями

Версия	Функция	Возвращает	Описание
1.7	.on(events, [selector], [data], handler)	jQuery	Устанавливает обработчик событий
1.7	.on(events-map, [selector], [data])	jQuery	Устанавливает сразу несколько разных обработчиков событий
1.7	.off([events], [selector], [handler])	jQuery	Удаляет обработчики событий, установленные с помощью метода .on()
1.7	.off(events-map, [selector])	jQuery	Удаляет сразу несколько разных обработчиков событий, установленных с помощью метода .on()
1.0	.bind(eventType, [eventData], handler (eventObject))	jQuery	Устанавливает обработчик для указанного события
1.4.3	.bind(eventType, [eventData], false)	jQuery	Отменяет обработчик для события
1.4	.bind(events)	jQuery	Устанавливает обработчики сразу для нескольких событий
1.3	.live(eventType, handler)	jQuery	Устанавливает обработчик события для текущих элементов страницы и всех, которые будут добавлены на страницу в дальнейшем
1.4	.live(eventType, eventData, handler)	jQuery	Устанавливает обработчик события для текущих элементов страницы и всех, которые будут добавлены на страницу в дальнейшем; при этом указываются данные, передаваемые в функцию

Продолжение ⇩

Таблица 3.34 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.4.2	.delegate(selector, eventType, [eventData], handler)	jQuery	Устанавливает обработчик события на текущие и любые новые (добавленные на страницу в дальнейшем) элементы, соответствующие заданному селектору
1.1	.one(eventType, [eventData], handler (eventObject))	jQuery	Устанавливает обработчик события, который будет вызван только один раз
1.0	.unbind ([eventType], [handler (eventObject)])	jQuery	Удаляет обработчики событий, установленные методами bind(), one() или методами click(), focus() и т. д.
1.4	.unbind ([eventType], false)	jQuery	Удаляет обработчики событий, установленные методом .bind(eventType, [eventData], false)
1.4	.unbind (eventObject)	jQuery	Удаляет обработчик события при его непосредственном вызове
1.3	.die(eventType, [handler])	jQuery	Удаляет указанные обработчики событий, установленные методом live()
1.4.1	.die()	jQuery	Удаляет все обработчики событий, установленные методом live()
1.4.2	.undelegate(selector, eventType, [handler])	jQuery	Удаляет обработчики событий, установленные методом delegate()
1.4.2	.undelegate()	jQuery	Удаляет все обработчики событий, установленные методом delegate()
1.6	.undelegate (namespace)	jQuery	Удаляет обработчики всех событий с заданным пространством имен
1.0	.trigger(eventType, [extraParameters])	jQuery	Вызывает событие у выбранных элементов
1.3	.trigger (eventObject)	jQuery	Повторно запускает событие из обработчика текущего события
1.2	.triggerHandler (eventType, [extraParameters])	jQuery	Вызывает выполнение обработчиков заданного события у выбранных элементов. Само событие при этом не происходит
1.4	jQuery. proxy(function, context)	function	Создает такую же функцию, внутри которой переменная this будет равна заданному значению
1.4	jQuery. proxy(context, name, [arg1], [arg2], ...)	function	На основе метода name создает функцию, внутри которой переменная this будет равна заданному значению
1.0	.click(handler(event-Object))	jQuery	Устанавливает функцию handler в качестве обработчика события click

Версия	Функция	Возвращает	Описание
1.4.3	.click(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события click. При этом в обработчик передаются дополнительные данные
1.0	.click()	jQuery	Вызывает событие click
1.0	.dblclick(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события dblclick
1.4.3	.dblclick(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события dblclick. При этом в обработчик передаются дополнительные данные
1.0	.dblclick()	jQuery	Вызывает событие dblclick
1.0	.hover(handlerIn (eventObject), handlerOut (eventObject))	jQuery	Устанавливает функции handlerIn и handlerOut в качестве обработчиков событий mouseenter и mouseleave
1.4	hover(handlerInOut (eventObject))	jQuery	Устанавливает функцию handlerInOut в качестве обработчика обоих событий (mouseenter и mouseleave)
1.0	.mousedown (handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mousedown
1.4.3	.mousedown (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mousedown. При этом в обработчик передаются дополнительные данные
1.0	.mousedown()	jQuery	Вызывает событие mousemove
1.0	.mouseup(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseup
1.4.3	.mouseup (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseup. При этом в обработчик передаются дополнительные данные
1.0	.mouseup()	jQuery	Вызывает событие mouseup
1.0	.mouseenter(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseenter
1.4.3	.mouseenter (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseenter. При этом в обработчик передаются дополнительные данные
1.0	.mouseenter()	jQuery	Вызывает событие mouseenter
1.0	.mouseleave(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseleave

Продолжение ⇩

Таблица 3.34 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.4.3	.mouseleave (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseleave. При этом в обработчик передаются дополнительные данные
1.0	.mouseleave()	jQuery	Вызывает событие mouseleave
1.0	.mousemove(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mousemove
1.4.3	.mousemove (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mousemove. При этом в обработчик передаются дополнительные данные
1.0	.mousemove()	jQuery	Вызывает событие mousemove
1.0	.mouseout(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseout
1.4.3	.mouseout (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseout. При этом в обработчик передаются дополнительные данные
1.0	.mouseout()	jQuery	Вызывает событие mouseout
1.0	.mouseover(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseover
1.4.3	.mouseover (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события mouseover. При этом в обработчик передаются дополнительные данные
1.0	.mouseover()	jQuery	Вызывает событие mouseover
1.0	.toggle(handler1 (eventObject), handler2 (eventObject), [handler3 (eventObject)])	jQuery	Поочередно выполняет одну из двух или более заданных функций handler в ответ на щелчок на элементе
1.0	.toggle([duration], [callback])	jQuery	Изменяет видимость выбранных элементов на противоположную (показывает/скрывает)
1.4.3	.toggle([duration], [easing],[callback])	jQuery	Изменяет видимость выбранных элементов на противоположную (показывает/скрывает) с заданной скоростью отображения/исчезновения
1.3	.toggle (showOrHide)	jQuery	Показывает (showOrHide = true) или убирает с экрана (showOrHide = false) выбранные элементы на странице
1.0	.keydown(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события keydown

Версия	Функция	Возвращает	Описание
1.4.3	.keydown (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события keydown. При этом в обработчик передаются дополнительные данные
1.0	.keydown()	jQuery	Вызывает событие keydown
1.0	.keyup(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события keyup
1.4.3	.keyup (eventData, handler (eventObject)	jQuery	Устанавливает функцию handler в качестве обработчика события keyup. При этом в обработчик передаются дополнительные данные
1.0	.keyup()	jQuery	Вызывает событие keyup
1.0	.keypress(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события keypress
1.4.3	.keypress (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события keypress. При этом в обработчик передаются дополнительные данные
1.0	.keypress()	jQuery	Вызывает событие keypress
1.0	.focus(handler(event-Object))	jQuery	Устанавливает функцию handler в качестве обработчика события focus
1.4.3	.focus (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focus. При этом в обработчик передаются дополнительные данные
1.0	.focus()	jQuery	Вызывает событие focus
1.0	.blur(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события blur
1.4.3	.blur(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события blur. При этом в обработчик передаются дополнительные данные
1.0	.blur()	jQuery	Вызывает событие blur
1.0	.focusin(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focusin
1.4.3	.focusin (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focusin. При этом в обработчик передаются дополнительные данные
1.0	.focusout(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focusout
1.4.3	.focusout (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события focusout. При этом в обработчик передаются дополнительные данные

Продолжение ⇩

Таблица 3.34 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.0	.select(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события select
1.4.3	.select (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события select. При этом в обработчик передаются дополнительные данные
1.0	.select()	jQuery	Вызывает событие select
1.0	.submit(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события submit
1.4.3	.submit (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события submit. При этом в обработчик передаются дополнительные данные
1.0	.submit()	jQuery	Вызывает событие submit
1.0	.change(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события change
1.4.3	.change (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события change. При этом в обработчик передаются дополнительные данные
1.0	.change()	jQuery	Вызывает событие change
1.0	.ready(handler (eventObject))	jQuery	Устанавливает обработчик готовности дерева DOM
1.0	.load(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события load
1.4.3	.load (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события load. При этом в обработчик передаются дополнительные данные
1.0	.unload(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события unload
1.4.3	.unload (eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события unload. При этом в обработчик передаются дополнительные данные
1.0	.error(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события error
1.4.3	.error(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события error. При этом в обработчик передаются дополнительные данные

Версия	Функция	Возвращает	Описание
1.0	.resize(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события resize
1.4.3	.resize(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события resize. При этом в обработчик передаются дополнительные данные
1.0	.resize()	jQuery	Вызывает событие resize
1.0	.scroll(handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события scroll
1.4.3	.scroll(eventData, handler (eventObject))	jQuery	Устанавливает функцию handler в качестве обработчика события scroll. При этом в обработчик передаются дополнительные данные
1.0	.scroll()	jQuery	Вызывает событие scroll

Таблица 3.35. Функции для работы с эффектами и анимацией

Версия	Функция	Возвращает	Описание
1.0	.animate(properties, [duration], [easing], [callback])	jQuery	Выполняет заданную пользователем анимацию
1.2	.queue ([queueName])	jQuery	Возвращает очередь функций (в виде массива), относящихся к выбранным элементам
1.2	.queue ([queueName], newQueue)	jQuery	Заменяет очередь с именем queueName, на newQueue
1.2	.queue ([queueName], callback(next))	jQuery	Устанавливает функцию callback в конец очереди
1.4	.clearQueue ([queueName])	jQuery	Очищает очередь функций
1.2	.dequeue ([queueName])	jQuery	Начинает выполнение следующей функции в очереди
1.2	.stop([clearQueue], [jumpToEnd])	jQuery	Останавливает выполнение текущей анимации
1.7	.stop([queue], [clearQueue], [jumpToEnd])	jQuery	Останавливает выполнение текущей анимации в указанной очереди
1.4	.delay(duration, [queueName])	jQuery	Задерживает выполнение следующей функции в очереди на duration миллисекунд

Продолжение ⇩

Таблица 3.35 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.0	.toggle(handler1 (eventObject), handler2 (eventObject), [handler3 (eventObject)])	jQuery	Поочередно выполняет одну из двух или более заданных функций handler в ответ на щелчок на элементе
1.0	.toggle([duration], [callback])	jQuery	Изменяет видимость элементов на противоположную
1.3	.toggle (showOrHide)	jQuery	Показывает (showOrHide = true) или убирает с экрана (showOrHide = false) выбранные элементы на странице
1.4.3	$.fx.interval		Содержит временной промежуток (в миллисекундах) между кадрами анимации
1.3	jQuery.fx.off	boolean	Установка этого свойства в true отключает все анимации
1.4.4	.fadeToggle ([duration], [easing],[callback])	jQuery	За счет изменения прозрачности плавно скрывает видимый элемент или отображает скрытый

Таблица 3.36. Функции для работы с Ajax

Версия	Функция	Возвращает	Описание
1.0	jQuery. get(url,[data], [callback], [dataType])	jqXHR	Осуществляет GET-запрос к серверу без перезагрузки страницы
1.0	jQuery.post(url, [data],[callback], [dataType])	jqXHR	Осуществляет POST-запрос к серверу без перезагрузки страницы
1.0	.load(url,[data],[callback],[dataType])	jqXHR	Осуществляет запрос к серверу без перезагрузки страницы и помещает полученные от сервера данные внутрь выбранных элементов
1.0	jQuery.getJSON (url,[data],[callback])	jqXHR	Осуществляет запрос JSON-данных у сервера методом GET
1.0	jQuery. getScript(url, [success(data, textStatus)])	XMLHttp-Request	Без перезагрузки страницы запрашивает JavaScript-файл и выполняет его после получения
1.5	jQuery. ajax(url,[settings])	jqXHR	Осуществляет запрос к серверу без перезагрузки страницы
1.0	jQuery. ajaxSetup(options)		Устанавливает параметры, которые будут использоваться по умолчанию при выполнении функции jQuery.ajax()

Версия	Функция	Возвращает	Описание
1.0	.ajaxSend(handler (event, XMLHttpRequest, ajaxOptions))	jQuery	Устанавливает обработчик отправления Ajax-запросов
1.0	.ajaxComplete (handler(event, XMLHttpRequest, ajaxOptions))	jQuery	Устанавливает обработчик завершения Ajax-запроса
1.0	.ajaxSuccess (handler(event, XMLHttpRequest, ajaxOptions))	jQuery	Устанавливает обработчик удачного завершения Ajax-запроса
1.0	.ajaxError(handler (event, XMLHttpRequest, ajaxOptions, thrownError))	jQuery	Устанавливает обработчик неудачного завершения Ajax-запроса
1.0	.ajaxStart(handler)	jQuery	Устанавливает функцию handler() в качестве обработчика запуска Ajax-запроса при условии, что в этот момент не выполняются другие Ajax-запросы
1.0	.ajaxStop(handler)	jQuery	Устанавливает функцию handler() в качестве обработчика завершения всех текущих Ajax-запросов
1.2	jQuery.param(obj)	string	Преобразует объект, массив или массив объектов в строку, закодированную для передачи через URL
1.4	jQuery.param(obj, [traditional])	string	Преобразует объект, массив или массив объектов в строку, закодированную для передачи через URL. При этом вы можете выбрать тип преобразования, которое будет выполнено (глубокое или поверхностное)
1.0	.serialize()	string	Возвращает строку (в виде параметров URL-запроса) с именами и значениями выбранных элементов формы
1.2	.serializeArray()	Array	Возвращает массив объектов, содержащий данные элементов формы

Таблица 3.37. Методы объекта callbacks

Версия	Функция	Возвращает	Описание
1.7	.add(func)	–	Добавляет заданную функцию/функции в список функций
1.7	.disable()	–	Блокирует текущий объект callbacks

Продолжение ⇩

Таблица 3.37 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.7	.empty()	–	Очищает список функций в текущем объекте callbacks
1.7	.fire(arguments)	–	Выполняет все функции из списка функций callbacks-объекта
1.7	.fireWith(context, [arguments])	–	Выполняет все функции из списка функций callbacks-объекта
1.7	.fired()	boolean	Проверяет, выполнялся ли текущий объект callbacks хотя бы один раз
1.7	.has(func)	boolean	Проверяет наличие заданной функции в списке функций текущего объекта callbacks
1.7	.remove(func)	–	Удаляет заданную функцию/функции из списка функций текущего объекта callbacks

Таблица 3.38. Методы объекта deferred

Версия	Функция	Возвращает	Описание
1.5	.done(handler)	deferred	Устанавливает заданную функцию handler в качестве обработчика перехода объекта deferred в состояние удачного выполнения
1.5	.fail(handler)	deferred	Обрабатывает переход объекта deferred в состояние ошибки выполнения
1.5	.then(doneHandler,failHandler)	deferred	Устанавливает обработчики в состояние удачного выполнения и состояние ошибки выполнения
1.6	.always(handler)	deferred	Устанавливает общий обработчик, реагирующий на переход в любое из двух состояний
1.7	.then(doneHandler, failHandler,progressHandler)	deferred	Устанавливает обработчики в состояние удачного выполнения и состояние ошибки выполнения
1.7	.progress(handler)	deferred	Устанавливает handler на объект deferred в качестве обработчика события progress
1.5	.resolve([args])	deferred	Переводит объект deferred из состояния «не выполнено» в «успешно выполнено»
1.5	.reject([args])	deferred	Переводит объект deferred из состояния «не выполнено» в «ошибка выполнения»
1.7	.notify([args])	deferred	Вызывает событие промежуточного этапа выполнения объекта deferred (прогресса), что приводит к выполнению обработчиков, установленных методами progress() и then()
1.5	.resolveWith(context,[args])	deferred	Переводит объект deferred из состояния «не выполнено» в «успешно выполнено»

Версия	Функция	Возвращает	Описание
1.5	.rejectWith(context, [args])	deferred	Переводит объект deferred из состояния «не выполнено» в «ошибка выполнения»
1.5	.isResolved()	boolean	Проверяет, находится ли deferred-объект в состоянии resolved
1.5	.isRejected()	boolean	Проверяет, находится ли deferred-объект в состоянии rejected
1.7	.state([doneFilter], [failFilter])	string	Возвращает текущее состояние объекта deffered
1.6	.pipe([doneFilter], [failFilter])	promise	Создает promise-объект, состояние которого будет меняться вместе с исходным объектом deffered
1.7	.pipe([doneFilter], [failFilter],[progress-Filter])	promise	Создает promise-объект, состояние которого будет меняться вместе с исходным объектом deffered
1.5	.promise([target])	promise	Создает заместителя deferred-объекта
	$(...).promise()	–	Наблюдение за очередью событий
1.5	jQuery. when(obj1,obj2,..)	deferred	На основе нескольких заданных объектов (обычно deferred) создает новый deferred-объект, следящий за состоянием всех заданных

Таблица 3.39. Утилитарные функции

Версия	Функция	Возвращает	Описание
1.4	$.contains (container, contained)	boolean	Возвратит true, если DOM-элемент contained находится внутри элемента container
1.0	$.extend(target, [object1],..,[objectN])	jQuery	Объединяет содержимое объектов target, object1.. objectN, помещая результат в объект target
1.1.4	$.extend([deep], target,[object1],.., [objectN])	jQuery	Объединяет содержимое объектов target, object1.. objectN, помещая результат в объект target
1.0.4	$.globalEval(text)	–	Выполняет заданный сценарий в глобальной области видимости
1.0	$.grep(array, function, [invert])	array	Возвращает найденные в заданном массиве элементы, удовлетворяющие условиям фильтрующей функции
1.2	$.inArray(value, array)	number	Ищет элемент в массиве. Возвращает –1 если элемент не найден
1.3	$.isArray(obj)	boolean	Проверяет, является ли заданный элемент массивом

Продолжение ⇩

Таблица 3.39 *(продолжение)*

Версия	Функция	Возвращает	Описание
1.4	$.isEmptyObject (obj)	boolean	Проверяет, является ли заданный объект пустым
1.2	$.isFunction(obj)	boolean	Проверяет, является ли заданный элемент функцией
1.7	$.isNumeric(value)	boolean	Проверяет, является ли заданный аргумент числом или строкой в виде числа
1.4	$.isPlainObject(obj)	boolean	Проверяет, является ли заданный элемент пользовательским объектом
1.4.3	$.isWindow(obj)	boolean	Проверяет, является ли заданный элемент объектом типа window или объектом фреймов
1.1.4	$.isXMLDoc(node)	boolean	Проверяет, находится ли DOM-объект внутри XML-документа (или сам является XML-документом)
1.2	$.makeArray(obj)	array	Конвертирует массивоподобные объекты в массивы
1.0	jQuery.map(array, callback(elem, index))	array	Выполняет функцию callback для каждого элемента массива array в отдельности
1.6	jQuery. map(arrayOrObj, callback(elem, indexOrKey))	array	Выполняет функцию callback для каждого элемента массива array в отдельности
1.0	$.merge(firsrArr, secondArr)	array	Объединяет содержимое массивов
1.4	$.noop()	–	Пустая функция, которая ничего не делает (используется, например, чтобы сделать задержку перед выполнением следующего кода)
1.4.1	$.parseJSON(json)	jQuery	Конвертирует строку с JSON-данными в JavaScript-объект
1.5	$.parseXML (xmlString)	jQuery	Конвертирует строку с XML-данными в XML-документ
1.4.3	$.type(el)	string	Определяет внутренний класс JavaScript (string, boolean и т. д.) заданного элемента
1.1.3	$.unique(arr)	array	Сортирует массив с DOM-элементами, выстраивая их в порядке расположения в DOM и удаляя повторения

Таблица 3.40. Другие функции

Версия	Функция	Возвращает	Описание
1.2.3	.data(key, value)	jQuery	Устанавливает переменную key со значением value всем выбранным элементам страницы

Версия	Функция	Возвращает	Описание
1.4	.data({key1: value1, key2: value2, ...}	jQuery	Устанавливает переменные с указанными значениями всем выбранным элементам страницы
1.2.3	.data(key)	any_type	Возвращает значение переменной key у первого выбранного элемента
1.4	.data()	Object	Возвращает объект со всеми переменными, прикрепленными к первому из выбранных элементов
1.2.3	.removeData ([name])	jQuery	Удаляет пользовательские переменные, установленные методом .data()
1.7	.removeData([list])	jQuery	Удаляет пользовательские переменные, установленные методом .data()
1.5	jQuery. hasData(element)	boolean	Проверяет, закреплены ли данные за указанным элементом страницы
1.5	.get([index])	–	Возвращает DOM-элементы, хранящиеся в объекте jQuery
1.5	.toArray()	–	Возвращает массив всех DOM-элементов, хранящихся в объекте jQuery
1.5	.index()	–	Возвращает номер позиции первого выбранного элемента относительно соседних элементов на странице
1.5	.index(selector)	–	Возвращает номер позиции первого выбранного элемента среди элементов, удовлетворяющих селектору selector
1.5	.index(element)	–	Возвращает номер позиции элемента element (DOM-элемент или объект jQuery) в наборе выбранных элементов
1.5	.size()	–	Возвращает количество выбранных элементов

Готовые сценарии

Рассмотрим несколько дополнительных сценариев, которые при разработке сайтов приходится использовать наиболее часто.

Темизация (изменение) полосы прокрутки (jScrollPane.js)

В главе данной книги, посвященной CSS, уже упоминалась задача темизации (изменения) полосы прокрутки. Поскольку полностью изменить вид полосы прокрутки посредством CSS невозможно, на помощь приходит сценарий `jScrollPane.js`. А вместе с ним и jQuery, так как без него данный сценарий не будет работать.

Данный сценарий скрывает полосу прокрутки по умолчанию, которая отображается у любого переполненного содержимым тега-контейнера, и заменяет ее своей полосой прокрутки, созданной с помощью JavaScript (рис. 3.11).

Рис. 3.11. Пользовательская полоса прокрутки

Скачать сценарий `jScrollPane.js` можно на странице http://jscrollpane.kelvinluck.com/. Вместе с JS-файлом идет CSS-файл, содержащий настройки вида полосы прокрутки по умолчанию. Темизация полосы прокрутки выполняется путем модификации данного CSS-файла.

Для работы сценария `jScrollPane.js` необходим также дополнительный сценарий `jquery.mousewheel.min.js`. Его можно скачать со страницы https://github.com/brandonaaron/jquery-mousewheel/.

Подключите JS и CSS-файлы сценария к своему HTML-документу. Далее оберните содержимое, для которого вы хотите создать пользовательскую полосу прокрутки, тегами `<div class="scroll-wrap"><div class="scroll-pane">`:

```
<div><div class="scroll-wrap"><div class="scroll-pane">Ваш текст…</div></div></div>
```

Добавьте CSS-код, определяющий ширину и высоту блока с полосой прокрутки:

```
.scroll-pane{
    overflow: hidden;
    width: 333px;
```

```
    height: 111px;
}
.scroll-wrap{
    margin: 0 auto;
}
```

Затем вызовите функцию `jScrollPane` после полной загрузки веб-страницы:

```
<link type="text/css" rel="stylesheet" href="css/jquery.
jscrollpane.css" media="screen" />
<script type="text/javascript" src="js/jquery-1.7.2.min.js">
</script>
<script type="text/javascript" src="js/jquery.mousewheel.min.
js"> </script>
<script type="text/javascript" src="js/jScrollPane.js"> </
script>
<script type="text/javascript">
jQuery(document).ready(function($){
   $('.scroll-pane').jScrollPane();
});
</script>
```

В качестве аргумента функции `jScrollPane` может выступать аргумент с дополнительными настройками сценария. Например:

```
$('.scroll-pane').jScrollPane({showArrows:false,scrollbarWidt
h:9,scrollbarMargin:7,dragMaxHeight:17,dragMinHeight:17,maint
ainPosition:7,scrollbarOnLeft:false,bottomCapHeight:6,topCapH
eight:6});
```

Тень под текстом (jquery.dropshadow.js или text-shadow.min.js)

Еще одной задачей, с которой мы уже встречались в главе, посвященной CSS, является создание тени под текстом (рис. 3.12). Справиться с данной задачей можно с помощью одного из двух популярных сценариев: `jquery.dropshadow.js` или `text-shadow.min.js`. Оба этих сценария требуют для своей работы jQuery.

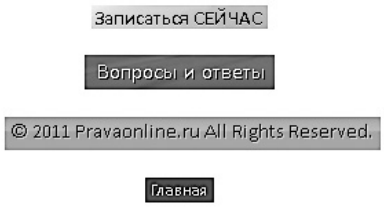


Рис. 3.12. Использование тени под текстом

Сказать, какой из данных сценариев лучше, нельзя. В некоторых случаях сценарий `jquery.dropshadow.js` ведет себя стабильнее своего оппонента. В других — качественнее выглядит сценарий `text-shadow.min.js`. Это определяется экспериментальным путем. Если при использовании одного из сценариев возникают какие-либо проблемы (как правило, тень неправильно смещается относительно текста), следует попробовать другой сценарий.

jquery.dropshadow.js. Сценарий `jquery.dropshadow.js` можно скачать на странице http://code.google.com/p/dropshadow/downloads/detail?name=jquery.dropshadow.js&can=2&q=.

Подключите сценарий и сам jQuery к вашему HTML-документу.

После этого для назначения тени элементу можно будет использовать функцию `dropShadow`. Аргументом данной функции является объект, состоящий из цвета тени, смещения тени по горизонтали и вертикали, а также размытости тени:

```
jQuery(document).ready(function($){

    jQuery("div#designed").dropShadow({color:"#ffffff",left:1,
top:1,blur:0});

});
```

text-shadow.min.js. Сценарий `text-shadow.min.js` можно скачать на странице http://recens.ru/jquery/plugin_text_shadow.html.

После этого для назначения тени элементу можно будет использовать функцию `textShadow`. Аргументом данной функции является строка, аналогичная значению CSS-свойства `text-shadow`:

```
jQuery(document).ready(function($){

   jQuery("div#designed").textShadow('1px 1px 5px #191919');

});
```

Темизация списков (ui.dropdownchecklist.js)

Стандартный раскрывающийся список HTML не очень удобен для выбора нескольких значений из списка. Он позволяет выбирать несколько значений только с нажатой клавишей Ctrl, что не совсем подходит для многих пользователей. По этой причине раскрывающиеся списки с множественным выбором предпочитают делать с помощью JavaScript и jQuery (рис. 3.13).

Рис. 3.13. Использование пользовательского раскрывающегося списка

Для создания пользовательских раскрывающихся списков используется сценарий `ui.dropdownchecklist.js`. Его можно скачать со страницы http://code.google.com/p/dropdown-check-list/source/browse/trunk/src/?r=51.

Для работы данного сценария необходима не только библиотека jQuery, но и библиотека с дополнительными возможностями jQuery UI. Ее можно скачать на странице http://jqueryui.com/download.

Подключив все сценарии и CSS-файлы к HTML-документу, создайте раскрывающийся список с поддержкой выбора нескольких значений:

```
<select multiple="multiple" class="sclass_area" name="ta[]">

<option value="0"></option>

<option value="7">ВАО</option>

<option value="3">ЗАО</option>

<option value="4">САО</option>

</select>
```

Обязательно присвойте какой-либо класс или идентификатор созданному раскрывающемуся списку. После этого добавьте вызов функции `dropdownchecklist` при полной загрузке веб-страницы:

```
jQuery(document).ready(function($){

    if(jQuery(".sclass_area").length){

        jQuery(".sclass_area").dropdownchecklist
({ maxDropHeight: 150, width: 116, explicitClose: '<i>...
закрыть</i>', textFormatFunction: function(options) {

        var selectedOptions = options.filter(":selected");

        var countOfSelected = selectedOptions.size();

        var size = options.size();

        switch(countOfSelected) {

            case 0: return "Любой";

            case 1: return selectedOptions.text();

            case size: return "Все округа";

            default: return  " Выбранных: " +
countOfSelected;

        }

        }

     });

    }

});
```

Аргументы функции `dropdownchecklist` позволяют настроить вид раскрывающегося списка:

- `maxDropHeight` — максимальную высоту блока с элементами списка, отображаемого после щелчка кнопкой мыши на списке;
- `width` — ширину раскрывающегося списка;
- `explicitClose` — отображающийся под элементами раскрывающегося списка текст, который позволяет закрыть блок с элементами списка;
- `textFormatFunction` — код JavaScript, возвращающий текст, который будет отображаться в раскрывающемся списке при выборе определенного количества элементов (на рис. 3.13 это текст «Выбранных: 2»).

Анимация цвета (jquery.color.js)

Изучая возможности jQuery по созданию анимации, мы познакомились с функцией `animate`. Она позволяет создавать эффект анимации с помощью любых CSS-свойств, значением которых является какой-либо размер. Однако в том же разделе книги было сказано, что существует дополнительный сценарий, расширяющий возможности функции `animate`. Это сценарий `jquery.color.js`.

Благодаря сценарию `jquery.color.js` у нас появляется возможность анимировать CSS-свойства, значением которых является цвет (рис. 3.14). Скачать данный сценарий можно со страницы http://archive.plugins.jquery.com/project/color.

Рис. 3.14. Анимация цвета: измение вида ссылки при наведении

Для работы данного сценария необходимы библиотеки jQuery и jQuery UI. Скачать jQuery UI можно на странице http://jqueryui.com/download.

Итак, подключите все библиотеки к HTML-документу, после чего добавьте код вызова функции `animate` с подходящей анимацией. Например:

```
jQuery(document).ready(function($){
   jQuery("#navigation-primary-main li a").mouseenter(
      function () {
         if(jQuery(this).parent().hasClass('active')){}else{
            jQuery(this).parent().animate({
               backgroundColor:"#f9f8f8",
            }, {queue:false, duration:150 } );
         }
      });
   jQuery("#navigation-primary-main li a").mouseleave(
      function() {
```

```
        if(jQuery(this).parent().hasClass('active')){}else{
            jQuery(this).parent().animate({
                backgroundColor:"#f88e06",
            }, {queue:false, duration:150 } );
      }
   });
});
```

Всплывающие окна (ColorBox)

Всплывающие окна являются очень популярной возможностью JavaScript. Под всплывающими окнами имеются в виду отображаемые поверх сайта и без перезагрузки текущей веб-страницы окна с дополнительной информацией (рис. 3.15). Содержимым всплывающего окна может быть:

- какая-либо картинка;
- блок текста, имеющийся на веб-странице и скрытый до тех пор, пока не откроется всплывающее окно;
- сторонняя веб-страница.

Одним из сценариев для создания всплывающего окна является сценарий `jquery.colorbox.js`. Архив с ColorBox можно скачать на странице http://www.jacklmoore.com/colorbox.

В качестве примера рассмотрим создание галереи всплывающих окон.

Подключите jQuery и ColorBox:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/
jquery.min.js"></script>

<script src="js/jquery.colorbox.js"></script>
```

После чего в HTML-документе создайте ссылки на картинки, присвоив им одинаковый класс:

```
<p><a class="group1" href="img/pict1.jpg" title="Заголовок 1"
>Картинка 1</a></p>

<p><a class="group1" href="img/pict2.jpg" title="Заголовок 2"
>Картинка 2</a></p>
```

```
<p><a class="group1" href="img/pict3.jpg" title="Заголовок 3"
>Картинка 3</a></p>
```

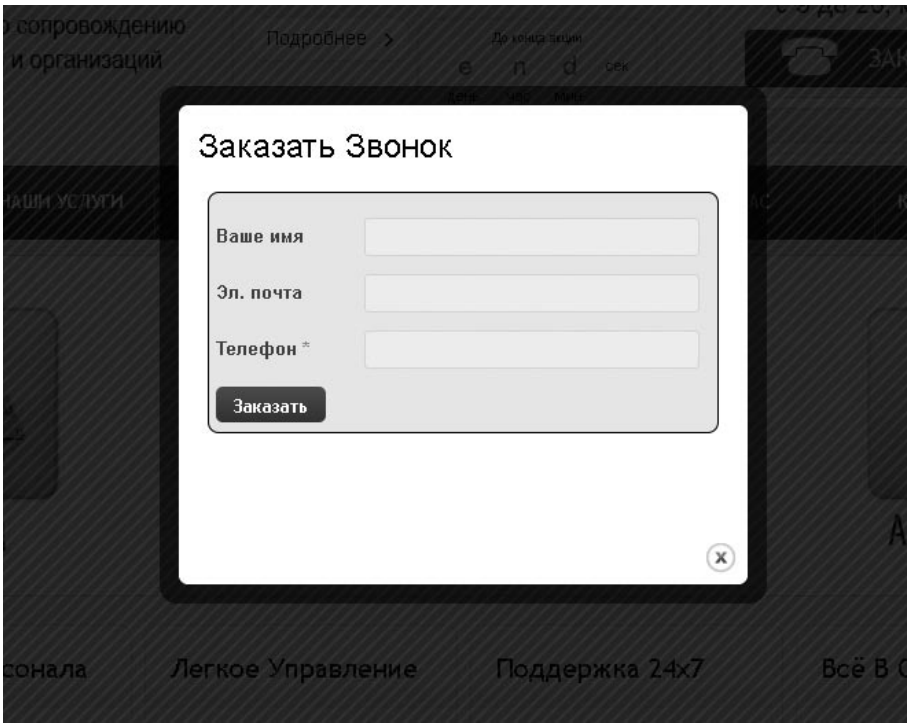


Рис. 3.15. Всплывающие окна

И добавьте JavaScript-код, вызывающий функцию `colorbox` при щелчке на ссылках с присвоенным вами классом:

```
jQuery(document).ready(function($){

   $(".group1").colorbox({rel:'group1'});

});
```

После этого при щелчке на любой ссылок с выбранным вами классом будет открываться окно, представленное на рис. 3.16.

Счетчик обратного отсчета (actions.js)

Для создания акцента на какой-либо акции или уникальном предложении часто используется счетчик обратного отсчета (рис. 3.17). Он позволяет побудить посетителя к покупке данного товара или услуги, повысив уровень продаж.

Рис. 3.16. Галерея изображений во всплывающем окне

Рис. 3.17. Счетчик обратного отсчета

Создать подобный счетчик обратного отсчета можно с помощью сценария `actions.js`, который идет вместе с файлами данной книги (находится в папке tech\actions.js), располагающимися на странице книги на сайте http://www.piter.com. Для него также необходима библиотека jQuery.

Подключите сценарий к своему HTML-документу, после чего добавьте разметку счетчика на веб-страницу:

```
<span class="action">

  <span class="action-date" textafter="" year="2012" month="4"
day="25"></span>

</span>

<div class="cur_action">
```

```
   <div class="title">До конца акции</div>

   <table><tbody><tr>

        <td class="tdwinitem tdwinitem1"><div class="posttd">
<div class="winitem"><span id="actionday"></span></div><div
class="windesc">день</div></div></td>

        <td class="tdwinitem tdwinitem2"><div class="posttd">
<div class="winitem"><span id="actionhour"></span></div><div
class="windesc">час</div></div></td>

        <td class="tdwinitem tdwinitem3"><div class="posttd">
<div class="winitem"><span id="actionmin"></span></div><div
class="windesc">мин</div></div></td>

        <td class="tdwinitem tdwinitem4"><div class="posttd">
<div class="winitem"><span id="actionsec"></span></div><div
class="windesc">сек</div></div></td>

   </tr></tbody></table>

</div>
```

Больше ничего делать не нужно.

Разметка счетчика обратного отсчета может быть любой. Главное, чтобы она содержала:

- тег с классом `action-date` и атрибутами:
  - `textafter` — текст, отображаемый при достижении счетчиком нулевого значения;
  - `year` — год окончания акции;
  - `month` — месяц окончания акции;
  - `day` — день окончания акции;
- теги с идентификаторами `actionday`, `actionhour`, `actionmin`, `actionsec`, в которых будет отображаться день, час, минута, секунда до конца акции.

PNG24 для Internet Explorer 6 (DD_belatedPNG.js)

Одной из проблем браузера Internet Explorer 6 является некорректное отображение изображений в формате PNG24 (с полупрозрачностью). По умолчанию данный браузер не способен показывать полупрозрачность. И полупрозрачные участки на изображении отображаются в нем серым цветом.

Для решения данной проблемы используется сценарий DD_belatedPNG.js. Он позволяет добавлять полупрозрачность как картинкам, присоединенным с помощью HTML, так и картинкам, вставленным через CSS (фоновым картинкам). Сценарий можно скачать на странице http://www.dillerdesign.com/experiment/DD_belatedPNG/.

Работать со сценарием достаточно легко. Просто добавьте в тег head вашего HTML-документа подобный код:

```
<!--[if IE 6]>

<script src="js/DD_belatedPNG.js"></script>

<script> DD_belatedPNG.fix('#logo-container, .picts'); </
script>

<![endif]-->
```

В функции DD_belatedPNG.fix через запятую перечисляются селекторы всех изображений в формате PNG24, используемых на странице.

Предзагрузка изображений (preloadCssImages.jQuery_v5.js)

Еще одним полезным сценарием является preloadCssImages.jQuery_v5.js. Он позволяет при открытии веб-страницы сразу загрузить все картинки, указанные в CSS, не дожидаясь их первого вызова.

В качестве примера рассмотрим случай, когда фоновая картинка пунктов вашего меню меняется при наведении указателя мыши.

Без использования данного сценария при первом наведении указателя мыши на пункт меню эффект ролловера (смены картинки) произойдет с задержкой. Браузеру сначала нужно будет скачать фоновую картинку, которая заменит собой текущую, а потом уже отобразить ее.

Если же используется сценарий preloadCssImages.jQuery_v5.js, все картинки будут загружены сразу после открытия веб-страницы. А значит, никакой задержки при первом наведении указателя мыши на пункт меню не произойдет.

Скачать данный сценарий можно со страницы http://code.google.com/p/mollify/source/browse/trunk/mollify/src/org/sjarvela/mollify/public/include/?r=710.

Подключите сценарий к HTML-документу. Затем вызовите функцию jQuery.preloadCssImages после полной загрузки веб-страницы:

```
jQuery(document).ready(function($){

   jQuery.preloadCssImages();

});
```

Слайдер (jcarousellite.js)

Слайдер позволяет:

- создать анимацию автоматической циклической смены ряда изображений (любых блоков HTML-кода, а не только изображений) (рис. 3.18, *вверху*);
- создать набор изображений, навигация по которым осуществляется стрелочками «влево/вправо» или колесиком мыши (рис. 3.18, *внизу*).

Рис. 3.18. Использование слайдера

Для создания слайдера можно использовать сценарий `jcarousellite.js`. Его можно скачать со страницы http://www.gmarwaha.com/jquery/jcarousellite/.

Помимо jQuery, для работы данного сценария могут пригодиться:

- `jquery.mousewheel.min.js` — поддержка навигации с помощью колесика мыши;
- `jquery.easing.1.3.js` — дополнительные эффекты ротации изображений.

Подключите все сценарии, после чего добавьте код слайдера. Например, такой:

```
<div class="prerotator_brands"><div class="rotator_brands">

   <ul class="items">

  <li><div class="views-field-tid"><span class="field-content"><a
href="/catalog?taxonomy_3=4&amp;fromb=1"><img src="t_0001.jpg"
alt="Anna Sui" title="Anna Sui" width="181" height="101" /></
a></span></div></li>

  <li><div class="views-field-tid"><span class="field-content"><a
href="/catalog?taxonomy_3=6&amp;fromb=1"><img src="t_0002.jpg"
alt="Armani" title="Armani" width="181" height="101" /></a></
span></div></li>

  <li><div class="views-field-tid"><span class="field-content"><a
href="/catalog?taxonomy_3=9&amp;fromb=1"><img src="t_0003.jpg"
alt="Balenciaga" title="Balenciaga" width="181" height="101"
/></a></span></div></li>

    <li class="views-row views-row-4 views-row-even"><div
class="views-field-tid"><span class="field-content"><a href="/
catalog?taxonomy_3=26&amp;fromb=1"><img src="brand_bebe.png"
alt="BEBE" title="BEBE" width="181" height="101" /></a></span></
div></li>

</ul>

</div></div>

<script> jQuery(function(){ jQuery(".prerotator_brands").
galleryCircle({ holderList: ".rotator_brands", scrollElParent:
"ul", scrollEl: "li", step: true, switchTime: true, duration :
1000, autoSlide:9000, funcOnclick:function(){ } }); }) </
script>
```

Это был автоматический слайдер. А это пример обычного слайдера:

```
<script type="text/javascript"> jQuery( function(){ jQuery(".
rotator").jCarouselLite({btnNext: ".next", btnPrev: ".prev",
mouseWheel: true, circular: false, visible: 4});} ); </script>

<div class="prerotator">

<img src="rotator_left.png" alt="" title="" class="prev" />

<img src="rotator_right.png" alt="" title="" class="next" />
```

```
<div class="rotator"><ul class="items">

   <li><div class="product_item"><a href="/jsnode/30" rel="li
ghtframe[search][Название товара]"><img src="bvlgari-jasmin-
noir-20080201-katemoss.jpg" alt="" title="" /><div class="views-
field-tid-1">Dior</div><div class="views-field-tid">test</
div><div class="views-field-sell-price"><span><span class="uc-
price-product uc-price-sell_price uc-price">3000 руб</span></
span></div></a></div></li>

   <li><div class="product_item"><a href="/jsnode/13" rel="li
ghtframe[search][Туника 1]"><img src="product4.jpg" alt=""
title="" /><div class="views-field-tid-1">Armani</div><div
class="views-field-tid">Туники</div><div class="views-field-sell-
price"><span>1395 руб</span><span class="discountyes"><span
class="uc-price-product uc-price-sell_price uc-price">1500 руб</
span></span></div></a><div class="flag_discount">-7%</div></
div></li>

   <li><div class="product_item"><a href="/jsnode/12" rel="li
ghtframe[search][Футболка 1]"><img src="product3.jpg" alt=""
title="" /><div class="views-field-tid-1">CK (Calvin Klein)</
div><div class="views-field-tid">Футболки</div><div class="views-
field-sell-price"><span>1395 руб</span><span class="discountye
s"><span class="uc-price-product uc-price-sell_price uc-
price">1500 руб</span></span></div></a><div class="flag_
discount">-7%</div></div></li>

  <li><div class="product_item"><a href="/jsnode/11" rel="light
frame[search][Сумка 1]"><img src="product2.jpg" alt="" title=""
/><div class="views-field-tid-1">Dior</div><div class="views-field-
tid">Аксессуары</div><div class="views-field-sell-price"><span>3150
руб</span><span class="discountyes"><span class="uc-price-product
uc-price-sell_price uc-price">3500 руб</span></span></div></a><div
class="flag_discount">-10%</div></div></li>

</ul></div></div>
```

В данной книге не ставилась цель досконально изучить JavaScript. Мы рассмотрели лишь основы, которые помогут нам при простой верстке и разработке сайтов.

Если вы хотите продолжить изучение JavaScript, рекомендую посетить сайт http://javascript.ru. Там вы узнаете много нового о данном языке.

Если же вы хотите подробнее узнать о jQuery, можно воспользоваться сайтами http://jquery.page2page.ru и http://jquery-docs.ru/.

Глава 4

PHP

Установка набора Denwer

Файлы PHP

Используем PHP

Вывод на экран

Переменные в PHP

Проверка содержимого

Конвертирование

Массивы

Условные операторы

Циклы

Функции

Данная совсем небольшая глава посвящена PHP. Мы изучим лишь основы этого языка программирования. Этого хватит для большинства работ по созданию сайтов на CMS Drupal.

Главное в PHP — знать основы языка. А с остальным можно знакомиться по мере необходимости. В любом случае все возможности языка PHP изучить невозможно. Ибо в стандартную поставку языка PHP входят тысячи стандартных функций. Не говоря уже о дополнительных библиотеках. Так что дальнейшее изучение языка PHP вы всегда сможете продолжить самостоятельно.

Начинать рассмотрение языка PHP лучше всего с главы книги, посвященной JavaScipt, так как основной синтаксис этих языков во многом совпадает. Как и толкование таких рассмотренных нами ранее понятий, как переменная, массив, цикл, рекурсивный цикл, условный оператор. В этой главе не будут заново объясняться все эти понятия. Так что, если вы еще не читали главу книги, посвященную JavaScript, сделайте это.

Установка набора Denwer

Перед тем как изучать возможности языка PHP, нам следует позаботиться о среде, в которой будут выполняться наши PHP-сценарии. Ведь по умолчанию в поставку операционной системы Windows не входит интерпретатор для языка PHP. Проще говоря, сценарии на PHP работать не будут.

Итак, перед тем, как создавать сценарии на PHP, необходимо установить интерпретатор языка. Его можно скачать на странице http://php.net/downloads.php.

Таким образом мы установим только интерпретатор. Но для работы веб-страниц, написанных на PHP, необходим также веб-сервер. Лучше всего для наших целей подойдет веб-сервер Apache. Его можно скачать на странице http://httpd.apache.org/download.cgi.

После установки PHP и Apache их нужно будет настроить на совместную работу.

Но и это еще не все. Сейчас ни один сайт на PHP не обходится без базы данных. В том числе и сайты на различных CMS. Поэтому вам также придется установить базу данных MySQL. Ее можно скачать на странице http://www.mysql.ru/download/.

В общем, это нелегкий путь. И, если честно, бессмысленный. Мы пойдем другим путем — установим набор для разработчиков Denwer. В него уже входят настроенные PHP, MySQL, Apache, а также интерфейс для отправки почты и интерфейс для работы с базой данных phpMyAdmin. А устанавливается Denwer за несколько минут.

Скачать последнюю версию набора Denwer можно на странице http://denwer.ru/. Вы можете также воспользоваться инсталлятором, который поставляется вместе с архивом файлов к данной книге и находится в папке tech\Denwer. (Вы можете скачать эти файлы на сайте www.piter.com.)

Итак, запустите установку набора Denwer. После подтверждения вами своих намерений откроется браузер Internet Explorer со страницей описания набора Denwer. Установка набора начнется сразу же, как только вы закроете браузер Internet Explorer.

После закрытия окна браузера перед вами отобразится командная строка с процессом установки набора Denwer (рис. 4.1). Убедитесь, что окно командной строки активно (на него установлен фокус), нажмите клавишу Enter и далее выполните следующие операции.

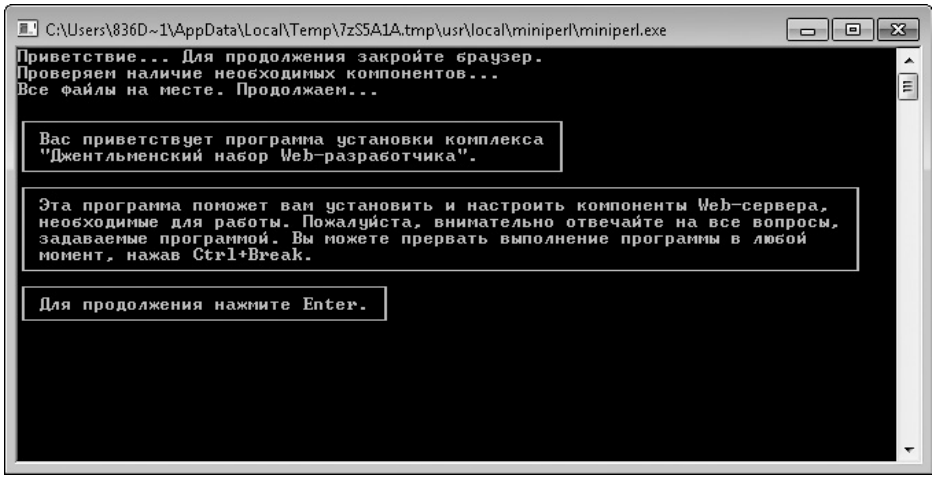


Рис. 4.1. Командная строка установки Denwer

1. Инсталлятор попросит вас ввести путь к папке, в которую будет установлен пакет Denwer. Лучше всего установить Denwer в какую-либо папку в корне диска. Вам потом будет удобнее открывать файлы сайтов, размещенных на Denwer. Например, в папку `C:\www` (конечно, если у вас есть раздел диска C:). Далее в книге для удобства папка с установленным Denwer будет называться просто `<Denwer>`.
2. После повторного нажатия Enter появится предложение указать букву для виртуального раздела диска, который будет автоматически создаваться при запуске Denwer. Этот виртуальный диск будет указывать на папку `<Denwer>`. Например, можете указать диск Z:, если он не используется в вашей системе. В дальнейшем вам достаточно будет открыть в проводнике диск Z:, чтобы попасть в папку `<Denwer>`.

3. После повторного нажатия Enter начнется процесс копирования файлов в указанную в первом шаге папку. Далее вам будет предложено ввести цифру 1 или 2:
   - 1 — виртуальный диск, букву которого вы указывали на втором шаге, будет создаваться при загрузке операционной системы;
   - 2 — виртуальный диск, букву которого вы указывали на втором шаге, будет создаваться при запуске Denwer.

   Какой из этих вариантов выбрать, зависит от вас. Мне больше подходит второй вариант.
4. Далее вам будет предложено решить, стоит ли создавать на Рабочем столе ярлыки для запуска и завершения работы Denwer. Лучше всего ввести букву y, то есть согласиться с созданием ярлыков. В дальнейшем их всегда можно будет удалить.
5. В конце процесса установки будет снова открыт браузер Internet Explorer. На этот раз с описанием дальнейших действий: как работать с установленным набором Denwer и каковы наиболее частые причины неполадок. Закройте это окно, и можно будет приступать к изучению рабочего пространства.

Итак, Denwer установлен и на Рабочем столе появились ярлыки для его запуска (рис. 4.2).

Рис. 4.2. Ярлыки для запуска, завершения и перезапуска набора Denwer

Пока Denwer запускать не нужно. Сначала создадим отдельный сайт, на котором в дальнейшем будем практиковаться в PHP и в работе с CMS Drupal.

Для этого откройте папку <Denwer>/home и создайте в ней папку с именем будущего сайта. Например, php.test. После этого внутри созданной папки создайте папку с именем www. В итоге должен получиться путь <Denwer>/home/php.test/www.

После запуска Denwer содержимое папки www будет доступно из браузера, если вы введете в адресной строке URL-адрес http://php.test. То есть название папки, в которой мы создавали папку www, будет адресом нашего будущего сайта (доменом).

Лучше не создавать папки с названиями реальных доменов. Например, если бы мы вместо папки php.test создали папку с названием microsoft.com, то, пока работает Denwer, мы не смогли бы открыть сайт корпорации Microsoft. Вместо него открывался бы наш локальный сайт.

После того как папка www создана, поместим в нее свой первый PHP-сценарий. Для этого создайте внутри папки www обычный текстовый документ. После чего смените расширение созданного документа на PHP. А имя на index. Получится текстовый документ с именем index.php. Это и будет наш первый сценарий.

Пока нам достаточно убедиться, что набор Denwer работает. Для этого откройте файл index.php в текстовом редакторе и напишите в нем любой текст. Например, «Проверка». Далее запустите Denwer, воспользовавшись ярлыком Start Denwer на Рабочем столе (если вы отказались от создания ярлыков на Рабочем столе, то запустите файл run.exe, расположенный в папке <Denwer>/denwer/).

О запуске Denwer сигнализирует появление двух значков в области уведомлений (рис. 4.3).

Рис. 4.3. Значки Denwer в области уведомлений

После появления значков Denwer в области уведомлений можно начинать работу с Denwer. Для этого запустите любой браузер и в адресной строке браузера введите адрес http://php.test. Если установка прошла успешно, появится введенная в файле index.php строка текста (рис. 4.4).

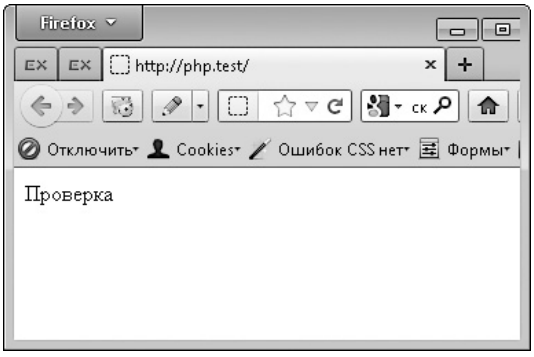


Рис. 4.4. Наш первый веб-сайт

ПРИМЕЧАНИЕ

Если вы назвали файл, расположенный в папке www, не index.php, то открыть его в браузере можно будет по адресу http://php.test/*имя файла*.php. Например, файл my.php в браузере будет доступен по адресу http://php.test/my.php. Главное, чтобы расширение у файла было PHP. А называться он может по-разному.

Поздравляю, у вас появился собственный веб-сайт. Пусть пока только локальный и недоступный из Интернета. Зато в остальных смыслах полноценный.

После того как вы закончили работать с Denwer, можете завершить работу локального веб-сервера и других компонентов, входящих в состав Denwer. Для этого используется ярлык Stop Denwer на Рабочем столе или файл `stop.exe`, расположенный в папке `<Denwer>`/denwer/.

Файлы PHP

Веб-сайты, написанные на языке PHP, состоят из файлов с расширением PHP, а не файлов с расширениями HTML или HTM. Фактически файл с расширением PHP — это тот же файл с расширением HTML, но только в нем можно использовать дополнительный синтаксис для работы с PHP. А можно и не использовать, это ваше дело.

Вы можете изменить расширение на PHP для всех HTML-файлов, которые создали ранее. И они все равно будут работать. Но, конечно, только если вы скопируете получившиеся файлы в папку `www` своего локального сайта. Ведь просто так ни один PHP-файл работать не будет.

Файлы PHP работают только в том случае, если:

- они размещены на хостинге с поддержкой PHP; неважно, на каком хостинге, локальном (набор Denwer) или в Интернете;
- вы запускаете PHP-файл из адресной строки браузера путем указания адреса сайта и имени PHP-файла (например, http://php.test/my.php).

Итак, в самом простом случае PHP-файлы представляют собой обычные HTML-документы. А это значит, что их содержимое должно быть точно таким же, как и содержимое HTML-файлов: сначала указывается `DOCTYPE`, потом тег `html`, внутри которого теги `head` и `body` и т. д. Разница появляется только тогда, когда вы начинаете использовать в PHP-файле вставки на языке PHP. Об этом мы поговорим в следующем разделе.

Используем PHP

Использование языка PHP начинается с конструкции `<?php ?>`. Лишь между тегами `<?php` и `?>` можно вводить код на языке PHP.

ПРИМЕЧАНИЕ

На самом деле вид конструкции, которая может обрамлять PHP-код, зависит от настроек PHP.

То есть:

```
<?php /* тег начала кода */

// ваш код на языке PHP

/* тег окончания кода */ ?>
```

Как можно заметить из примера, в PHP действует точно такой же синтаксис комментариев, как и в JavaScript. Но не будем отвлекаться.

Если вы попытаетесь ввести код PHP не между тегами `<?php` и `?>`, код будет воспринят как обычный HTML-текст.

Итак, перед тем, как передать PHP-документ браузеру, веб-сервер ищет в нем все теги `<?php` и `?>` и передает их содержимое на обработку PHP. Результат обработки заменяет собой содержимое тегов `<?php` и `?>` и сами эти теги. Таким образом, браузер получает обычный HTML-файл, без PHP-кода.

Отсюда можно сделать один важный вывод: язык PHP чаще всего используется для того, чтобы отобразить часть HTML-документа, структура которой может отличаться в зависимости от условий.

Теперь несколько слов о самом коде PHP. Как и в JavaScript, код в PHP состоит из отдельных команд, которые отделяются друг от друга точкой с запятой (`;`).

Вывод на экран

Чтобы вернуть содержимое, которое должно заменить собой текущую конструкцию `<?php ?>`, используется ключевое слово `echo` либо функция `print`.

Синтаксис ключевого слова `echo` следующий:

```
<?php

echo "какой-либо текст с возможностью использовать HTML";

?>
```

Функция `print` имеет следующий синтаксис:

```
<?php

print("какой-либо текст с возможностью использовать HTML");

?>
```

Например (рис. 4.5):

```
Проверка

<?php

echo '<h1>это тоже проверка</h1>';

echo '<br />кстати, это была проверка функции <strong>echo</
strong>';

print('<p>а это проверка функции <strong>print</strong></p>');

?>
```

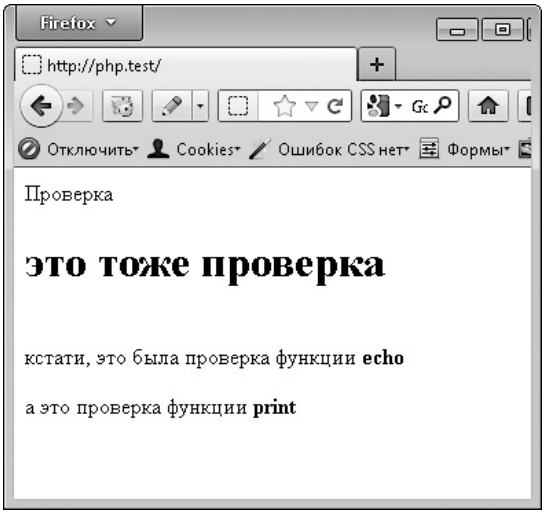


Рис. 4.5. Использование echo и print

Как можно заметить из примера, количество ключевых слов `echo` внутри одной конструкции `<?php ?>` может быть любым. Кроме того, вы можете использовать в одной конструкции `<?php ?>` и ключевое слово `echo`, и функцию `print`. Содержимое всех `echo`, `print` и других способов что-либо вывести на экран объединяется обработчиком PHP, после чего заменяет собой текущую конструкцию `<?php ?>`.

Переменные в PHP

Все переменные в PHP начинаются с символа `$`, за которым идет имя переменной. В именах переменных следует использовать только латинские буквы и цифры.

Причем начинаться имя переменной должно с латинской буквы. Например: `$var1`, `$test`, `$i`.

В PHP переменные не объявляются. Однако перед первым использованием переменной крайне рекомендуется задавать ей какое-либо начальное значение либо удалять саму переменную функцией `unset(переменная)`.

Это связано с безопасностью — в некоторых конфигурациях (в зависимости от настроек PHP) значения переменных можно передавать при открытии веб-страницы, через адресную строку браузера. Таким образом, не заданным явно переменным теоретически возможно задать значение через строку браузера. А это может привести к неожиданным результатам. Вплоть до взлома сайта.

В остальном использование переменных и арифметические операции над ними выполняются так же, как и в JavaScript:

```
$test = 23; // присвоение

$var1 = $test*2; // умножение

$var1 = $var1%2; // остаток от деления

$var1 = $var1/3; // деление

$var1 /= 3; // точно такое же деление

$var1 = $var1+17; // сложение

$var1 += 17; // точно такое же сложение

$var1 = 100-$var1; // вычитание

$var1 -= 100; // аналог команды $var1 = $var1 - 100;

$var1 = 1000-($var1*3+75); // математическое выражение

++$var1; // префиксный инкремент

$var1++; // постфиксный инкремент

--$var1; // префиксный декремент

$var1--; // постфиксный декремент
```

Единственное исключение — операция конкатенации. В PHP она выполняется с помощью точки (.), а не знака +. Например:

```
var $test = "моя первая"." строка, с ".$mystring;
```

Проверка содержимого

Такого понятия, как тип переменной, в PHP не существует. В одной и той же переменной может храниться и строка, и число, и массив, и какой-либо объект.

По этой причине в PHP весьма актуальны функции, позволяющие определить, значение какого типа в данный момент находится в переменной:

- большинство из этих функций начинается с `is`;
- все они возвращают `true` при положительной проверке и `false` при отрицательной; поэтому чаще всего их используют внутри условных операторов;
- все эти функции принимают один аргумент — переменную, которую нужно проверить.

isset. В первую очередь следует изучить функцию `isset`. Она возвращает значение `true`, если переменная, переданная в качестве аргумента, существует:

```
if(isset($var1)){
    $var2+=$var1;
}
```

Если же вам нужно проверить переменную на «несуществование»:

```
if(!isset($var1)){
    $var1=$var2;
}
```

empty. Проверить, содержит ли переменная какое-либо пустое или нулевое значение, можно с помощью функции `empty`. Данная функция возвращает `true`, если значением переменной является `""`, `0`, `"0"`, `NULL`, `FALSE`, `array()` либо объект с пустыми свойствами:

```
if(empty($var1)){
    // команды
}
```

is_numeric. Чтобы проверить, содержит ли переменная число (в том числе 0 или строку в виде числа), достаточно воспользоваться функцией `is_numeric(переменная)`. Ничего необычного в вызове этой функции нет — все точно так же, как и в ранее рассмотренных функциях.

Другие функции is_. Помимо функции `is_numeric`, существуют и другие функции, которые начинаются с префикса `is_`:

- `is_array` — проверяет, является ли массивом (в том числе пустым);
- `is_binary` — определяет, является ли бинарной строкой;
- `is_bool` — проверяет, является ли значение булевым (содержит ли значения `true` или `false`);
- `is_double`, или `is_float`, или `is_real` — определяет, является ли числом с плавающей точкой;
- `is_infinite` — проверяет, содержит ли положительную или отрицательную бесконечность;
- `is_finite` — определяет, является ли конечным числом в диапазоне чисел с плавающей точкой, с которыми PHP может выполнять какие-либо операции на данной платформе;
- `is_int`, или `is_integer`, или `is_long` — проверяет, является ли целым числом;
- `is_nan` — определяет, содержит ли значение `NaN` (не является числом);
- `is_null` — проверяет, содержит ли значение `NULL`;
- `is_object` — определяет, является ли объектом;
- `is_scalar` — проверяет, является ли значение скалярным (скалярными являются значения типа `integer`, `float`, `string` или `boolean`);
- `is_string` — определяет, является ли строкой;
- `is_unicode` — проверяет, является ли строкой в кодировке Unicode (UTF-16).

Конвертирование

Операции конвертирования в PHP представлены функцией `intval(переменная)`. Она пытается преобразовать значение переменной в число и возвращает результат своих попыток:

```
$res = intval(23); // $res равно 23

$res = intval("23"); // $res равно 23

$res = intval("23asd"); // $res равно 23

$res = intval("2w3"); // $res равно 2

$res = intval("w23"); // $res равно 0
```

Массивы

Создание массивов. Создать массив в PHP можно следующей строкой:

```
$mymas = array(значение1, значение2);
```

В результате первый элемент (элемент с индексом 0) массива `$mymas` будет содержать значение `значение1`. А второй элемент — `значение2`.

С помощью функции `array` можно также создавать ассоциативные массивы. Это делается следующим образом:

```
$массив = array(ключ1 => значение1, ключ2 => значение2);
```

Например:

```
$mymas = array(
"key1" => "значение1",
"key2" => "значение2"
);
```

В данном примере для удобства каждый элемент массива объявляется с новой строки.

Поскольку в PHP нет никаких ограничений по использованию значений в переменных, вы можете вообще не создавать массивы. А сразу присваивать значения его элементам.

Присваивание значений. Самый простой способ создать массив:

```
unset($mymas); //на всякий случай удаляем значения переменной
$mymas[0]="значение1";
$mymas[1]="значение2";
$mymas[7]="значение7";
```

Как видите, работа с элементами массива происходит точно так же, как и в языке JavaScript.

Есть, правда, одно очень полезное исключение. При присвоении начальных значений элементам массива можно вообще не указывать индексы:

```
unset($mymas); // на всякий случай удаляем
               // значения переменной
```

```
$mymas[]="значение1"; // $mymas[0] равно "значение1"

$mymas[]="значение2"; // $mymas[1] равно "значение2"

$mymas[]="значение7"; // $mymas[2] равно "значение7"
```

В этом случае значения будут присваиваться элементам массива последовательно, начиная с нулевого.

Длина массива. Для определения длины массива в PHP используется функция `count(массив)`. Вы можете использовать ее как для получения текущего количества элементов в массиве, так и для проверки массива на наличие элементов:

```
if(is_array($mymas) && count($mymas)){

   // массив существует и содержит элементы

}
```

Вывод содержимого массива. Довольно часто у разработчика возникает необходимость проверить, какая информация хранится в массиве. Решить данную задачу можно перебором значений всех элементов массива. Но гораздо проще воспользоваться специальной функцией `print_r`.

Функция `print_r` в виде строки выводит значение переданной в нее переменной. Переменная может быть любого типа. В том числе массивом или объектом.

Например (рис. 4.6, *слева вверху*):

```
$mymas = array(

   "key1" => "значение1",

   "key2" => "значение2"

);

print_r($mymas);
```

Приведенный в примере синтаксис функции `print_r` выводит содержимое переменной сразу в окно браузера. Как видно из рисунка, в окне браузера вывод функции `print_r` отображается в виде одной строки. Однако если открыть исходный код страницы, то можно просмотреть отформатированный вывод (рис. 4.6, *слева внизу*). Это удобно при выводе объектов со сложным содержимым.

Чтобы не открывать исходный код страницы, вместо приведенного выше синтаксиса можно воспользоваться следующим: `переменная2 = print_r(переменная1, true)`. В этом случае функция `print_r` вернет содержимое `переменной1` в `переменную2`. Например (рис. 4.6, *справа*):

```
$mymas = array(
   "key1" => "значение1",
   "key2" => "значение2"
);
echo '<pre>'.print_r($mymas, true).'</pre>';
```

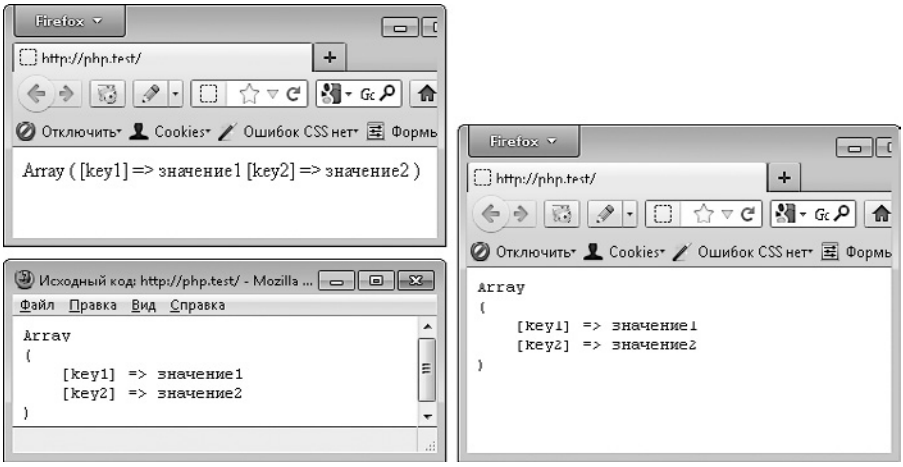


Рис. 4.6. Использование функции print_r

Многомерные массивы. Работать с многомерными массивами в PHP несложно. Достаточно вместо одной пары квадратных скобок использовать две пары (для двухмерного массива), три (для трехмерного) и т. д. Например, двухмерный массив:

```
unset($mymas);
$mymas[0][] = 'элемент [0][0]';
$mymas[0][] = 'элемент [0][1]';
$mymas[1][] = 'элемент [1][0]';
$mymas[1][] = 'элемент [1][1]';
$mymas[1][] = 'элемент [1][2]';
$mymas[1][] = 'элемент [1][3]';
$mymas[0][] = 'элемент [0][2] !!!';
echo $mymas[0][2];
```

Многомерные массивы можно также создавать с помощью функции `array`:

```
$mymas = array(
   array(10,21,32,43),
   array(50,61,72),
);
echo $mymas[0][2];
```

Условные операторы

Работа с условными операторами в PHP практически ничем не отличается от использования условных операторов в JavaScript.

if. Ниже приведены варианты синтаксиса оператора `if`. Подробнее о данном операторе читайте в главе 3, посвященной JavaScript.

Синтаксис с командами только для истинного условия:

```
if(условие1){ // если условие1 истинно, то
   команда1;  // выполнить команды в фигурных скобках
   команда2;
}
```

Синтаксис с командами для истинного и ложного условий:

```
if(условие1){ // если условие1 истинно, то
   команда1;  // выполнить команды в первых фигурных скобках
   команда2;
}else{        // если условие1 ложно, то
   команда3;  // выполнить команды во вторых фигурных скобках
   команда4;
}
```

Синтаксис с несколькими условиями:

```
if(условие1){
```

```
   команда1;
   команда2;
}else if(условие2){
   команда3;
   команда4;
}else{
   команда5;
   команда6;
}
```

ПРИМЕЧАНИЕ

В PHP вместо ключевого слова else if можно использовать синоним elseif.

?:. Основной синтаксис данного условного оператора ничем не отличается от синтаксиса в JavaScript:

```
переменная=(условие)?(значение1):(значение2);
```

`Переменной` присваивается `значение1`, если условие истинно. В противном случае `переменной` присваивается `значение2`.

switch. Синтаксис условного оператора `switch` также полностью совпадает с синтаксисом JavaScript:

```
switch(переменная){
case значение1:
   команды1;
   break;
case значение2:
   команды2;
   break;
default:
   команды3;
}
```

Условия. В целом синтаксис условий, которые используются в условных операторах, также не отличается от того, что применяется в JavaScript:

- `значение1 == значение2` — «если `значение1` равно `значению2`, то»;
- `значение1 != значение2` — «если `значение1` не равно `значению2`, то»;
- `значение1 === значение2` — «если `значение1` строго равно `значению2` (совпадает не только значение, но и тип значения), то»;
- `значение1 !== значение2` — «если `значение1` строго не равно `значению2` (операнды не равны и/или не совпадает их тип), то»;
- `значение1 > значение2` — «если `значение1` больше `значения2`, то»;
- `значение1 >= значение2` — «если `значение1` больше или равно `значению2`, то»;
- `значение1 < значение2` — «если `значение1` меньше `значения2`, то»;
- `значение1 <= значение2` — «если `значение1` меньше или равно `значению2`, то»;
- `!значение1` — «если `значение1` равно `false` (то есть не существует, либо равно `false`, `""` (пустая строка), `0`), то»;
- `значение1` — «если `значение1` равно `true` (то есть существует и не равно `false`, `""` (пустая строка), `0`), то».

Объединение условий. Для объединения нескольких условий в одном условном операторе может использоваться как синтаксис JavaScript, так и собственный синтаксис PHP.

Синтаксис JavaScript нам уже знаком:

- `&&` — логическое И, то есть «если и первое и второе условия истинны»;
- `||` — логическое ИЛИ, то есть «если истинно хотя бы одно из условий».

Собственный синтаксис PHP более понятен:

- `and` — логическое И, то есть «если и первое и второе условия истинны»;
- `or` — логическое ИЛИ, то есть «если истинно хотя бы одно из условий».

Например:

- `if(($var1 === false) and ($var2 > 23)){}` — если `$var1` содержит значение `false` и при этом значение переменной `$var2` больше 23;
- `if(($var1 == 10) or ($var1 == 20)){}` — если `$var1` равно 10 или 20;
- `if(((($var1 == 10) or ($var1 == 20)) and ($var2 > 23)){}` — если `$var1` равно 10 или 20 и при этом `$var2` больше 23.

Циклы

Для изучения циклов также следует открыть главу 3, посвященную JavaScript. Виды циклов и их синтаксис в JavaScript в большинстве случаев совпадают с синтаксисом одноименных циклов в PHP.

for. Цикл `for` позволяет выполнить набор команд указанное количество раз:

```
for(инициализация; условие; шаг){
    команды
}
```

Аргументы цикла `for`:

- `инициализация` — переменная-счетчик, определяющая, сколько еще раз нужно повторить выполнение команд;
- `условие` — условие, задающее, когда цикл будет завершен;
- `шаг` — правило, по которому будет изменяться значение заданной в первом аргументе переменной при завершении очередного шага цикла.

Например:

```
for($i=0; $i<10; $i++){
    // команды
}
```

foreach. Наконец-то мы нашли отличие PHP от JavaScript — в JavaScript цикла `foreach` нет. Впрочем, там есть аналогичный ему цикл `for in`.

Цикл `foreach` используется для работы с массивами.

Чтобы перебрать все элементы обычного массива (либо значения элементов ассоциативного массива), используется следующий синтаксис:

```
foreach($массив as $переменная){
    // команды
}
```

Данный цикл проходит по всем элементам массива `$массив`, помещая каждый из них в переменную `$переменная`. При первом проходе массива в `$переменной` находится первый элемент массива, при втором проходе — второй и т. д.

Например:

```
$mymas = array(
"key1" => "значение1",
"key2" => "значение2"
);
foreach($mymas as $value){
    echo $value;
}
```

Если вам нужно получить не только значения элементов ассоциативного массива, но и названия его ключей, следует использовать данный синтаксис:

```
foreach($массив as $ключ => $переменная){
    // команды
}
```

Данный цикл проходит по всем элементам массива `$массив`, помещая значение каждого из них в переменную `$переменная`, а ключ — в переменную `$ключ`.

Например:

```
$mymas = array(
"key1" => "значение1",
"key2" => "значение2"
);
foreach($mymas as $key => $value){
    echo $key.': '.$value;
}
```

while. Синтаксис цикла `while` не отличается от JavaScript:

```
while(условие){
   команды
}
```

Цикл будет выполняться до тех пор, пока условие истинно.

do while. В синтаксисе цикла `do while`, который применяется в PHP, есть небольшое отличие от синтаксиса JavaScript. Наверное, вы его даже не заметите:

```
do{
   команды
}while(условие);
```

Отличие заключается в точке с запятой после круглых скобок условия. Без этой точки с запятой цикл работать не будет.

Простой пример:

```
$d=10;
do{
    echo $d;
    $d--;
}while($d>3);
```

break и continue. Во всех видах циклов в PHP можно использовать ключевые слова `break` и `continue`:

- `break` — незамедлительно завершить выполнение цикла;
- `continue` — перейти к следующему шагу цикла, пропуская выполнение команд, расположенных ниже вызова `continue`.

Функции

Чтобы получить базовые знания по языку PHP, нам осталось лишь рассмотреть синтаксис применения функций.

Как само понятие функции, так и основные способы их использования в PHP не отличаются от JavaScript.

Создаем функцию. Функции без аргументов и без возвращаемого значения создаются следующим образом:

```
function имя(){
   команды
}
```

Если вам нужно вернуть какое-либо значение из функции, используется оператор `return`:

```
function имя(){

   команды

   return значение;

}
```

При необходимости передавать аргументы в функцию применяется следующий синтаксис:

```
function имя(аргумент1, аргумент2, аргумент3){

   команды1

   return значение;

}
```

В PHP более строго выполняется разделение областей видимости переменных. И из функции нельзя просто так получить значения переменных, объявленных за пределами функции. Впрочем, никто не говорит, что это невозможно.

Чтобы получить значение переменной за пределами функции, нужно объявить эту переменную внутри функции следующим образом: `global $переменная;`. Например:

```
$d=10;

test();


function test(){

    global $d;

    echo $d; // отобразит в окне браузера строку "10"

}
```

Вызываем функцию. Варианты вызова функций:

- `имя_функции()` — если функция не имеет аргументов и не возвращает никакого значения;

- `имя_функции(значение_для_аргумента1, значение_для_аргумента2)` — если функция имеет два аргумента, но не возвращает никакого значения;
- `переменная = имя_функции(значение_для_аргумента1, значение_для_аргумента2)` — если функция имеет два аргумента и возвращает значение.

На этом мы закончим изучение языка PHP. Мы рассмотрели лишь самые основы данного языка, но в рамках настоящей книги больше и не нужно. Язык PHP нам понадобится лишь для создания тем оформления для CMS Drupal, а также для написания несложных сниппетов для данной CMS. Но об этом далее в книге.

Продолжить изучение языка PHP вам поможет сайт http://php.net/manual/ru/langref.php.

Глава 5

CMS Drupal

Устанавливаем CMS

Модули

Основные понятия

Мы с вами уже изучили HTML, CSS, JavaScript, jQuery и PHP. Теперь перед вами две дороги:

- либо изучать какую-либо CMS и создавать сайты на ней, тратя на каждый сайт одну-две недели;
- либо создавать сайты с нуля, тратя на каждый сайт много месяцев работы.

Как вы догадались, в этой книге был выбран первый путь. И сейчас мы с вами рассмотрим возможности CMS Drupal.

Что же вообще такое CMS? Фактически CMS — это уже готовый, написанный кем-то другим сайт. Вам предлагается использовать чужие разработки, чтобы на их основе создать свой собственный сайт. Это гораздо быстрее, чем разрабатывать сайты с нуля.

CMS Drupal считается одной из лучших бесплатных CMS. Для нее разработаны тысячи сторонних модулей, расширяющих функционал системы. Хотя у нее есть и недостатки.

В первую очередь, она довольно сильно нагружает веб-сервер. Но этот недостаток можно побороть с помощью различных модулей кэширования.

Кроме того, для создания полноценных сайтов на CMS Drupal нужно знать не только клиентские технологии (HTML, CSS, JavaScript), но и серверные (PHP). Однако базовые знания вы уже имеете.

На данный момент последней версией CMS Drupal считается версия 7. Именно ее мы будем рассматривать в книге. Хотя для некоторых задач больше подходит предыдущая версия CMS Drupal — Drupal 6. В частности, полноценные интернет-магазины с поддержкой электронных валют лучше создавать на Drupal 6, так как для Drupal 7 до сих пор нет подходящих модулей, позволяющих принимать электронные деньги.

Устанавливаем CMS

Знакомство с CMS Drupal начинается с сайта http://drupal.org. Это официальный сайт данной CMS. Именно оттуда берутся последние версии CMS Drupal, а также сторонние модули и темы оформления для данной CMS. А если вы владеете английским языком, то сможете почерпнуть оттуда еще и массу полезных знаний по данной CMS (сайт на английском языке).

На сайте http://drupal.org все модули, темы оформления, а также сама CMS Drupal находятся на страницах вида http://drupal.org/project/«имя модуля».

Скачиваем CMS Drupal. Скачаем последнюю версию CMS Drupal. Для этого откройте страницу http://drupal.org/project/drupal. В нижней части страницы будут ссылки на скачивание (рис. 5.1). Из верхней таблицы можно скачать полностью готовые проекты. А в нижней таблице располагаются те версии проектов, которые находятся на стадии разработки. Это не значит, что ссылки из нижней таблицы ведут на нерабочие проекты. Они работают, просто при их использовании больше вероятности обнаружить какую-либо ошибку.

Downloads

Recommended releases

Version	Downloads	Date	Links
7.12	tar.gz (2.95 MB) | zip (3.36 MB)	2012-Feb-01	Notes
6.25	tar.gz (1.05 MB) | zip (1.22 MB)	2012-Feb-29	Notes

Development releases

Version	Downloads	Date	Links
7.x-dev	tar.gz (2.97 MB) | zip (3.4 MB)	2012-Apr-26	Notes
6.x-dev	tar.gz (1.05 MB) | zip (1.22 MB)	2012-Apr-17	Notes

View all releases

Рис. 5.1. Вид страницы загрузки модулей, тем оформления и самой CMS Drupal

Скачайте последнюю версию CMS Drupal 7 (на рис. 5.1 это версия 7.12).

Подготавливаем хостинг. Устанавливать CMS Drupal мы будем на набор Denwer. Вы ведь установили его по инструкции, приведенной в главе 4?

Поэтому следующим нашим шагом будет выход из Denwer (если он в данный момент запущен) и создание нового сайта. После выхода откройте папку `<Denwer>/home` и создайте в ней папку для вашего сайта, например, с именем `denwer.my`. Внутри этой папки создайте папку `www`. У вас получится путь `<Denwer>/home/denwer.my/www`.

Распакуйте архив с CMS Drupal в папку `<Denwer>/home/denwer.my/www`. В итоге содержимое папки `www` должно выглядеть так, как показано на рис. 5.2.

Если все совпадает, запустите Denwer и откройте в браузере страницу http://denwer.my (или как вы назвали созданную внутри `<Denwer>/home` папку?). Появится страница установки CMS Drupal (рис. 5.3).

ПРИМЕЧАНИЕ

Далее в книге в URL-адресах вместо домена denwer.my (или ваше имя папки, созданной внутри ‹Denwer›/home) будет использоваться аббревиатура ‹drupal›.

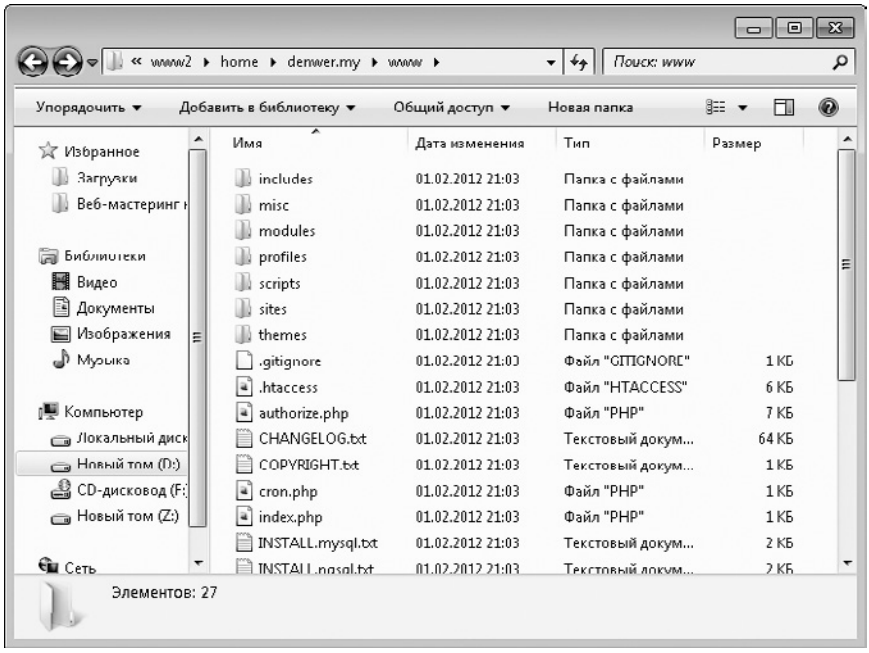


Рис. 5.2. Содержимое папки www

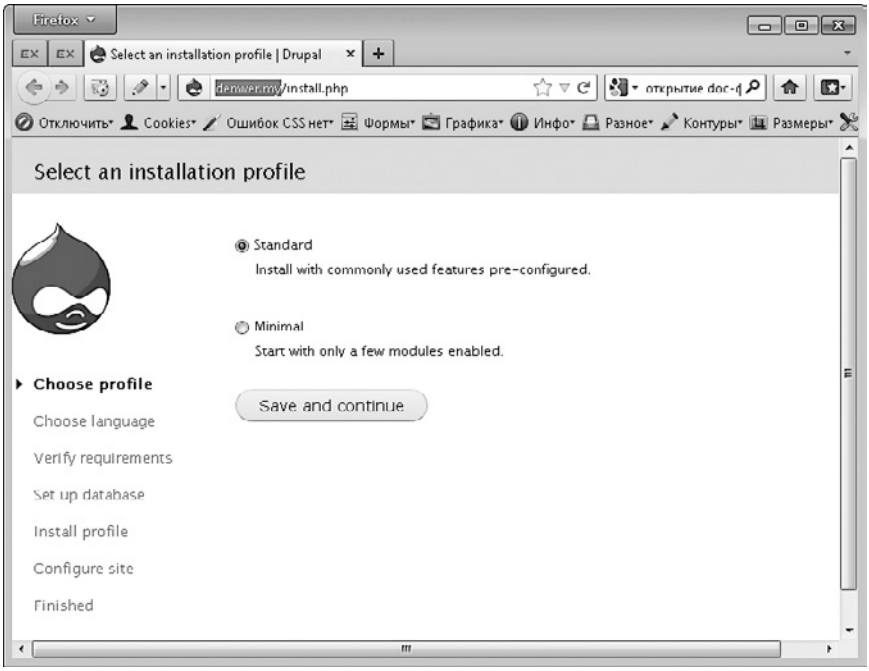


Рис. 5.3. Страница установки CMS Drupal

Создаем базу данных. Перед тем как начать установку CMS Drupal, нам необходимо сделать еще одно действие, а именно создать базу данных, которая в дальнейшем будет использоваться для CMS Drupal. В этом нам поможет сервис phpMyAdmin, который входит в состав набора Denwer.

В окне браузера откройте страницу http://localhost/Tools/phpmyadmin/index.php (набор Denwer при этом должен быть запущен). Откроется интерфейс администрирования баз данных. В этот момент из всех возможностей phpMyAdmin нам необходимо только поле Create new database (Создать новую базу данных). Введите в этом поле имя базы данных, например bd. После чего нажмите кнопку Create (Создать) (рис. 5.4).

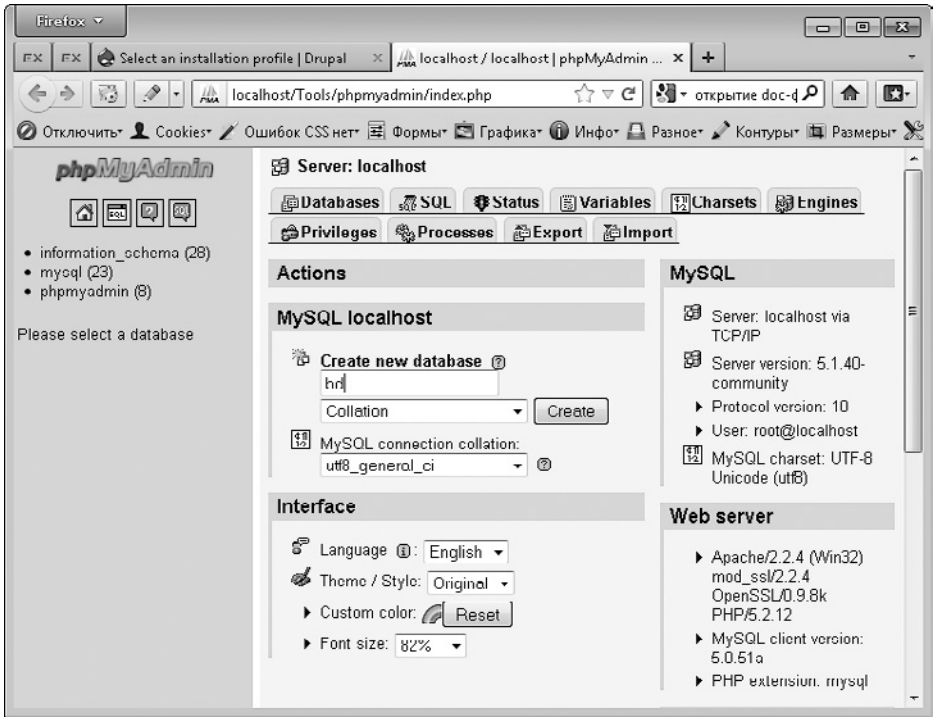


Рис. 5.4. Создание базы данных

Установка CMS Drupal. Все подготовительные шаги выполнены. Теперь можно переходить к установке.

1. На странице http://denwer.my установите переключатель в положение Standard (Стандартная) и нажмите кнопку Save and continue (Сохранить и продолжить).
2. На этапе выбора языка интерфейса еще раз нажмите кнопку Save and continue (Сохранить и продолжить). Русифицировать CMS Drupal мы будем уже после установки.

3. На открывшейся странице для настройки конфигурации базы данных (рис. 5.5) выполните следующее:
   - выберите положение переключателя Database type (Тип базы данных), в котором есть MySQL;
   - в поле Database name (Имя базы данных) введите имя созданной вами базы данных (ранее мы создали базу с именем `bd`);
   - в поле Database username введите логин для подключения к базе данных (для набора Denwer введите логин `root`);
   - в поле Database password (Пароль для базы данных) введите пароль для подключения к базе данных (для набора Denwer оставьте это поле пустым);
   - раскройте область Advanced Options (Дополнительные параметры). Большинство отобразившихся дополнительных полей для набора Denwer заполнять не нужно. Лишь в поле Table prefix (Префикс таблицы) можно ввести префикс, который будет добавляться всем создаваемым CMS Drupal таблицам. Это позволит повысить безопасность системы.

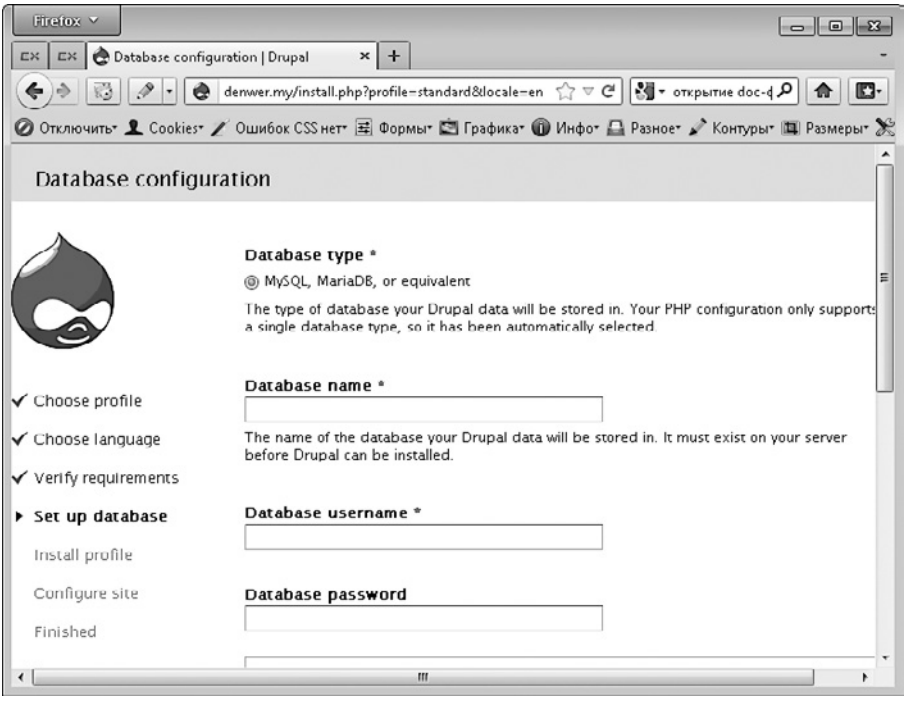


Рис. 5.5. Шаг настройки подключения к базе данных

4. После настройки подключения к базе данных начнется процесс установки CMS Drupal.

5. Завершающим этапом установки будет ввод дополнительных данных:
   - имени сайта;
   - электронного адреса, используемого в качестве адреса сайта (этот адрес будет указываться в поле От всех писем, создаваемых CMS Drupal, например письма с паролем, которое будет приходить посетителю при создании учетной записи на сайте);
   - логина, электронного адреса и пароля административной учетной записи и т. д.
6. Нажмите ссылку Visit your new site (Посетите свой новый сайт) на странице подтверждения успешной установки CMS Drupal.

В результате откроется ваш будущий сайт. При этом вы уже будете авторизированы на нем в качестве администратора системы (с тем логином и паролем, которые вы указали на шаге 5 установки CMS Drupal) (рис. 5.6).

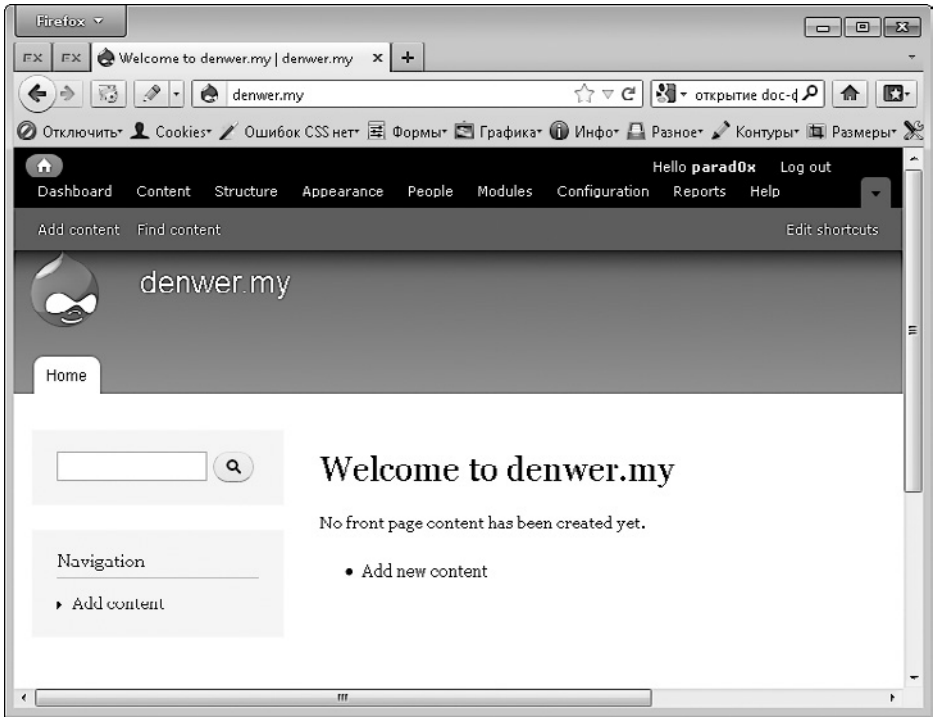


Рис. 5.6. Установленный сайт CMS Drupal

Русификация CMS Drupal. Следующим шагом после установки CMS Drupal чаще всего идет ее русификация. Для этого откройте страницу http://localize.drupal.org/

translate/languages/ru и скачайте перевод для 7-й версии CMS Drupal. В итоге вы станете обладателем файла drupal-7.11.ru.po.

Чтобы русифицировать CMS Drupal, нам сначала нужно включить стандартный модуль CMS Drupal Locale (Локальный). Для этого на административной панели нажмите Modules (Модули) (<drupal>/admin/modules) и на открывшейся странице установите флажок Locale (Локальный). После чего нажмите кнопку Save configuration (Сохранить конфигурацию).

Далее откройте страницу Configuration ▸ Languages (Конфигурация ▸ Языки) (<drupal>/admin/config/regional/language), то есть на административной панели нажмите Configuration (Конфигурация). На открывшейся странице перейдите по ссылке Languages (Языки). Далее в книге подобные записи будут сокращаться до вида Configuration ▸ Languages (Конфигурация ▸ Языки).

На открывшейся странице воспользуйтесь ссылкой Add language (Добавить язык), чтобы добавить русский язык. После чего в списке языков сделайте русский языком по умолчанию (установите переключатель Default (По умолчанию) напротив русского языка).

После добавления на сайт русского языка необходимо импортировать на сайт строки перевода, которые находятся в файле `drupal-7.11.ru.po`. Для этого откройте страницу Configuration ▸ Translate interface ▸ Import (Конфигурация ▸ Перевести интерфейс ▸ Импорт) (<drupal>/admin/config/regional/translate/import) и загрузите файл `drupal-7.11.ru.po`, выбрав в раскрывающемся списке Import into (Импортировать в) язык Russian (Русский) (рис. 5.7).

Модули

CMS Drupal построена по модульной системе. Весь функционал данной CMS разбит на отдельные модули. Вы можете отключать модули с ненужным функционалом, тем самым ускоряя работу сайта и избавляясь от ненужных функций.

Для работы с модулями предназначен раздел Модули (<drupal>/admin/modules). В нем содержится список всех модулей, установленных в системе. Вы можете включать и отключать их, а также переходить к странице настройки и изменения разрешений для включенных модулей.

Стандартные модули

Разделим все стандартные модули CMS Drupal на те, которые включены по умолчанию, и те, которые по умолчанию отключены. Это будет здравое разделение. Ведь

сейчас перед нами стоит задача отключить те из включенных по умолчанию модулей, которые нам не нужны. И включить те из модулей, которые по умолчанию не включены, но очень полезны.

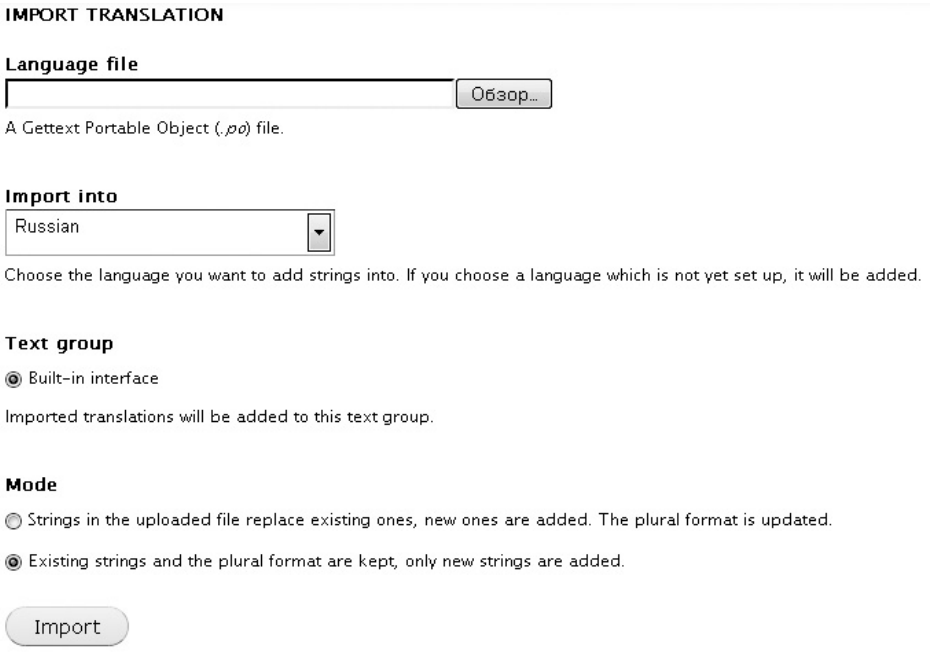


Рис. 5.7. Страница импортирования русских строк

Модули, включенные по умолчанию

Среди модулей, включенных по умолчанию, вы можете обнаружить следующие.

- Block — разбивает страницу сайта на отдельные блоки и позволяет добавлять в эти блоки любую информацию. Это базовый модуль, который не стоит отключать.
- Color — позволяет из административного интерфейса изменять цвета темы оформления и страниц, добавляемых сторонними модулями. Однако для этого как тема оформления, так и сторонние модули должны поддерживать данную возможность. Этот модуль можно отключить. Он используется крайне редко.
- Comment — добавляет возможность комментирования публикаций на сайте. Если комментарии вам точно не нужны, вы можете отключить данный модуль. Если же в этом вопросе вы не уверены, то, вместо отключения модуля, лучше запретить комментирование для созданных типов материалов.
- Contextual links — крайне полезный для администратора модуль. Он добавляет видимые только администратору контекстные меню к элементам интерфейса

страницы, позволяющие быстро переходить на страницы редактирования данного элемента.

- Dashboard — данный модуль добавляет админ-меню. Его можно отключить. Мы заменим его функционал более мощным сторонним модулем.
- Database logging — записывает события, происходящие в системе, в журнал. С целью повышения производительности сайта данный модуль по окончании разработки часто отключают.
- Field — добавляет возможность создания дополнительных полей для типов материалов, профилей пользователей, таксономии (подробнее о ней читайте ниже). Это базовый модуль, который не стоит отключать.
- Field SQL storage — также добавляет возможность создания дополнительных полей для типов материалов, профилей пользователей, таксономии. Это базовый модуль, который отключать не стоит.
- Field UI — добавляет административный интерфейс для создания/редактирования дополнительных полей. Это базовый модуль, который не стоит отключать.
- File — позволяет пользователям сайта прикреплять к материалам различные файлы. Данный модуль лучше не отключать. Он всегда может пригодиться.
- Filter — предназначен для создания фильтров ввода, ограничивающих теги, которые можно использовать на сайте при создании публикаций. Данный модуль также не стоит отключать.
- Help — отображает справку для административных разделов CMS Drupal. После изучения CMS данный модуль лучше отключать.
- Image — позволяет пользователям публиковать на сайте изображения (в том числе прикреплять их к публикациям). Данный модуль лучше не отключать. Он всегда может пригодиться.
- List — дает возможность создавать поля со списком значений. Данный модуль лучше не отключать.
- Menu — позволяет создавать различные меню на сайте. Отключать данный модуль не стоит.
- Node — предназначен для создания публикаций на сайте. Также не стоит отключать.
- Number — позволяет создавать численные поля. Данный модуль лучше не отключать.
- Options — расширяет возможности полей. Данный модуль лучше не отключать.
- Overlay — отображает административный интерфейс поверх самой страницы сайта. Весьма спорный модуль. Без него пользоваться сайтом удобнее. Поэтому данный модуль лучше отключить.

- Path — позволяет создавать синонимы для страниц сайта (например, обращаться к какой-либо публикации не по адресу <drupal>/node/111, а по более информативному <drupal>/article/kak_ya_provel_leto). Данный модуль лучше не отключать.
- RDF — добавляет метаданные в исходный код страниц. Эра HTML5 наступит еще очень нескоро. Поэтому данный модуль лучше отключить.
- Search — добавляет поиск по сайту. Поиск работает аналогично поисковым системам Google, «Яндекс» и т. д., то есть сначала происходит индексация всей информации на сайте (занесение в базу данных отдельных слов и словосочетаний). После чего поиск выполняется по созданному индексу.
- Shortcut — позволяет создавать ярлыки для быстрого открытия разделов сайта. Вместо использования данного модуля лучше применять более функциональный сторонний. Поэтому отключим данный модуль.
- System — базовый модуль, не отключайте его.
- Taxonomy — добавляет поддержку таксономии, то есть возможность создания и присвоения публикациям и другим видам материалов тегов. Данный модуль лучше не отключать.
- Text — позволяет создавать текстовые поля. Данный модуль лучше не отключать.
- Toolbar — отображает административную панель. Данный модуль лучше отключить, а вместо него установить более функциональный модуль Admin menu.
- User — управляет пользователями и процессом авторизации на сайте. Данный модуль лучше не отключать.

Модули, отключенные по умолчанию

Теперь рассмотрим модули, которые вы при желании можете включить.

- Aggregator — предназначен для автоматического сбора информации со сторонних сайтов. Принцип действия следующий. Вы указываете адрес RSS-канала сайта, а модуль периодически открывает RSS и переносит на ваш сайт обнаруженные в RSS новые публикации.
- Blog — добавляет каждому зарегистрированному пользователю собственный блог.
- Book — позволяет организовывать публикации на сайте в книги, то есть связывать публикации между собой, назначая родительские и дочерние публикации, а также публикации, которые идут перед и после текущей.
- Contact — добавляет на сайт страницу с формой быстрой связи, с помощью которой любой посетитель легко сможет написать вам сообщение. Вместо данного модуля лучше использовать более функциональный сторонний модуль Webform.

- Content translation — предназначен для создания мультиязычных сайтов.
- Forum — добавляет форум на сайт.
- Locale — используется для перевода содержимого сайта на другие языки. Мы уже включили данный модуль.
- OpenID — позволяет производить авторизацию на сайте с помощью OpenID.
- PHP filter — включите данный модуль, он может вам пригодиться. Он добавляет возможность использования PHP при создании публикаций, блоков и добавлении любой другой текстовой информации.
- Poll — позволяет проводить на сайте голосования.
- Statistics — чаще всего используется, чтобы добавить к публикациям счетчик, показывающий количество просмотров публикации.
- Syslog — предназначен для более подробного журналирования происходящих на сайте событий.
- Testing — может использоваться в процессе разработки сайта.
- Tracker — предназначен для отслеживания последнего содержимого для конкретного пользователя.
- Trigger — позволяет назначать действия, автоматически выполняемые при возникновении определенных событий на сайте. Вместо стандартного интерфейса действий лучше использовать сторонний модуль Rules.
- Update manager — предназначен для информирования администратора о наличии обновлений для установленных сторонних модулей, тем оформления и самой CMS Drupal. Позволяет также устанавливать обновления в автоматическом режиме. Однако автоматическим обновлением лучше не пользоваться, поскольку такое обновление часто приводит к проблемам в работе сайта.

Сторонние модули

Как правило, стандартных модулей CMS Drupal не хватает для создания полноценного сайта. Поэтому ниже будет рассмотрен список сторонних модулей, которые я использую чаще всего при разработке сайтов.

Однако сначала стоит рассказать о том, как вообще устанавливаются на сайт новые модули.

Установка модулей

Итак, вы скачали на свой компьютер архив с модулем, который хотели бы установить. Скорее всего, вы сделали это с сайта http://drupal.org, хотя это и не обязательно — некоторые полезные модули размещены в других источниках. Что же дальше делать?

Распакуйте полученный архив в текущую папку. У вас появится папка с именем модуля.

Далее откройте FTP своего сайта и перейдите в папку <корень сайта>/sites/all/modules. Если на вашем FTP папки modules не существует, создайте ее.

Осталось лишь перенести папку стороннего модуля на FTP вашего сайта в папку <корень сайта>/sites/all/modules.

Как правило, на этом установка модуля заканчивается. Вам остается лишь зайти на страницу Модули (<drupal>/admin/modules) и включить новый модуль.

В редких случаях для работы стороннего модуля необходимы какие-либо библиотеки. В этом случае, помимо перечисленных выше шагов, нужно будет также скачать эти библиотеки, распаковать их и с помощью FTP поместить их в папку <корень сайта>/sites/all/libraries на вашем хостинге. Если папки libraries на хостинге не существует, ее нужно создать.

В любом случае в папке практически любого стороннего модуля есть файл `readmy.txt`, открыв который вы сможете прочитать, что именно нужно сделать, чтобы установить данный модуль. Иногда, правда, процесс установки модуля описывается в другом файле — файле `install.txt`.

Итак, для CMS Drupal 7 созданы тысячи сторонних модулей. Не все из них могут показаться вам полезными. Но вообще без сторонних модулей, как правило, не обходится ни один сайт на Drupal.

Ниже будут перечислены модули, которые я часто использую при разработке сайтов. Это не значит, что, помимо перечисленного ниже списка, других полезных модулей для CMS Drupal нет. Данный список состоит лишь из тех модулей, которые понадобились мне для решения поставленных задач. Возможно, для своего сайта вам понадобятся и другие сторонние модули, не входящие в перечисленный список.

Для решения популярных задач существует более одного стороннего модуля. Ниже для каждой задачи будет рассмотрен только один модуль. Опять-таки это не означает, что рассмотренный модуль самый лучший. Возможно, вам понравится или лучше подойдет для решения поставленных задач другой модуль. В общем, это просто список модулей, которые я считаю полезными.

Зависимые модули

В первую очередь следует установить модули, которые необходимы для работы других модулей. Как правило, такие модули ничего не добавляют на сайт. Они просто содержат различные функции и сервисы, которые требуются для работы других модулей.

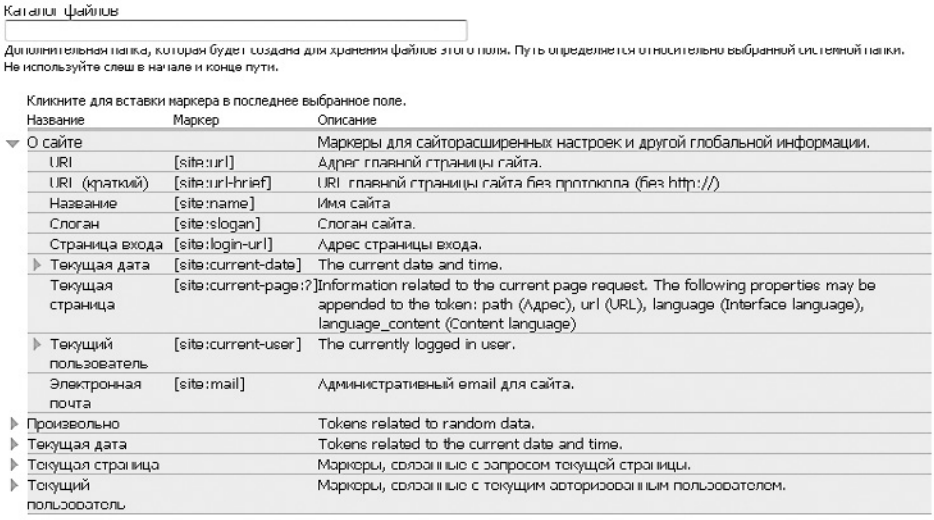
Chaos tool suite (ctools) (http://drupal.org/project/ctools). Требуется для работы многих сторонних модулей. В том числе Display suite, Meta tags, Ajax Driven Cart, Ubercart, Product Power Tools, Onpay, Date Views (часть модуля Date), Quicktabs, Views translation (часть модуля i18n), Variable views (часть модуля Variable), Fieldgroup, Views, Views Slideshow: Cycle, Service Links Displays (часть модуля Service links).

Entity API (http://drupal.org/project/entity). Также необходим для работы многих модулей: Rules, LoginToboggan Rules Integration (часть модуля LoginToboggan), Privatemsg Rules Integrations (часть модуля Privatemsg), Simplenews rules (часть модуля Simplenews), Ubercart, Ajax Driven Cart, Onpay.

Libraries API (http://drupal.org/project/libraries). Используется модулями Superfish и Views Slideshow.

Session API (http://drupal.org/project/session_api). С помощью данного модуля можно расширить функционал модуля Flags. Модуль Session API добавляет возможность работы с закладками для анонимных пользователей.

Token (http://drupal.org/project/token). Полезный модуль, упрощающий процесс разработки и администрирования сайта. Он добавляет набор переменных (имя сайта, электронный адрес пользователя, nid материала и т. д.), которые можно использовать в различных текстовых полях (рис. 5.8). Данный модуль также необходим для работы модулей File (Field) Paths, Meta tags, Page Title, Pathauto.


Каталог файлов

Дополнительная папка, которая будет создана для хранения файлов этого поля. Путь определяется относительно выбранной системной папки. Не используйте слеш в начале и конце пути.

Кликните для вставки маркера в последнее выбранное поле.

Название	Маркер	Описание
О сайте		Маркеры для сайторасширенных настроек и другой глобальной информации.
URI	[site:url]	Адрес главной страницы сайта.
URI (краткий)	[site:url-brief]	URI главной страницы сайта без протокола (без http://)
Название	[site:name]	Имя сайта
Слоган	[site:slogan]	Слоган сайта.
Страница входа	[site:login-url]	Адрес страницы входа.
Текущая дата	[site:current-date]	The current date and time.
Текущая страница	[site:current-page:?]	Information related to the current page request. The following properties may be appended to the token: path (Адрес), url (URL), language (Interface language), language_content (Content language)
Текущий пользователь	[site:current-user]	The currently logged in user.
Электронная почта	[site:mail]	Административный email для сайта.
Произвольно		Tokens related to random data.
Текущая дата		Tokens related to the current date and time.
Текущая страница		Маркеры, связанные с запросом текущей страницы.
Текущий пользователь		Маркеры, связанные с текущим авторизованным пользователем.

Рис. 5.8. Использование модуля Token

Variable (http://drupal.org/project/variable). Требуется для работы модулей, относящихся к созданию мультиязычных сайтов.

Voting API (http://drupal.org/project/votingapi). Используется модулями, предназначенными для присвоения рейтинга различным материалам.

Администрирование сайта

Далее следует позаботиться об администрировании сайта.

Administration menu (http://drupal.org/project/admin_menu). Добавляет удобное раскрывающееся меню, содержащее ссылки на все административные разделы сайта (рис. 5.9). Данное меню доступно только администраторам. По своим возможностям оно превосходит стандартный модуль Toolbar.

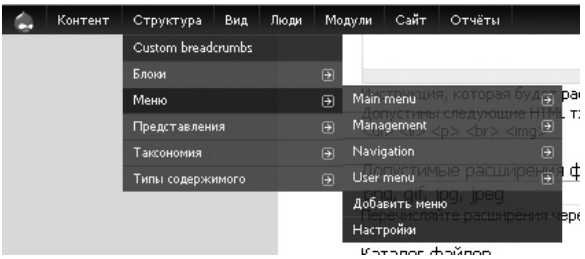


Рис. 5.9. Использование модуля Administration menu

ПРИМЕЧАНИЕ

Далее в книге все пути к административным страницам CMS Drupal будут указываться с помощью административной панели, добавляемой модулем Administration menu.

Backup and Migrate (http://drupal.org/project/backup_migrate). Позволяет создавать архивы содержимого базы данных вашего сайта, а также восстанавливать сайт из созданного ранее архива.

DB Maintenance (http://drupal.org/project/db_maintenance). Автоматически выполняет оптимизацию таблиц базы данных вашего сайта.

Структура сайта

Следующий набор модулей позволит вам расширить функционал сайта.

Block Title Link (http://drupal.org/project/block_titlelink). Дает возможность назначать ссылки заголовкам любых блоков в CMS Drupal.

Colorbox (http://drupal.org/project/colorbox). Позволяет отображать любой контент (изображения, страницы сайта, содержимое блоков) во всплывающих окнах (рис. 5.10) (в главе 4, посвященной JavaScript, мы уже рассматривали сценарий ColorBox).

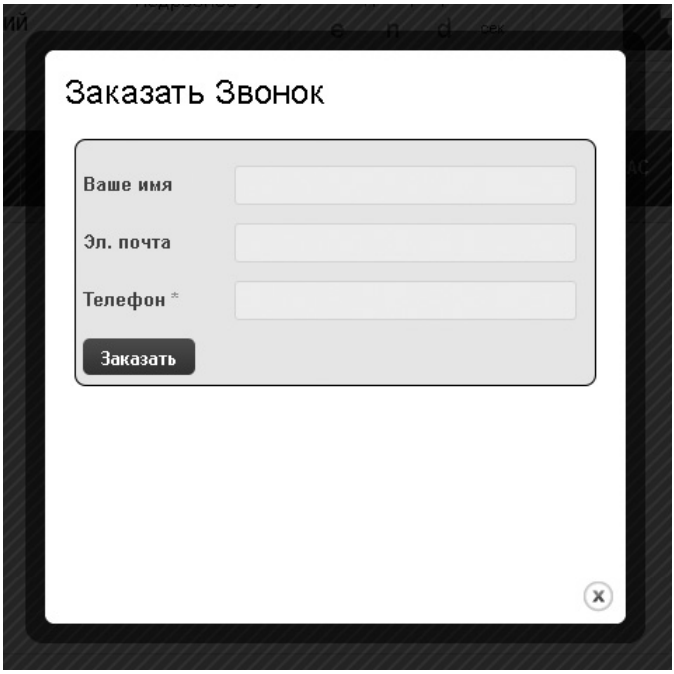


Рис. 5.10. Использование модуля Colorbox

Custom breadcrumbs (http://drupal.org/project/custom_breadcrumbs). Дает возможность добавлять в «хлебные крошки»[1] свои ссылки на разделы сайта.

Custom Search (http://drupal.org/project/custom_search). Содержит дополнительные настройки поискового механизма CMS Drupal, позволяющие скрывать отдельные части формы поиска, а также содержимого страницы результатов поиска.

Display Suite (http://drupal.org/project/ds). Позволяет изменять содержимое и расположение элементов на отдельных страницах сайта: аннотации, страницы публикации, страницы поиска и т. д.

Form Block (http://drupal.org/project/formblock). Дает возможность создать блок, содержащий форму добавления публикации определенного типа на сайт.

404 Navigation (http://drupal.org/project/navigation404). Устраняет ошибки отображения страницы «Страница не найдена», которые присутствуют в некоторых темах оформления.

---

[1] «Хлебные крошки» (от англ. Breadcrumbs) — элемент навигации по сайту, представляющий собой путь по сайту от его «корня» до текущей страницы, на которой находится пользователь.

Pathauto (http://drupal.org/project/pathauto). Позволяет автоматически по добавленному шаблону назначать синонимы адресам создаваемых публикаций, терминам таксономии, а также профилям пользователей.

Privatemsg (http://drupal.org/project/privatemsg). Добавляет на сайт механизм личных сообщений, то есть любой пользователь посредством специальной формы сможет написать другому пользователю сайта. Написанные сообщения хранятся и отображаются в профиле пользователя. Пользователь, которому было написано сообщение, сможет ответить на него также на сайте, создавая тем самым переписку.

Rules (http://drupal.org/project/rules). Расширяет и заменяет стандартные механизмы действий и триггеров CMS Drupal, то есть позволяет назначать события, которые будут автоматически происходить при наступлении определенного действия.

Quick Tabs (http://drupal.org/project/quicktabs). Дает возможность создавать вкладки, содержащие блоки информации (рис. 5.11). Навигация по таким вкладкам выполняется без перезагрузки страницы.

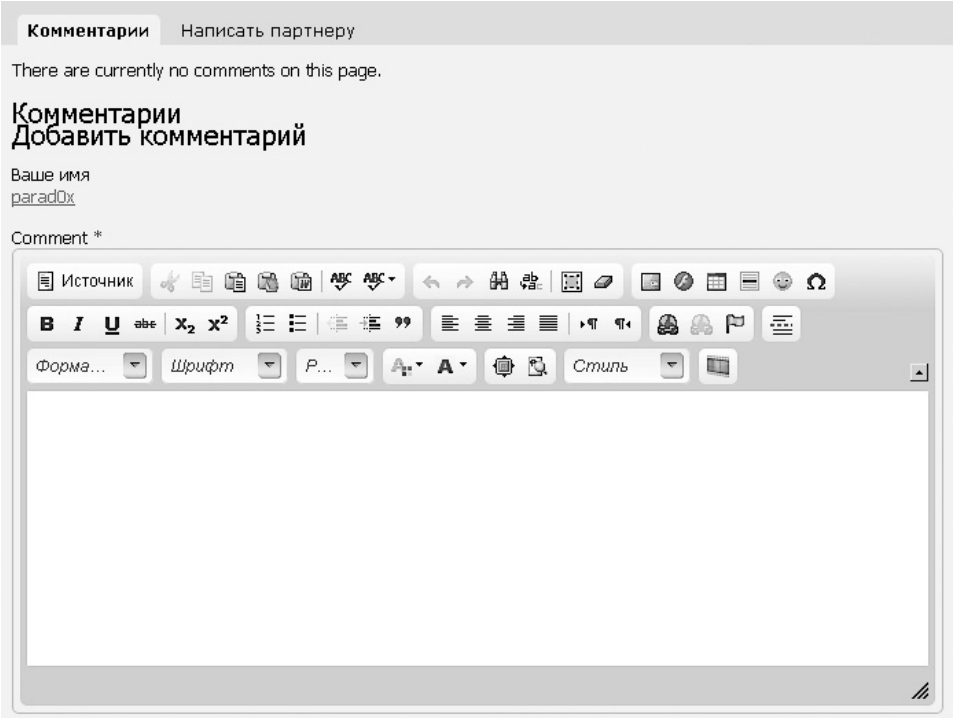


Рис. 5.11. Использование модуля Quick Tabs

Simplenews (http://drupal.org/project/simplenews). Позволяет вести почтовую рассылку, отправляя содержимое созданных публикаций на почту всем подписавшимся на рассылку пользователям.

Site map (http://drupal.org/project/site_map). Добавляет раздел с картой сайта (ссылками на все разделы сайта).

Superfish (http://drupal.org/project/superfish). Позволяет создавать раскрывающиеся меню (рис. 5.12).

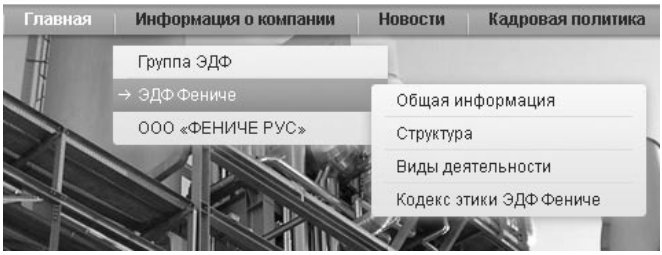


Рис. 5.12. Использование модуля Superfish

Tab Tamer (http://drupal.org/project/tabtamer). Дает возможность скрывать отдельные вкладки CMS Drupal, а также редактировать названия вкладок.

Talk (http://drupal.org/project/talk). Переносит комментарии со страницы публикации на отдельную вкладку.

Transliteration (http://drupal.org/project/transliteration). Автоматически переименовывает все добавляемые на сайт файлы, заменяя в названии файла русские символы на английские.

Views (http://drupal.org/project/views). Без данного модуля уже не обходится ни один сайт. Этот модуль позволяет создавать разделы сайта, содержащие список публикаций определенного типа, а также пейджер для навигации по страницам этого списка.

Views Autocomplete Filters (http://drupal.org/project/views_autocomplete_filters). Заменяет текстовые поля раскрытых фильтров модуля Views на автодополняемые, отображая список возможных значений при вводе содержимого в текстовое поле фильтра.

Views Slideshow (http://drupal.org/project/views_slideshow). Позволяет создавать слайд-шоу, отдельными слайдами которого будут позиции, выводимые модулем Views (рис. 5.13).

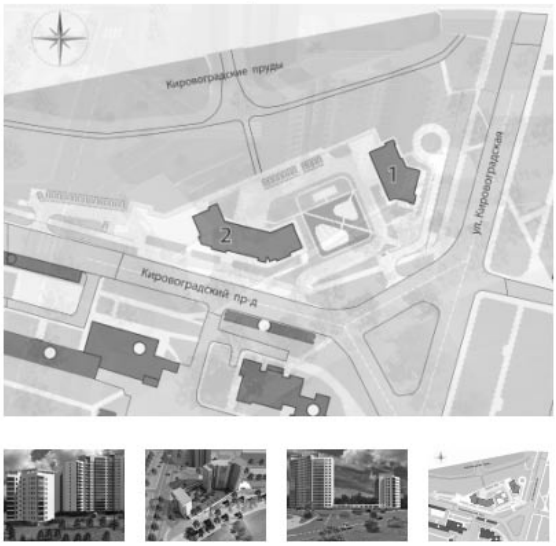


Рис. 5.13. Использование модуля Views Slideshow

Работа с контентом

Немаловажной частью любого сайта является контент. Собственно говоря, любой сайт состоит из контента. Поэтому следующий набор модулей, который мы рассмотрим, будет предназначен для добавления новых типов контента, а также расширения стандартного функционала.

Advertisement (http://drupal.org/project/ad). Добавляет возможность создания блоков текстовой и графической рекламы. При этом модуль ведет учет щелчков на рекламных ссылках.

Automatic Nodetitles (http://drupal.org/project/auto_nodetitle). Позволяет скрыть поле заголовка для определенного типа материалов либо сделать его необязательным для заполнения.

CAPTCHA (http://drupal.org/project/captcha). Дает возможность добавить математическую (ввод результата математического выражения) или графическую (ввод символов, изображенных на картинке) капчу на любую форму (рис. 5.14).

CAPTCHA Pack (http://drupal.org/project/captcha_pack). Расширяет возможности модуля CAPTCHA, добавляя множество дополнительных видов капчи (рис. 5.15).

Fivestar (http://drupal.org/project/fivestar). Позволяет добавлять публикациям, профилям пользователей или терминам таксономии рейтинг (рис. 5.16).

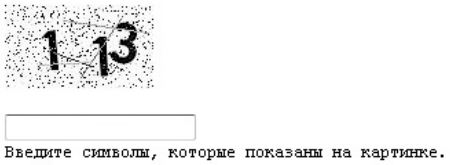


Рис. 5.14. Использование модуля CAPTCHA

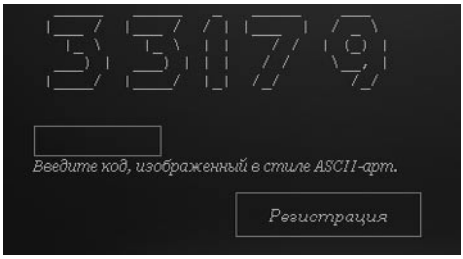


Рис. 5.15. Использование модуля CAPTCHA Pack

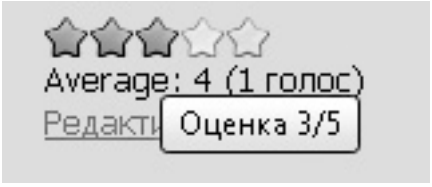


Рис. 5.16. Использование модуля Fivestar

Flag (http://drupal.org/project/flag). Дает возможность добавлять публикации в избранное — специальный раздел сайта, где отображаются только те публикации, которые данный посетитель добавил в избранное.

Node clone (http://drupal.org/project/node_clone). Добавляет возможность создания новых публикаций путем клонирования существующих.

Node and Comments Form Settings (http://drupal.org/project/nodeformsettings). Расширяет список настроек типов материалов и формы комментирования. В основном новые настройки предназначены для скрытия различных элементов формы, предназначенной для добавления публикации/комментария.

phpBB Forum Integration (http://drupal.org/project/phpbbforum). Интегрирует в Drupal популярный форум phpBB.

SimpleAds (http://drupal.org/project/simpleads). Добавляет возможность создания блоков текстовой и графической рекламы в SWF-формате. При этом модуль ведет учет щелчков на рекламных ссылках.

Sort Comments (http://drupal.org/project/sort_comments). Добавляет возможность сортировать комментарии по возрастанию или убыванию.

Webform (http://drupal.org/project/webform). Позволяет создавать различные формы для ввода какой-либо информации. Результаты заполнения форм хранятся на сайте, а также могут быть автоматически отправлены на указанный электронный адрес.

ImageCache Actions (http://drupal.org/project/imagecache_actions). Расширяет возможности видов показа изображений, позволяет автоматически добавлять изображениям водяной знак, фон, а также применять множество других эффектов.

Все для CKEditor

Крайне полезным при создании публикаций на сайте является текстовый редактор с поддержкой технологии WYSIWYG[1], позволяющий создавать блоки текста практически с любым форматированием вообще без знания HTML. В своих проектах я использую текстовый редактор CKEditor. Поэтому ниже будет рассмотрен именно он, а также модули, расширяющие его возможности.

CKEditor — WYSIWYG HTML editor (http://drupal.org/project/ckeditor). Заменяет текстовые области на сайте текстовым редактором CKEditor. Данный редактор облегчает создание форматированного текста, позволяя создавать полноценные публикации вообще без знания HTML (рис. 5.17).

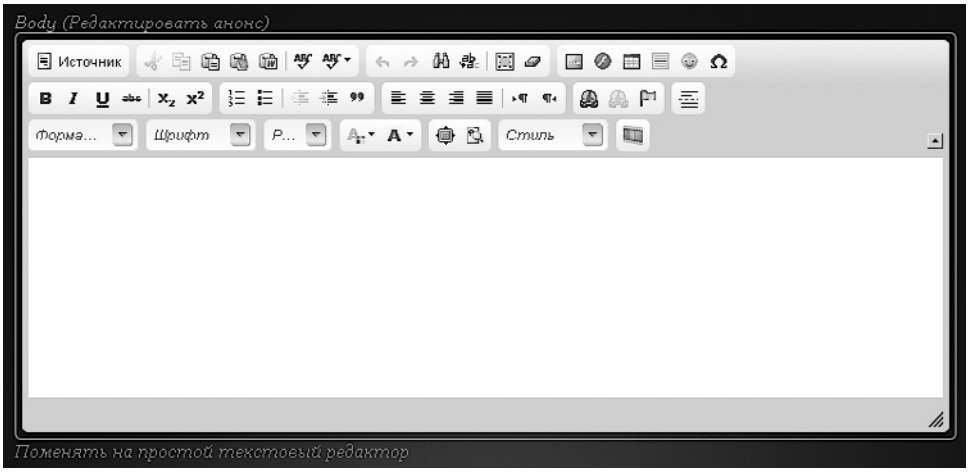


Рис. 5.17. Использование модуля CKEditor

[1] WYSIWYG (от англ. What You See Is What You Get — «Что видишь, то и получишь») — содержимое отображается в процессе редактирования максимально близко к тому, как оно будет выглядеть на веб-странице.

Image Resize Filter (http://drupal.org/project/image_resize_filter). Уменьшает изображения, добавленные через текстовый редактор, до указанных размеров и добавляет ссылку, при щелчке на которой открывается изображение в оригинальном размере.

IMCE (http://drupal.org/project/imce). Интегрирует файловый менеджер в текстовый редактор. Благодаря файловому менеджеру вы легко сможете вставлять изображения прямо в тело публикации, не размещая их предварительно в Интернете (изображение копируется на сайт непосредственно при добавлении его в публикацию).

IMCE Crop (http://drupal.org/project/imce_crop). Расширяет возможности файлового менеджера IMCE, добавляя функции уменьшения добавляемых изображений до указанной ширины и высоты.

IMCE Mkdir (http://drupal.org/project/imce_mkdir). Расширяет возможности файлового менеджера IMCE, позволяя с его помощью создавать каталоги на вашем сайте.

Video Filter (http://drupal.org/project/video_filter). Позволяет добавлять в тело публикации ролики из YouTube и других видеохостингов (рис. 5.18). Достаточно просто указать URL к ролику, а плеер для воспроизведения ролика добавит данный модуль.

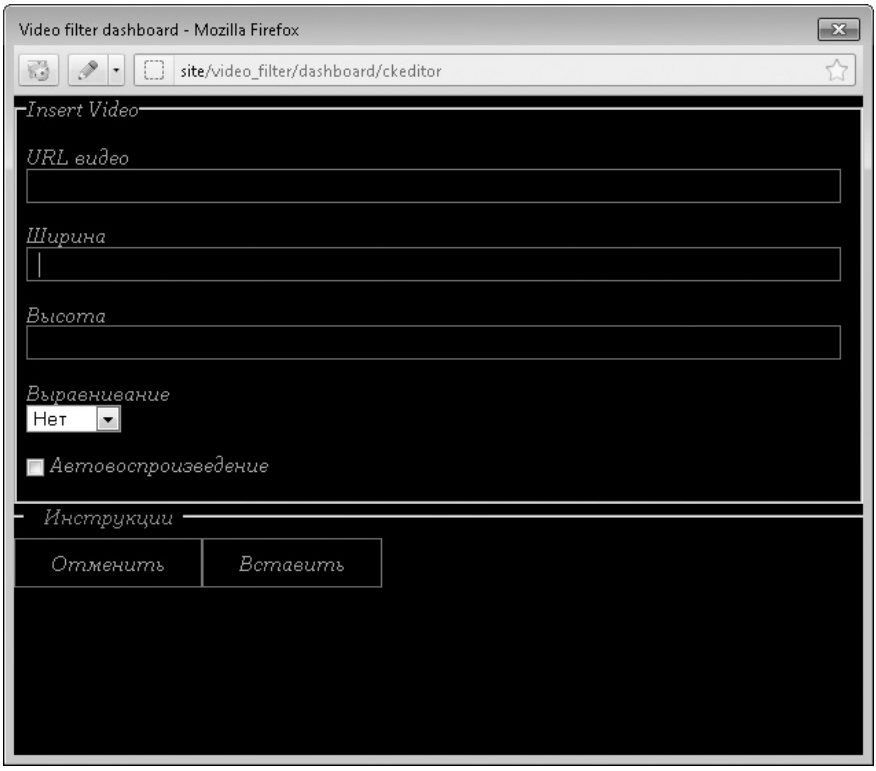


Рис. 5.18. Использование модуля Video Filter

Поля

Практически ни один тип материалов в CMS Drupal не обходится без дополнительных полей. Хотя бы одно поле — поле для добавления к публикации картинки, — но добавляется к каждому типу материалов.

Ниже будут рассмотрены модули, расширяющие возможности полей в CMS Drupal.

Conditional Fields (http://drupal.org/project/conditional_fields). Позволяет создавать зависимые поля, то есть поля, которые будут отображаться только в том случае, если пользователь указал конкретную информацию в определенном поле (например, установил или снял определенный флажок) (рис. 5.19).

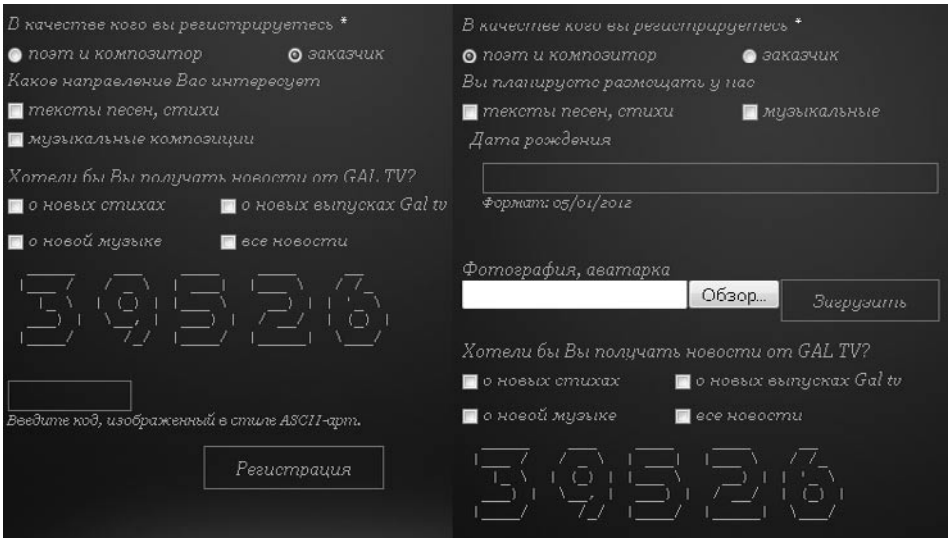


Рис. 5.19. Использование модуля Conditional Fields

Date (http://drupal.org/project/date). Добавляет поле типа Дата, позволяющее вводить дату либо выбирать ее из всплывающего календаря или раскрывающихся списков с годом, месяцем, днем и т. п.

Field group (http://drupal.org/project/field_group). Добавляет возможность группировки полей. Группе полей можно присвоить название, а также определить способ вывода данной группы: либо простая группировка, либо группа полей с возможностью сворачивать/разворачивать группу.

В CMS Drupal 6 функционал полей добавлялся сторонним модулем CCK, и в состав этого модуля входил модуль для группирования полей. В Drupal 7 основной функ-

ционал модуля CCK был встроен в ядро. Однако возможности группировки полей в ядро добавлены не были, и для реализации данной функции необходимо устанавливать дополнительный модуль.

References (http://drupal.org/project/references). Добавляет поля типа Ссылка на материал и Ссылка на пользователя.

В CMS Drupal 6 функционал данного модуля входил в состав модуля CCK. В Drupal 7 для реализации данного функционала необходимо устанавливать дополнительный модуль.

Taxonomy Term Reference Tree Widget (http://drupal.org/project/term_reference_tree). Расширяет возможности вывода полей с терминами таксономии. В частности, позволяет запретить возможность выбора родительских терминов таксономии в списке (рис. 5.20).

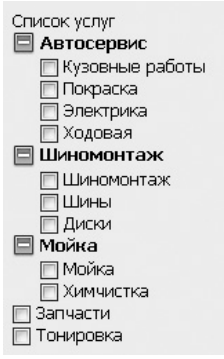


Рис. 5.20. Использование модуля Taxonomy Term Reference Tree Widget

SEO

Вопросы SEO (раскрутки сайта, повышения его посещаемости) в современном мире находятся далеко не на последнем месте. Поэтому забывать о модулях, предназначенных для решения SEO-задач, не стоит.

Global Redirect (http://drupal.org/project/globalredirect). Позволяет настраивать редиректы с одних страниц сайта на другие.

Meta tags (http://drupal.org/project/metatag). Добавляет возможность ввода метатегов (ключевых слов, описания, а также заголовка страницы) для создаваемых на сайте публикаций.

Meta tags quick (http://drupal.org/project/metatags_quick). Добавляет возможность ввода метатегов (ключевых слов и описания) для создаваемых на сайте публикаций, а также для любой другой страницы сайта.

Page Title (http://drupal.org/project/page_title). Добавляет возможность ввода собственного заголовка (содержимого тега `title` страницы) для публикаций, представлений, терминов таксономии.

XML sitemap (http://drupal.org/project/xmlsitemap). Позволяет настроить автоматическое создание файла `sitemap.xml`.

Авторизация

Еще один набор модулей, которые мы рассмотрим, используется на этапе регистрации и авторизации посетителя на вашем сайте.

Legal (http://drupal.org/project/legal). Добавляет на страницу регистрации нового пользователя флажок Я согласен с условиями работы сайта (рис. 5.21).

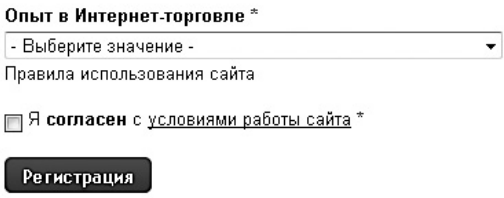


Рис. 5.21. Использование модуля Legal

LoginToboggan (http://drupal.org/project/logintoboggan). Расширяет возможности механизма регистрации и аутентификации на сайте. В частности, позволяет использовать для входа на сайт указанный адрес электронной почты, а не свой логин.

uLogin (advanced version) (http://drupal.org/project/ulogin). Добавляет возможность входа и регистрации на сайте с помощью учетных записей, созданных на других популярных сайтах (в основном в социальных сетях) (рис. 5.22).

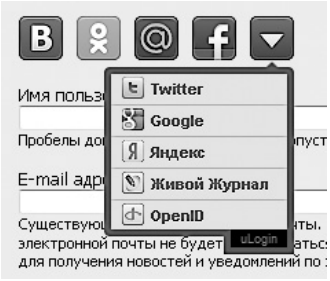


Рис. 5.22. Вход/регистрация через социальные сети

Социальные медиа

Несколько рассмотренных ниже модулей позволят расширить социальные возможности вашего сайта.

Forward (http://drupal.org/project/forward). Добавляет ссылку, с помощью которой посетитель сможет отправить по электронной почте своему другу URL текущей страницы вашего сайта (рис. 5.23).

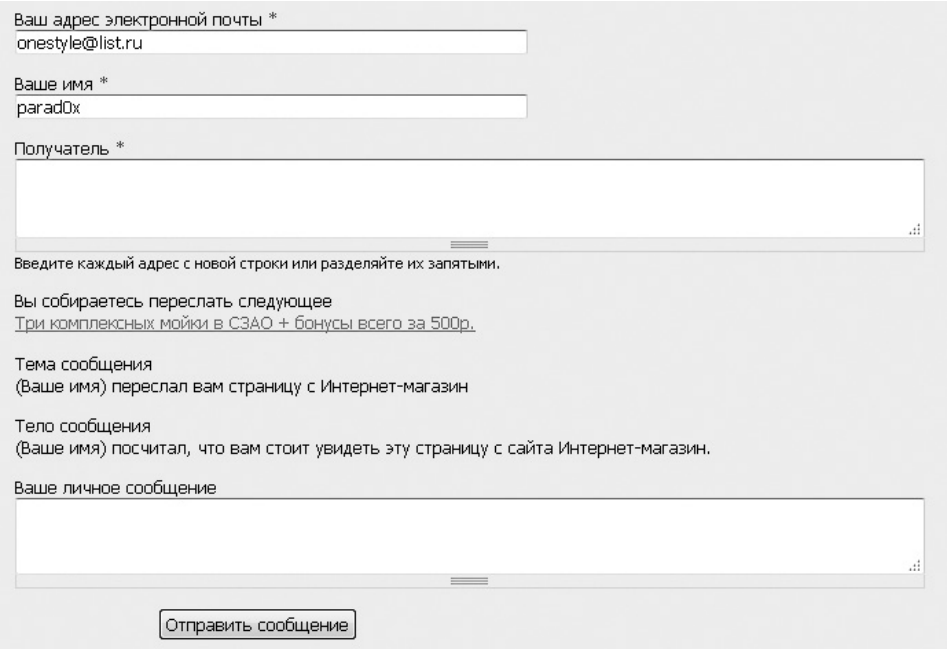


Рис. 5.23. Использование модуля Forward

Printer, email and PDF versions (http://drupal.org/project/print). Добавляет к публикациям версию для печати, а также ссылку на автоматически создаваемые PDF-версии публикаций.

Service links (http://drupal.org/project/service_links). Добавляет ссылки, с помощью которых посетитель сможет добавить текущий URL в какой-либо сервис социальных закладок (рис. 5.24).

Производительность

О производительности сайта никогда не стоит забывать. Если есть возможность снизить нагрузку на сервер, то ею обязательно нужно воспользоваться.

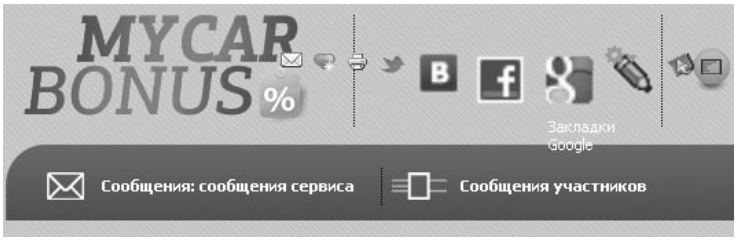


Рис. 5.24. Использование модуля Service links

Authenticated User Page Caching (Authcache) (http://drupal.org/project/authcache). Заменяет стандартный механизм кэширования Drupal более производительным.

CSS Embedded Images (http://drupal.org/project/css_emimage). Встраивает небольшие изображения, ссылки на которые указаны в CSS-файле, непосредственно в файл CSS. При этом изображения загружаются вместе с CSS-файлом и их не приходится загружать отдельно.

Entity cache (http://drupal.org/project/entitycache). Добавляет функционал кэширования типам публикаций и пользователей. Кэширование выполняется автоматически, и после включения модуля никаких дополнительных действий не требуется.

File Cache (http://drupal.org/project/filecache). Заменяет стандартный механизм кэширования в базе данных более производительным файловым кэшем.

Memcache API and Integration (http://drupal.org/project/memcache). Заменяет стандартный механизм кэширования Drupal более производительным.

Мультиязычность

Отдельный класс модулей предназначен для создания мультиязычного сайта, то есть сайта на нескольких языках, в котором посетитель сам выбирает, на каком языке ему будет предоставляться информация.

Internationalization (http://drupal.org/project/i18n). Основной модуль, предназначенный для создания мультиязычных сайтов.

Internationalization Views (http://drupal.org/project/i18nviews). Расширяет возможности модуля Internationalization, позволяя переводить заголовки представлений, созданных модулем Views.

Language icons (http://drupal.org/project/languageicons). Добавляет на сайт блок со значками языков, с помощью которого можно легко переключаться между языками вашего мультиязычного сайта (рис. 5.25).

Рис. 5.25. Использование модуля Languageicons

Интернет-магазин

И последний набор модулей, которые мы рассмотрим, предназначен для создания интернет-магазинов. Основным из перечисленных модулей является модуль интернет-магазина Ubercart. Остальные модули расширяют его функционал.

Ubercart (http://drupal.org/project/ubercart). Основной модуль, позволяющий создавать интернет-магазины на CMS Drupal.

Ubercart AJAX Cart (http://drupal.org/project/uc_ajax_cart). Заменяет стандартную корзину модуля Ubercart на более удобную AJAX-корзину, то есть добавляемая данным модулем корзина обновляется без перезагрузки страницы. Все операции по работе с корзиной происходят без перезагрузки страницы: добавление товара в корзину, удаление товара из корзины, изменение количества товаров.

Uc_onpay (http://onpay.ru/drupal). Добавляет возможность оплаты купленных товаров с помощью электронных денег: Webmoney, «Яндекс.Деньги», кредитной карты, СМС-сервисов и т. д.

Ubercart Product Power Tools (http://drupal.org/project/uc_product_power_tools). Данный модуль добавляет дополнительные возможности настройки формы создания товара. Как правило, данные настройки позволяют скрыть отдельные элементы формы создания товара.

Обновляем модули

Немаловажной частью администрирования CMS Drupal является обновление сторонних модулей, которые вы используете на своем сайте. Ведь модули постоянно дорабатываются. В них вносится новый функционал, а также исправляются найденные ошибки. И, что самое главное, исправляются найденные в модулях «дыры» в безопасности.

Но как правильно обновить модули, чтобы наименьшим образом повлиять на работу сайта? Сейчас мы попробуем этому научиться.

1. В первую очередь нам необходимо определить список модулей, для которых имеются обновления. Для этого откройте страницу <сайт>/admin/reports/updates/update (Отчеты ▸ Доступные обновления ▸ Обновить). После чего нажмите ссылку

Проверить вручную (информация, которая отображается на данной странице, может быть неактуальной, поэтому нажатие данной ссылки является обязательным).

2. Начнется процесс поиска обновлений, который закончится через некоторое время списком ссылок на модули, для которых обнаружены обновления (рис. 5.26). Перейдите по ссылкам в данном списке и скачайте последние версии модулей.

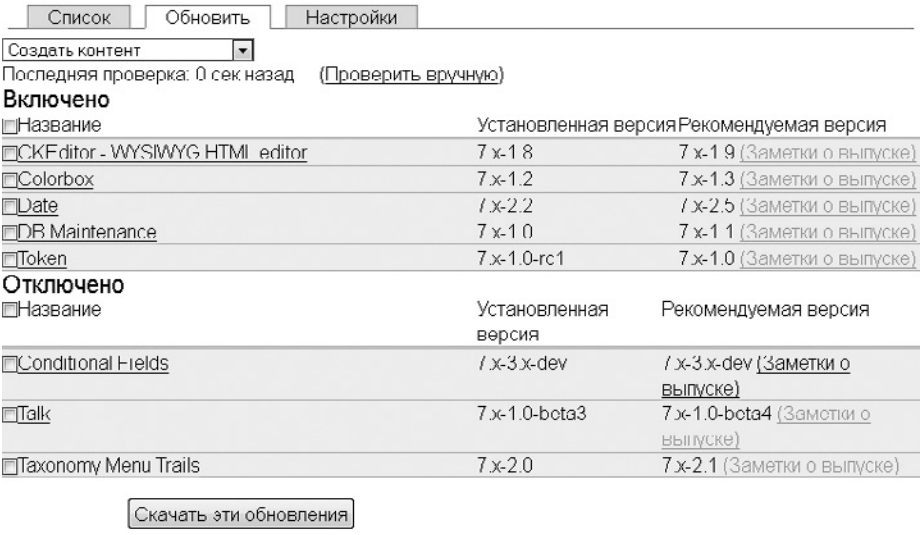


Рис. 5.26. Список найденных обновлений

3. Новые версии модулей мы получили. Теперь подумаем о безопасности нашего сайта. В редких случаях процесс обновления модулей чреват различными неприятностями. Например, потерей части функционала, которая была реализована на сайте. Поэтому перед тем, как приступать к обновлению модулей, необходимо обязательно сделать архив базы данных сайта.

   Откройте страницу <сайт>/admin/config/system/backup_migrate (Сайт ▸ Система ▸ Резервное копирование и миграция). Поскольку модуль резервного копирования мы настроили ранее, сейчас достаточно просто нажать кнопку Копировать сейчас. После этого файл резервной копии будет загружен на ваш компьютер.

4. Переходим к архивам с новыми версиями модулей, которые мы скачали на шаге 2. Распакуйте их в текущую папку (именно в текущую, а не в отдельную папку (рис. 5.27)), после чего откройте FTP вашего сайта и перейдите в папку sites/all. Создайте в ней папку `new` и скопируйте в папку `new` папки новых версий модулей, которые были получены после распаковки архивов.

5. Дождитесь завершения копирования. После чего можно приступать к замене старых версий модулей на новые. Для этого:

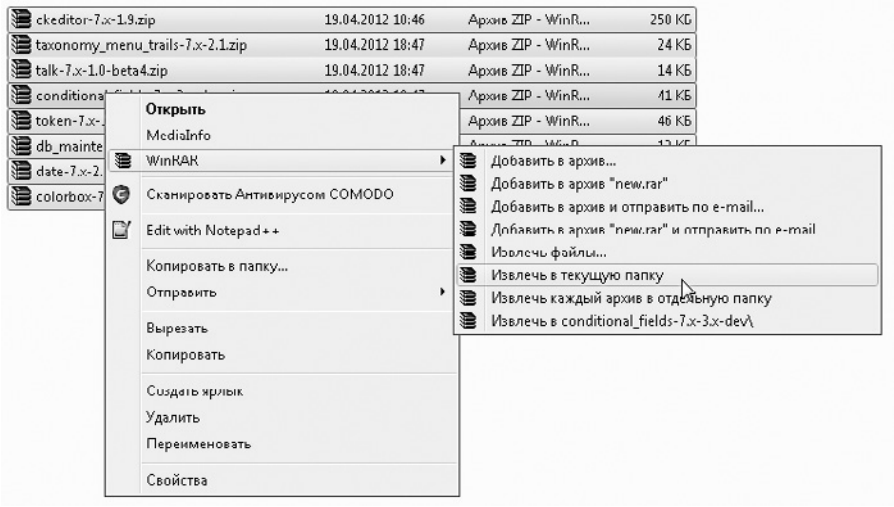


Рис. 5.27. Извлекаем архивы в текущую папку

1) переместите папку со старой версией модуля из папки sites/all/modules в папку sites/all;
2) переместите папку с новой версией данного модуля из папки sites/all/new в папку sites/all/modules.

Ни в коем случае не удаляйте папку со старой версией модуля. Возможно, она вам еще пригодится, если после обновления будут обнаружены проблемы в работе сайта.

Папку со старой версией модуля обязательно необходимо перемещать. Простое переименование папки не поможет, поскольку CMS Drupal определяет модули не по названию папки модуля, а по названию и содержимому файлов, которые в ней находятся.

6. Вы заменили все модули новыми версиями. Теперь необходимо сделать еще один обязательный шаг — обновить структуру базы данных сайта, чтобы внести в нее все возникшие в новых версиях модулей изменения. Для этого откройте страницу <сайт>/update.php, после чего нажмите кнопку Continue (Продолжить). Далее возможны два варианта действий:

   - перед вами отобразилась страница с надписью Нет ожидающих обновлений — значит, вам повезло и новые версии модулей никак не изменяют структуру базы данных; в этом случае нажмите ссылку Administration pages (Страницы администрирования), чтобы просто перейти на страницу администрирования сайта;
   - перед вами отобразилась страница со списком обновлений структуры базы данных, которые будут выполнены, — запустите процесс обновления и дождитесь его результатов.

7. Теперь процесс обновления можно считать завершенным. Вам осталось лишь проверить на работоспособность весь функционал сайта, который был реализован с помощью обновленных модулей. Можно также просмотреть страницы настройки обновленных модулей — вдруг там появились новые возможности.

Основные понятия

Чтобы лучше понимать возможности CMS Drupal, достаточно изучить ее структуру — логические блоки, из которых она состоит. После этого знакомиться с возможностями CMS Drupal будет гораздо проще.

Тема оформления

То, как выглядит сайт на CMS Drupal, зависит от темы оформления, которая в данный момент используется. Тема оформления — это фактически дизайн и верстка вашего сайта. Тема оформления также определяет расположение элементов на веб-странице. А некоторые темы оформления еще и добавляют новые возможности: слайд-шоу на главной странице, реализацию раскрывающегося меню и т. д.

Управление темами оформления. Список тем оформления, которые можно использовать, находится на странице <drupal>/admin/appearance (Вид) (рис. 5.28).

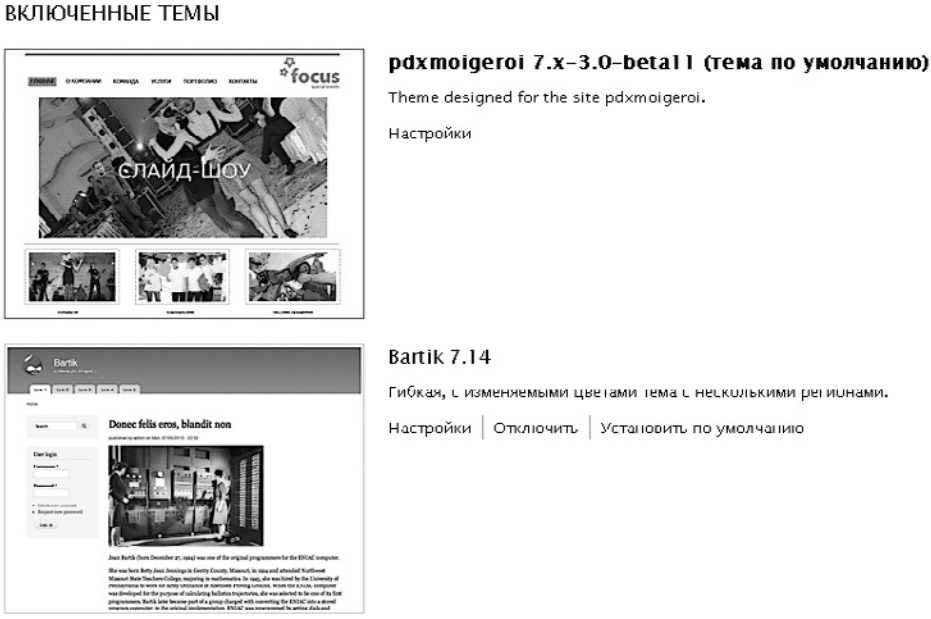


Рис. 5.28. Список тем оформления

Темы оформления в CMS Drupal могут иметь три состояния:

- отключенная — тема оформления, которую нельзя использовать (до тех пор пока администратор ее не включит);
- включенная — если у пользователя есть соответствующие разрешения, он может установить любую тему оформления из списка включенных в качестве своей темы (для своей учетной записи);
- тема оформления по умолчанию — включенная тема оформления, которая в данный момент используется на сайте.

Работать с темами оформления несложно. Если вы хотите поменять тему оформления, которая в данный момент используется на сайте, достаточно нажать ссылку Установить по умолчанию (для включенных тем) или Включить и установить по умолчанию (для отключенных тем) напротив нужной темы оформления.

Стандартные темы оформления. По умолчанию в CMS Drupal имеется четыре темы оформления: Bartik, Garland, Seven и Stark (рис. 5.29).

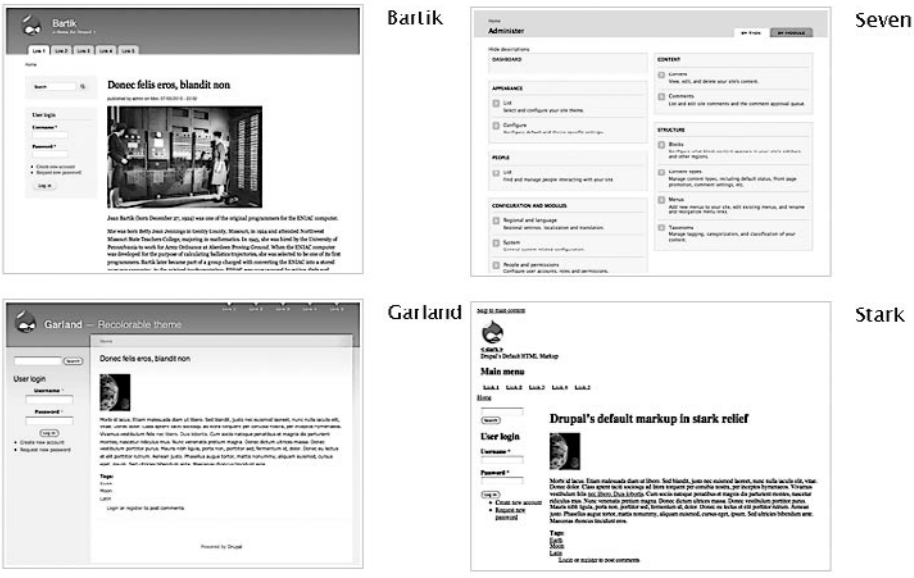


Рис. 5.29. Стандартные темы оформления

Вы можете использовать одну из них либо установить сторонние темы оформления.

Сторонние темы оформления. Сотни бесплатных тем оформления можно найти на странице http://drupal.org/project/themes. Откройте эту страницу, после чего в списке Filter by compatibility (Сортировать по совместимости) выберите элемент 7.x.

Чтобы установить стороннюю тему оформления, сделайте следующее.

1. Скачайте архив с темой оформления.
2. Распакуйте архив в текущую папку (у вас появится папка с названием, аналогичным названию темы оформления).
3. Убедитесь, что внутри папки с темой оформления находятся файлы данной темы, а не еще одна папка (распаковывать архив с темой оформления нужно именно в текущую папку, а не в папку с именем архива).
4. Откройте FTP вашего сайта и перейдите в каталог sites/all.
5. Если в данном каталоге нет папки themes, создайте ее, после чего поместите папку с темой оформления в каталог sites/all/themes.
6. Откройте страницу <drupal>/admin/appearance вашего сайта и включите новую тему оформления.

Настройки темы оформления. В CMS Drupal имеется два вида настроек темы оформления: общие настройки и настройки конкретной темы оформления.

Общие настройки для всех тем оформления находятся на странице <drupal>/admin/appearance/settings. С их помощью вы можете изменить логотип и favicon[1] вашего сайта, а также включить или отключить отображение логотипа, названия сайта, слогана, аватарок пользователей, значков, а также меню сайта.

Как правило, настройки конкретной темы оформления заменяют собой общие настройки, то есть со страницы настройки конкретной темы оформления можно изменить точно то же, что и со страницы с общими настройками. А кроме того, каждая тема оформления имеет собственные дополнительные настройки.

Блоки

Блоком называется отдельная часть веб-страницы, в которую вы можете поместить любой HTML-текст либо PHP-код. Блоки создаются на странице <drupal>/admin/structure/block/add (Структура ▸ Блоки ▸ Добавить блок). При создании блока необходимо указать:

- административный заголовок блока (виден только администраторам, идентифицирует блок в списке всех блоков);
- заголовок блока, который будет отображаться на веб-странице над содержимым блока (это можно задать при желании);

[1] Значок сайта, который отображается в адресной строке браузера слева от URL открытой в данный момент страницы.

- содержимое блока и используемый для него формат ввода (какие теги разрешены в содержимом и разрешено ли использовать PHP);
- условия, при которых блок будет отображаться на веб-странице:
  - страницы сайта, на которых он будет показываться;
  - роли пользователей, для которых он будет отображаться;
  - типы материалов, на страницах которых он будет выводиться;
- регион темы оформления, в котором будет отображаться блок.

Просто создать блок недостаточно. Чтобы блок появился на сайте, его нужно разместить в одном из регионов. Список регионов, их расположение и названия зависят от текущей темы оформления.

Список всех блоков на сайте можно посмотреть на странице <drupal>/admin/appearance/settings (рис. 5.30). Блоки в этом списке сгруппированы по регионам, в которых они расположены. Отключенные блоки (те, которые не расположены в каком-либо регионе темы оформления) находятся в самом конце списка.

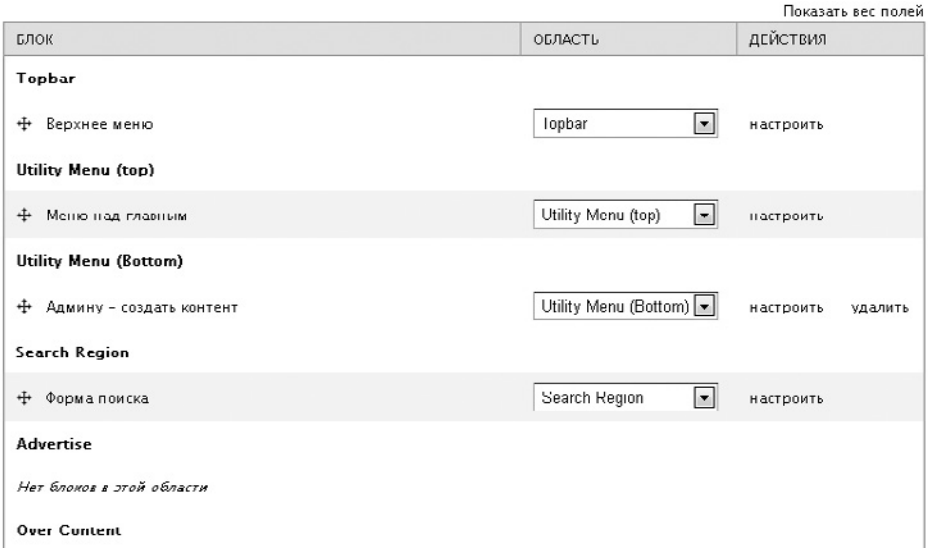


Рис. 5.30. Список блоков

Меню

Меню представляет собой плоский или иерархический набор ссылок на страницы вашего сайта. Под иерархическим списком понимаются раскрывающиеся меню, то есть когда пункты меню вложены друг в друга.

В CMS Drupal можно создать любое количество меню, после чего разместить их в любом из регионов текущей темы оформления.

Для работы с меню используется страница <drupal>/admin/structure/menu (Структура ▸ Меню). На ней находится список стандартных и созданных вами меню. По умолчанию в CMS Drupal имеются следующие меню:

- Main menu (Главное меню) — главное меню сайта; по умолчанию не содержит ни одного пункта;
- Management (Управление) — административное меню, включающее в себя ссылки на все страницы административного раздела CMS; данное меню отображается только для администраторов; в CMS Drupal 7 данное меню потеряло свою актуальность — его отлично заменяют стандартный модуль Toolbar либо сторонний модуль Admin menu;
- Navigation (Навигация) — различные модули добавляют в данное меню ссылки на второстепенные разделы сайта; в частности, в данном меню находятся пункты для создания контента, а также ссылка на корзину (для интернет-магазина) и страницу поиска;
- User menu (Пользовательское меню) — пункты меню, связанные с учетной записью: ссылки для входа на сайт под своей учетной записью и выхода, а также ссылка на профиль вошедшего пользователя.

Чтобы создать новый пункт меню, достаточно воспользоваться ссылкой Добавить ссылку напротив нужного меню на странице <drupal>/admin/structure/menu.

Если вы хотите изменить расположение пунктов меню, воспользуйтесь ссылкой список ссылок на странице <drupal>/admin/structure/menu, после чего перетащите пункты меню в нужное место операцией Drag and drop (перетаскивать необходимо не сам пункт меню, а расположенное слева от него изображение со стрелочками, направленными в четыре стороны (рис. 5.31)).

По умолчанию ни одно из стандартных или созданных вами меню не отображается на сайте (за исключением Main menu (Главное меню) — в некоторых темах оформления это меню выводится непосредственно через шаблон страницы). Чтобы отобразить их, необходимо открыть страницу со списком блоков (<drupal>/admin/structure/block), после чего перенести блок, административное название которого совпадает с названием меню, в один из регионов вашей темы оформления.

Форматы ввода

Мы уже встречались с форматами ввода во время создания блоков. Формат ввода — это в первую очередь возможность ограничить список тегов, которые пользо-

ватель может применять в добавляемом на сайт тексте. Хотя, помимо функции ограничения, у форматов ввода есть и другие возможности.

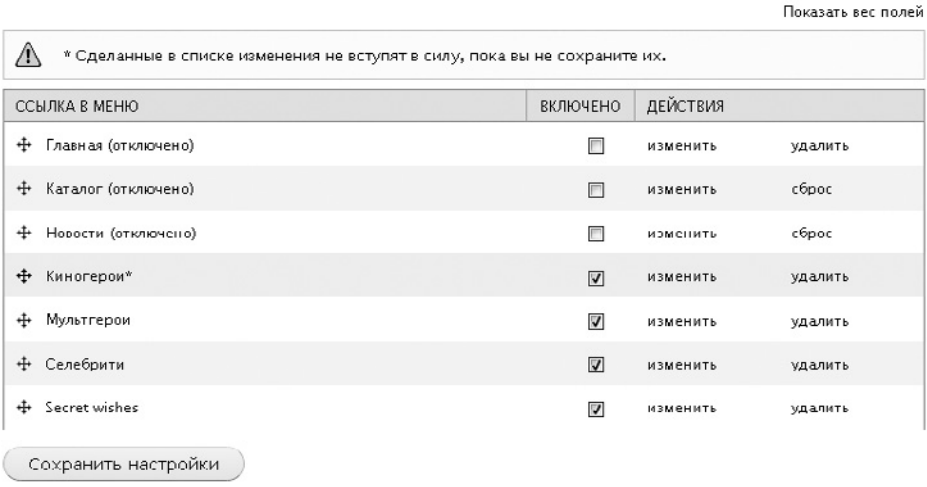


Рис. 5.31. Список пунктов меню

Список существующих форматов ввода находится на странице <drupal>/admin/config/content/formats (Сайт ▸ Автор контента ▸ Форматы текста). По умолчанию существуют следующие форматы ввода:

- Full HTML (Полный HTML) — по умолчанию доступен только администраторам и позволяет использовать в тексте любые теги HTML; помимо этого, в данном формате ввода URL-адреса автоматически преобразуются в ссылки, переводы строк заменяются тегами `br`, а также автоматически исправляются незакрытые и ошибочные теги;
- Plain text (Простой текст) — по умолчанию доступен всем пользователям; он не позволяет пользователям применять в тексте теги, но при этом URL-адреса автоматически преобразуются в ссылки, а переводы строк заменяются тегами `br`;
- PHP code (PHP-код) — данный формат ввода добавляется только после включения стандартного модуля PHP Code и доступен только администратору; он позволяет использовать в тексте PHP-код;
- Display Suite code (Код Display Suite) — данный формат ввода добавляется только после включения стороннего модуля Display Suite.

На самом деле Full HTML (Полный HTML), Plain text (Простой текст) и т. д. — это просто названия форматов ввода. Действие формата ввода зависит не от его названия, а от фильтров, которые для него включены. Чтобы посмотреть включенные фильтры, а также включить новые или отключить существующие, достаточно нажать

ссылку настроить напротив нужного формата ввода на странице <drupal>/admin/config/content/formats.

На странице настройки формата ввода можно также указать роли пользователей, которым будет доступен данный формат ввода. О ролях мы поговорим в следующем подразделе.

Роли и разрешения

В CMS Drupal каждому зарегистрированному на сайте пользователю может быть присвоено любое количество ролей. Сами по себе роли значат немного, но их можно применять, чтобы разрешить или запретить выполнение каких-либо операций на сайте.

Роли. Список созданных на сайте ролей находится на странице <drupal>/admin/people/permissions/roles (Люди ▸ Разрешения ролей ▸ Роли). По умолчанию в CMS Drupal существуют следующие роли:

- Анонимный пользователь — системная роль (ее нельзя удалить), которая автоматически назначается всем посетителям сайта, еще не авторизовавшимся на сайте под своей учетной записью;
- Зарегистрированный пользователь — системная роль, которая автоматически назначается всем посетителям сайт, вошедшим под своей учетной записью;
- Administrator — создается по умолчанию, но не является системной. Ее можно удалить, в отличие от предыдущих.

Вы можете добавить новые роли на странице <drupal>/admin/people/permissions/roles. Для этого используется кнопка Добавить роль.

Если вы хотите удалить роль, сначала необходимо нажать ссылку изменить роль на странице <drupal>/admin/people/permissions/roles. После чего на открывшейся странице нажать кнопку Удалить.

А вот удаление и присвоение ролей пользователям выполняется на странице <drupal>/admin/people (Люди). Установите флажок напротив нужного вам пользователя, после чего в раскрывающемся списке Обновить параметр выберите необходимую роль в подэлементе Добавление роли выбранным пользователям (рис. 5.32). После этого не забудьте нажать кнопку Обновить.

Разрешения ролей. По умолчанию новые роли ничего не значат — для них не установлено ни одного дополнительного разрешения. И возможности зарегистрировавшегося пользователя определяются разрешениями роли Зарегистрированный пользователь.

Рис. 5.32. Список пользователей

Чтобы установить или отменить разрешения любой созданной роли, используется страница <drupal>/admin/people/permissions (Люди ▸ Разрешения ролей). Ничего сложного в этом нет — чтобы добавить нужное разрешение, достаточно просто установить флажок напротив нужной роли (рис. 5.33).

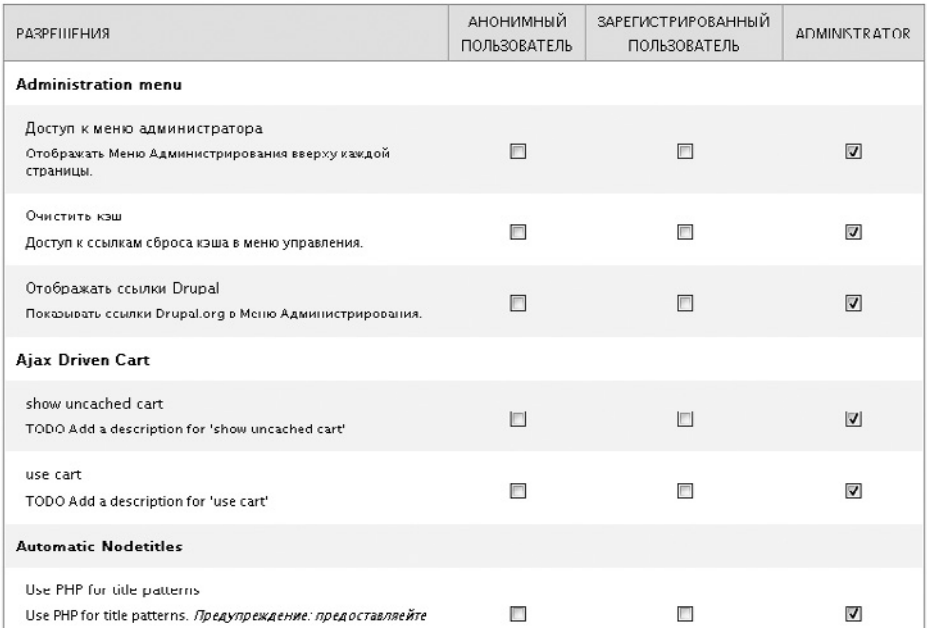


Рис. 5.33. Список разрешений

Типы материалов

Любой сайт представляет собой набор страниц.

Среди этих страниц можно найти такие, как О компании, Правила сайта, Наши достижения и т. д., то есть обычные разделы сайта, содержащие в себе какую-либо текстовую информацию.

Кроме того, на многих сайтах есть статьи, новости и другие структурированные типы информации. В основном такие страницы создаются по определенному шаблону, который практически никогда не меняется. Возьмем, например, новости. Каждая новость состоит из заголовка, даты написания, источника, аннотации, содержимого новости и тематической картинки. И, как правило, при добавлении новости стараются внести все перечисленные данные.

Типы материалов. В CMS Drupal каждая страница сайта создается по определенному шаблону — так называемому типу материалов. Изначально существует два типа материалов:

- basic page — обычная страница сайта, для которой отсутствует возможность добавления комментариев и не отображается дата создания и автор материала;
- story — шаблон для создания статей на сайте; изначально для каждой статьи отображается форма комментирования, а также дата написания статьи и ее автор.

Вы можете создавать любое количество типов материалов. Для этого предназначена страница сайта <drupal>/admin/structure/types/add (Структура ▸ Типы содержимого ▸ Добавить тип содержимого). На ней вы сможете:

- изменить название поля для ввода заголовка публикации, а также ввести инструкцию, которая будет отображаться при создании публикаций данного типа;
- отключить возможность предпросмотра публикации перед ее добавлением либо сделать предпросмотр обязательным;
- определить, будет ли созданный материал опубликован по умолчанию, либо администратор должен будет выполнять модерацию каждого созданного материала;
- включить или отключить отображение даты создания публикации, а также ее автора;
- отключить либо настроить форму добавления комментариев к публикациям данного типа;
- определить меню сайта, в которые при создании публикации можно будет добавлять пункты со ссылкой на публикацию и т. д.

Перечисленные настройки можно как указывать при создании нового типа материалов, так и изменять для уже существующих типов материалов.

Список созданных на сайте типов материалов доступен на странице <drupal>/admin/structure/types (Структура ▸ Типы содержимого). На этой же странице вы сможете отредактировать либо удалить любой из созданных типов материалов.

Настройки, которые вы указали для типа материалов, будут применяться по умолчанию для каждой создаваемой публикации данного типа. Однако при создании публикации вы всегда можете изменить их для данного конкретного случая.

Создание публикаций. Процесс создания публикаций в CMS Drupal начинается с выбора типа материалов, на основе которого вы хотите создать публикацию. Это делается на странице <drupal>/node/add (Контент ▸ Добавить содержимое).

После того как тип материалов выбран, остается лишь заполнить поля заголовка и содержимого материала и при необходимости изменить настройки по умолчанию, которые вы указывали при создании типа материалов, на собственные.

Список публикаций. Все публикации, созданные на сайте, доступны на странице <drupal>/admin/content (Контент). Здесь вы можете отредактировать, удалить, снять с публикации или опубликовать любую из созданных статей. Если публикаций слишком много, вы можете воспользоваться фильтром, чтобы отобразить публикации определенного типа, определенного языка (для мультиязычных сайтов) либо опубликованные/снятые с публикации/размещенные на главной странице/не размещенные на главной странице материалы.

Поля

Полей для ввода заголовка и содержимого публикации часто бывает недостаточно. Для структурирования информации к форме добавления публикации часто хочется присоединить отдельные поля, например для ввода размера, состава, источника либо для каких-нибудь других сведений. Кроме того, часто к каждой публикации определенного типа необходимо добавлять картинку.

Для решения перечисленных задач в CMS Drupal используются поля. Технология полей позволяет добавить к типу материалов любое количество дополнительных полей, а также определить, как именно их содержимое будет отображаться на странице публикации.

CMS Drupal поддерживает самые разные типы полей: текстовые, числовые, поля для загрузки файлов, поля для добавления картинок и т. д.

Добавление полей. Поля могут добавляться не только к типам материалов, но и к профилям пользователей, а также к терминам таксономии.

Для добавления полей к типу материалов либо для их удаления используется страница вида <drupal>/admin/structure/types/manage/<тип материала>/fields (Структура ▸ Типы содержимого ▸ «тип» ▸ Управлять полями).

По умолчанию каждый тип материалов имеет одно поле — для ввода содержимого. На указанной странице вы можете удалить либо изменить настройки этого поля.

Для добавления полей к профилю пользователя либо для их удаления используется страница вида <drupal>/admin/config/people/accounts/fields (Сайт ▸ Люди ▸ Настройки учетной записи ▸ Управлять полями).

Для добавления полей к словарю терминов таксономии либо для их удаления применяется страница вида <drupal>/admin/structure/taxonomy/«словарь»/fields (Структура ▸ Таксономия ▸ «словарь» ▸ Управлять полями).

Отображение полей. Настройка отображения полей выполняется на страницах:

- <drupal>/admin/structure/types/manage/<тип материала>/display (Структура ▸ Типы содержимого ▸ «тип» ▸ Управлять отображением);
- <drupal>/admin/config/people/accounts/display (Сайт ▸ Люди ▸ Настройки учетной записи ▸ Управлять отображением);
- <drupal>/admin/structure/taxonomy/«словарь»/display (Структура ▸ Таксономия ▸ «словарь» ▸ Управлять отображением).

Здесь вы можете:

- полностью скрыть поле;
- скрыть метку поля либо указать, как она должна отображаться (на одной строке со значением или над значением);
- определить способ вывода изображения (просто как картинка или, если установлен модуль Colorbox, как картинка со ссылкой на увеличенное изображение) и его заранее заданные настройки (размеры изображения).

Таксономия

Вы уже знаете, что такое теги. На многих сайтах они используются для структурирования информации. На большинстве сайтов даже есть облако тегов — список наиболее популярных тегов, при щелчке на которых открывается страница сайта со списком всех публикаций, которым присвоен данный тег.

Таксономия в CMS Drupal — это механизм, который позволяет назначать публикациям теги.

Отдельный тег в Drupal называется термином таксономии. Все термины таксономии в данной CMS организованы в словари. Термин таксономии не может не принадлежать к какому-либо словарю.

По умолчанию в CMS Drupal создан только один словарь — tags. Но вы можете создать любое количество словарей. Добавить новый словарь можно на странице <drupal>/admin/structure/taxonomy (Структура ▸ Таксономия ▸ Добавить словарь).

Список всех созданных словарей находится на странице <drupal>/admin/structure/taxonomy (Структура ▸ Таксономия). На этой странице вы можете добавить новые термины в любой из созданных словарей.

Создания словаря и добавления в него терминов таксономии недостаточно для того, чтобы настроить выбор тегов при создании публикации. Чтобы иметь возможность выбирать теги при создании публикации, вы должны нужному типу материалов добавить поле типа Ссылка на термин.

Поля типа Ссылка на термин можно добавлять не только типам материалов, но и профилям пользователей и даже словарям терминов таксономии.

Продолжить изучение возможностей CMS Drupal можно либо на официальном сайте CMS Drupal http://drupal.org (если вы знаете английский), либо на популярном русскоязычном сайте http://drupal.ru.

Глава 6

Раскрутка сайта

Поисковые системы

Каталоги сайтов

Рейтинговые системы

Прочие разновидности сервисов

Активная реклама

В современном мире недостаточно просто создать сайт. Более того, создание сайта — это лишь первый и далеко не самый важный шаг. Более важным шагом является раскрутка сайта, то есть привлечение посетителей на сайт.

И тут возникает вопрос: откуда можно взять этих самых посетителей?

Конечно, можно обзвонить всех своих друзей и рассказать им о новом замечательном сайте. А они, в свою очередь, вполне возможно, расскажут о сайте своим друзьям. А те своим. И так далее до тех пор, пока новость о вашем сайте не обойдет весь мир.

Думаете, мечты? Конечно, мечты. Хотя и такой способ активно используется рекламными компаниями. Он называется «сарафанное радио» и отлично подходит для различных социальных и рассчитанных на потребителя проектов. Только место телефона в этом способе давно уже заняли другие средства связи: социальные сети, блоги, иногда форумы.

Если вы человек не очень общительный и у вас нет нескольких тысяч друзей в социальных сетях и блогах, то стоит рассмотреть и другие источники посетителей для сайта. В первую очередь, поисковые системы.

Поисковые системы

Куда интернет-пользователь приходит, когда ему нужно что-то найти в Интернете? Конечно, на сайт популярной поисковой системы. Как правило, это Google (`http://google.ru`) или «Яндекс» (`http://yandex.ru`).

Пользователь Интернета просто вводит свой запрос, например «купить пушистых зайчиков», и поисковая система выдает список сайтов, на которых встречается искомая пользователем фраза в каком-либо контексте.

Добавляем сайт

Первой задачей, которую вам нужно решить, является добавление вашего сайта в поисковые системы, чтобы они проиндексировали содержимое сайта и он начал участвовать в поиске.

Конечно, поисковые роботы могут обнаружить и проиндексировать ваш сайт самостоятельно. Однако для этого другие сайты в Интернете должны ссылаться на ваш сайт. И, конечно, должно пройти какое-то время, пока ссылка на ваш сайт будет обнаружена роботом и пока у поисковой системы дойдут руки до индексации вашего сайта. В лучшем случае придется подождать несколько недель.

В общем, не стоит полагаться на волю случая. Тем более что добавление сайта в поисковые системы займет у вас примерно пять минут. Итак, приступим.

Первым делом добавим сайт в поисковые системы «Яндекс» и «Рамблер» (поисковая система «Рамблер» использует базу «Яндекса», поэтому вам достаточно добавить сайт только на «Яндекс»). Для этого введите адрес вашего сайта на странице http://webmaster.yandex.ua/addurl.xml, после чего нажмите кнопку Добавить. Собственно говоря, это все.

Далее не стоит забывать еще об одном гиганте среди поисковых систем — Google. Чтобы добавить сайт в данную поисковую систему, нужно ввести адрес сайта на странице http://www.google.ru/intl/ru/addurl.html.

Остальные поисковые системы не настолько важны, чтобы вручную добавлять на них сайт. В любом случае рано или поздно, но любая поисковая система обнаружит где-либо ссылку на ваш сайт и добавит его в свой индекс.

Файл sitemaps.xml

После того как сайт добавлен в поисковую систему, ее робот начнет периодически посещать страницы вашего сайта и заносить обнаруженную на них информацию в свой индекс.

В первую очередь робот посетит главную страницу вашего сайта — index.html, index.php, index.htm и т. д. Весьма вероятно, что на главной странице робот обнаружит ссылки на второстепенные страницы и посетит их. Там он также найдет ссылки на страницы сайта. И таким образом, путешествуя по ссылкам на сайте, робот проиндексирует весь ваш сайт.

Это стандартный способ общения поискового робота с вашим сайтом. Но есть и еще один более современный и более быстрый вариант индексации — файл `sitemap.xml`. Вы просто создаете файл, содержащий список ссылок на все страницы вашего сайта, после чего размещаете данный файл у себя на сайте и указываете поисковой системе, где он находится.

Создание файла sitemap.xml. Если вы разрабатываете сайты на CMS Drupal, то создание файла `sitemap.xml` заключается в установке и настройке модуля xmlsitemap. После этого файл `sitemap.xml` будет создан. И, более того, будет постоянно обновляться при каждом запуске крона[1].

Модуль xmlsitemap состоит из набора модулей. Вы можете включить только те из них, функционал которых вам действительно нужен.

---

[1] Сценарий в CMS Drupal, который периодически запускается и выполняет различные служебные задачи.

- XML sitemap — основной модуль, без которого не будет работать ни один из перечисленных ниже.
- XML sitemap custom — позволяет добавить в файл `sitemap.xml` ссылку на произвольную страницу вашего сайта. Вам достаточно просто указать ссылку на страницу, которую нужно добавить.
- XML sitemap engines — добавляет возможность оповестить популярные поисковые системы о наличии у вашего сайта файла `sitemap.xml` и его расположении.
- XML sitemap internationalization — используется при создании мультиязычного сайта.
- XML sitemap menu — позволяет добавить в `sitemap.xml` ссылки, являющиеся пунктами меню на вашем сайте. При этом вы можете указать меню, пункты которых будут добавлены в `sitemap.xml`.
- XML sitemap node — дает возможность добавить в `sitemap.xml` ссылки на публикации на вашем сайте. При этом вы можете указать типы материалов, публикации которых будут добавлены в `sitemap.xml`.
- XML sitemap taxonomy — позволяет добавить в `sitemap.xml` ссылки на разделы таксономии на вашем сайте. При этом вы можете указать словари, ссылки на теги из которых будут добавлены в `sitemap.xml`.
- XML sitemap user — дает возможность добавить в `sitemap.xml` ссылки на профили пользователей.

Если вы не используете CMS Drupal, файл `sitemap.xml` придется создавать самостоятельно. Либо же воспользоваться сервисами для создания файла `sitemap.xml`. Например, сервисом http://htmlweb.ru/analiz/sitemap.php.

Пример содержимого файла `sitemap.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="/sitemap.xsl"?>

<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">

<url><loc>http://www.angedesign.ru/booking/view</loc></url>

<url><loc>http://www.angedesign.ru/node/114</loc><lastmod>2011-
05-27T10:30Z</lastmod><changefreq>yearly</changefreq><priorit
y>0.4</priority></url>

</urlset>
```

Таким образом, каждая страница сайта описывается в отдельном теге `url`. При этом вложенные теги позволяют подробнее описать данную веб-страницу:

- `loc` — определяет URL веб-страницы;
- `lastmod` — дата последнего изменения содержимого страницы;
- `changefreq` — частота обновления содержимого страницы;
- `priority` — важность содержимого страницы и частота ее обновления.

При ручном создании файла `sitemap.xml` достаточно сохранять приведенную структуру.

ПРИМЕЧАНИЕ

На файл sitemap.xml налагается несколько ограничений. Он должен содержать не более 50 000 ссылок и имеет размер не более 10 Мбайт. Если ваш сайт содержит большее количество ссылок, файл sitemap.xml следует разбить на несколько файлов. После чего создать файл sitemaps.xml, в котором будут находиться ссылки на все созданные файлы sitemap.xml.

Публикация файла sitemap.xml. Как правило, файл `sitemap.xml` размещают в корне сайта.

Оповещение поисковых систем. После того как файл `sitemap.xml` создан и размещен на хостинге, о его наличии нужно сообщить поисковым системам. Для этого можно просто открыть веб-страницу вида:

- Google — http://google.com/webmasters/sitemaps/ping?sitemap=<путь к sitemap.xml>;
- «Яндекс» — http://webmaster.yandex.ru/wmconsole/sitemap_list.xml?host=<путь к sitemap.xml>;
- Yahoo! — http://search.yahooapis.com/SiteExplorerService/V1/ping?sitemap=<путь к sitemap.xml>;
- Ask — http://submissions.ask.com/ping?sitemap=<путь к sitemap.xml>;
- Bing — http://www.bing.com/webmaster/ping.aspx?siteMap=<путь к sitemap.xml>.

Или же вы можете зарегистрироваться в поисковых системах в качестве владельца сайта, после чего добавить файл `sitemap.xml` со страницы веб-мастера:

- для «Яндекса» — http://webmaster.yandex.ru;
- для Google — http://www.google.ru/webmasters/.

Основные правила SEO

Просто добавить сайт в поисковую систему недостаточно. Какой от этого толк, если сайт будет отображаться, допустим, на 500-й позиции в результатах поиска? Ни один человек со здравым рассудком не будет просматривать более 10 страниц с резуль-

татами поиска. Проще конкретизировать свой поисковый запрос. Поэтому толку от добавления вашего сайта в поисковую систему все равно не будет.

И тут возникает вопрос: как именно поисковая система определяет, на какой позиции размещать тот или иной сайт в результатах поиска. Вообще-то, подробности ответа на этот вопрос являются коммерческой тайной каждой поисковой системы. Но основные положения известны.

Любая поисковая система стремится в первой десятке результатов поиска отобразить наиболее полезные сайты, то есть сайты, на которых расположена наиболее полная и точная информация, касающаяся искомой фразы. Определяется «уровень полезности» сайта на основе разных условий, например следующих.

- Текст, расположенный в тегах заголовков (`h1`, `h2`, `h3`, ...), считается более важным, чем текст, размещенный в других тегах. Поэтому никогда не стоит создавать заголовки страниц с помощью тегов `div`, `span` и т. д. Для этого предназначены теги `h1`, `h2` и т. д. Пренебрегая ими, вы не только отклоняетесь от правильной семантической верстки, но и снижаете позиции вашего сайта в поисковых системах.
- Заголовок уровня `h1` на веб-странице должен быть только один. Его содержимое особенно важно с точки зрения поисковых систем. Поэтому не стоит использовать тег `h1` для указания названия сайта, слогана или другой маловажной информации. Лучше применять его, чтобы кратко указать содержимое текущей веб-страницы, а не заносить в него информацию, которая повторяется на каждой странице веб-сайта.
- Не стоит также пренебрегать тегами `strong` и `i`. Содержимое этих тегов поисковыми системами считается более важным, чем обычный текст, расположенный в тегах `div` и `span`.
- Для каждого значащего (не являющегося элементом интерфейса) изображения на веб-странице следует указывать атрибут `alt` с кратким описанием содержимого. Не стоит забывать о существовании такого сервиса, как поиск картинок. Данный тип поиска осуществляется на основе содержимого атрибута `alt` картинки, а также текста, который расположен рядом с ней.
- Среди известных нам HTML-тегов есть такой, который напрямую относится к SEO. Это тег `meta`, точнее, его разновидности, позволяющие задавать ключевые слова и краткое описание текущей страницы. Содержимое этих тегов может использоваться поисковой системой для определения позиции вашего сайта в результатах поиска.
- Ключевые слова указываются через запятую в теге вида `<meta http-equiv="keywords" content="ключевые слова">`. Как вы уже знаете, данный тег должен быть расположен внутри тега `head`. Содержимое атрибута `content`

должно состоять из слов, которые описывают содержимое текущей веб-страницы. При этом в начале должны следовать наиболее значимые слова. Кроме того, важно, чтобы указанные вами ключевые слова также повторялись в тексте самой веб-страницы.

На данный момент многие поисковые системы утверждают, что больше не используют ключевые слова для определения позиции сайта. Поэтому указание ключевых слов в большей части стало традицией.

- Краткое описание указывается в теге вида `<meta http-equiv="description" content="описание">`. Этот тег должен располагаться внутри тега `head`. Данный тег должен содержать краткое описание всего сайта, его направление, цели и т. д. На остальных страницах значение атрибута `content` описывает содержимое самой страницы. Длина значения для атрибута `content` ограничена 1024 символами.
- Содержимое тега `title` также важно с точки зрения поисковых систем. Его следует начинать со значащего слова: услуги, тематики, продаваемого продукта, торговой марки.

Это лишь небольшой список наиболее простых правил, которые следует всегда учитывать верстальщику и разработчику сайта.

Кроме того, позиция каждого сайта зависит от уровня доверия поисковой системы к этому сайту, то есть от возраста сайта, количества ссылок на этот сайт с других сайтов в Интернете и т. д.

Раскрутка сайта в поисковых системах — это целая наука, достойная отдельной книги. Люди, занимающиеся ею, называются SEO-специалистами (или «сеошниками»). При необходимости всегда можно заказать раскрутку сайта такому специалисту.

Каталоги сайтов

Разновидностью поисковых систем являются каталоги сайтов — сайты, которые содержат краткое описание других сайтов. Каталоги сайтов менее популярны, чем поисковые системы. Но тем не менее многие люди ими пользуются, чтобы найти интересные для себя сайты.

Как правило, регистрация сайта в каталоге заключается во внесении таких сведений, как название и описание вашего сайта, выбор направления работы сайта из списка. Чаще всего регистрация нового сайта выполняется бесплатно. Вам достаточно зайти на сайт каталога, найти там страницу добавления нового сайта и заполнить все поля.

На просторах Интернета можно найти десятки тысяч различных каталогов сайтов. Я перечислю лишь наиболее популярные из русскоязычных каталогов:

- «АПОРТ» (http://catalog.aport.ru/rus/thememap.aspx);
- Mail.Ru (http://list.mail.ru/index.html);
- Russia on the Net (http://www.ru);
- Chat.ru (http://catalog.chat.ru);
- «УЛИТКА» (http://www.ulitka.ru).

Рейтинговые системы

Рейтинговыми системами называются разновидности каталогов сайтов, которые не только хранят сведения о сайтах, но и определяют их популярность с помощью счетчика количества посетителей. Таким образом, чтобы добавить свой сайт в какой-либо рейтинг, необходимо не только указать сведения о нем, но и поставить HTML-код счетчика рейтинга на свой сайт. В результате чем выше будет рейтинг вашего сайта, тем больше посетителей будет приходить к вам с рейтинговой системы.

Наиболее популярны следующие рейтинговые системы:

- «Рамблер Топ 100» (http://top100.rambler.ru);
- Mail.Ru (http://top.mail.ru);
- Openstat (http://openstat.ru).

Прочие разновидности сервисов

Поисковые системы, каталоги ссылок и рейтинги — это далеко не единственные разновидности сервисов, где можно оставить информацию о своем сайте. Помимо них, существуют и другие интересные для владельца сайта сервисы.

Закладки. Сервисы закладок являются некой разновидностью каталогов ссылок. Они также хранят сведения о других сайтах. Однако в сервисах закладок ссылки можно добавлять не только на главные страницы сайтов, но и на любые внутренние страницы.

Наиболее популярны следующие русскоязычные сервисы закладок:

- http://memori.qip.ru;
- http://bobrdobr.ru;

- http://moemesto.ru;
- http://100zakladok.ru;
- http://mister-wong.ru;
- http://vaau.ru;
- http://google.com/bookmarks;
- http://friendfeed.com;
- http://zakladki.yandex.ru;
- http://toodoo.ru;
- http://zakladok.net.

Каталоги статей. Как можно судить из названия, каталоги статей позволяют публиковать статьи на различные темы. При этом в теле статьи можно указывать ссылки на любые сайты. В нашем случае на свой сайт. Можно выделить следующие каталоги статей:

- http://pisali.ru;
- http://50rus.info;
- http://articles-directory.ru;
- http://catalog-statey.ru;
- http://pautinka.biz;
- http://gammalist.ru;
- http://kata-log.ru.

Пресс-релизы. Разновидностью каталогов сайтов являются сайты, позволяющие добавить пресс-релиз какого либо события, продукта или компании.

Общетематические сервисы:

- http://b2blogger.com;
- http://subscribe.ru/release;
- http://openmarket.ru/p_releases.asp;
- http://www.press-release.ru;
- http://www.karta-smi.ru;
- http://www.pikabu.ru;
- http://www.pr-release.ru;
- http://www.prtime.ru;

- http://pr.adcontext.net;
- http://press.prnews.ru/release_list.asp;
- http://inthepress.ru;
- http://presuha.ru/release.

Интернет, ИТ:

- http://www.big-news.ru;
- http://it4business.ru/about/add;
- http://www.cybersecurity.ru/press;
- http://www.fubix.ru/press;
- http://www.pressroom.ru;
- http://soft.mail.ru/press-release.php;
- http://www.cityindex.ru/ci/pr_add.shtml.

Логистика, перевозки:

- http://www.lexim.ru/invite/releases;
- http://www.perevozki.ru.

Недвижимость:

- http://www.realestate-today.ru/news/add_release.html;
- http://www.restate.ru/add_news.html;

Нефть, химия, фармакология:

- http://rcc.ru/Rus/?ID=11567;
- http://www.cmna.ru/?ID=46496;

Промышленность:

- http://www.oborudunion.ru;
- http://www.opt-union.ru;
- http://www.snabjenci.ru;
- http://avtomatpro.info/press;
- http://masteroff.ru;
- http://press-24.ru;
- http://upack.by/press_release.

Реклама:

- http://re-port.ru/press;
- http://www.reclama.su;
- http://4p.ru;
- http://propel.ru/news.

Туризм:

- http://www.tourinfo.ru/press_release/add;
- http://www.otpusk.com/news.

Финансы, банки:

- http://www.gaap.ru/press/client;
- http://www.credcard.ru/admin/registration.

Украинские ресурсы:

- http://www.press-release.com.ua;
- http://www.newsroom.com.ua.

Онлайн RSS-агрегаторы. Сервисы данного класса каталогизируют и собирают информацию с RSS-каналов сайтов. Вы легко можете добавить в них и RSS-канал своего сайта:

- http://www.prime-rss.ru/add.php;
- http://rss.ukrnews.net/reg.php;
- http://blogs.yandex.ru/add.xml;
- http://liverss.ru/;
- http://wmas.msk.ru/signup.php;
- http://www.blogdir.ru/add_blog;
- http://www.redtram.ru/pages/addsource/;
- http://rssreader.ru/addfeed;
- http://www.plazoo.com/ru/addrss.asp;
- http://www.rss.zelenkov.net/rss_add.php.

Активная реклама

Все перечисленные выше способы раскрутки сайта являются пассивной рекламой, то есть вы размещаете где-либо ссылку на свой сайт, а потом просто ждете, пока

заинтересованный в вас посетитель найдет вашу ссылку и перейдет по ней. В этом случае посетитель ищет ваш сайт.

Противоположностью пассивной является активная реклама. В этом случае сайт ищет посетителя и сведения о вашем сайте или предложении навязываются пользователям Интернета.

В отличие от пассивной рекламы, активная реклама платная. Вы платите либо реальными деньгами, либо просмотрами баннеров других сайтов на своем сайте.

К активной рекламе относятся баннерообменные сети, контекстная реклама, рекламные ссылки.

Баннерообменные сети позволяют разместить ваш баннер на десятках тысяч сайтов в Интернете. При этом вы можете ограничить список сайтов, на которых будет размещен ваш баннер, тематикой, языком, местоположением и т. д.

В баннерообменных сетях платят за показы, то есть вы платите за количество показов вашего баннера на других сайтах. Как правило, количество показов покупается тысячами. При этом цена тысячи показов баннера варьируется от $1 до 2.

Популярные баннерные сети:

- http://rle.ru;
- http://rb2.design.ru;
- http://ir.ru;
- http://tbn.ru.

Контекстная реклама занимается размещением блоков с рекламной информацией на сайтах партнеров. Причем контекстной такая реклама называется потому, что посетителю предлагается только та рекламная информация, которая ему интересна: либо она соответствует содержимому текущей страницы, либо пользователь недавно искал что-то подобное в поисковых системах.

В контекстной рекламе вы платите только за переходы на ваш сайт по рекламной ссылке. Однако стоимость одного перехода достаточно дорога и может достигать нескольких долларов.

Популярные сервисы контекстной рекламы:

- http://www.begun.ru;
- http://direct.yandex.ru;
- http://google.com/adsense.

Приложение 1

Описание сообщений, отображаемых валидаторами на основе HTML Tidy

В данном приложении приведено описание основных ошибок и предупреждений, которые отображают валидаторы, работающие на основе HTML Tidy; а также способы решения обнаруженных проблем.

- `entity "..." doesn't end in ";"`

  В конце спецсимвола нет точки с запятой (например, ` ` вместо ` `).

- `numeric character reference "..." doesn't end in ';'`

  В конце числового спецсимвола нет точки с запятой (например, `&#8212` вместо `&#8212;`).

- `unescaped & or unknown entity "&..."`

  В HTML-коде обнаружен символ `&`, зарезервированный для создания спецсимволов. Для решения проблемы замените символ `&` на спецсимвол `&`.

- `missing </...>`

  Для парного тега не обнаружен закрывающий тег.

- `missing </aaa> before <bbb>`

  Блочный тег находится внутри встроенного, что недопустимо правилами HTML.

- `discarding unexpected <...>`

  Для парного тега не обнаружен либо открывающий тег, либо закрывающий тег.

- `nested emphasis ...`

  Ошибка говорит о повторном применении одного и того же тега физического форматирования. Например, `<b><b>текст</b></b>` нужно заменить на `<b>текст</b>`.

- `replacing unexpected ... by </...>`

  Внутри парного тега обнаружены один незакрытый парный тег и один парный тег без открывающей пары. Например, `<div><em>текст</strong></div>`.

- `isn't allowed in <...> elements`

  Внутри парного тега обнаружен тег, который по правилам не может находиться в данном парном теге.

- `missing <...>`

  Не обнаружен тег, который обязательно должен располагаться внутри данной структуры. Как пример, отсутствие тега `<tr>` при создании таблицы.

- `inserting implicit <...>`

  Обнаружена ошибка, в возникновении которой виноваты ранее обнаруженные валидатором проблемы.

- `Insert missing <title> element`

  В HTML-документе не обнаружен тег `<title>`.

- `Multiple <frameset> elements`

  В HTML-документе обнаружено несколько тегов `<frameset>`, не вложенных друг в друга. Для решения проблемы необходимо вложить теги `<frameset>` друг в друга.

- `<...> is not approved by W3C`

  В HTML-документе обнаружен тег, не входящий в спецификацию HTML. Замените его аналогичным по функционалу.

- `Error: <...> is not recognized!`

  В HTML-документе обнаружен тег, неизвестный валидатору.

- `Trimming Empty Tag`

  Внутри парного тега нет никакого текста либо стоит пробел. Для решения проблемы замените обычный пробел спецсимволом ` ` либо введите в теге какой-либо текст.

- `<тег> is probably intended as </тег>`

  Для указанного в сообщении тега обнаружен предположительно закрывающий тег, но без слеша. Например: `<a href="">текст<a>`.

- `... shouldn't be nested`

  Обнаружено несколько одинаковых тегов, вложенных друг в друга. Причем по правилам для данных тегов вложение друг в друга запрещено.

- `Text found after closing </body>-tag`

  В HTML-документе после закрывающего тега `</body>` обнаружен какой-то текст.

- `Adjacent hyphens within comment`

  В теле комментария обнаружены два и более идущих подряд дефиса. Такие конструкции в комментариях запрещены. Например: `<!-- текст -- текст -->`.

- `SYSTEM, PUBLIC, W3C, DTD, EN must be upper case`

  Слова `SYSTEM`, `PUBLIC`, `W3C`, `DTD`, `EN` в `DOCTYPE` обязательно должны быть написаны в верхнем регистре. Иначе возникнет подобная ошибка.

- `missing <!DOCTYPE> declaration`

  В HTML-документе не обнаружен `DOCTYPE`.

- `Too much <...>-elements`

  В HTML-документе обнаружено несколько экземпляров тега, который должен встречаться только один раз.

- `<...> inserting "..." attribute`

  `<...> lacks "..." attribute`

  Для тега не указан обязательный атрибут.

- `... attribute ... lacks value`

  Для атрибута тега не указано обязательное значение.

- `... attribute "..." has invalid value "..."`

  Для атрибута тега указано некорректное значение.

- `<...> missing > for end of tag`

  В HTML-коде обнаружен символ >, вместо использования спецсимвола `>`: `<p>текст > текст</p>`. Еще одной причиной возникновения данной ошибки является отсутствие символа > в конце тега: `<p` вместо `<p>`.

- `<...> proprietary attribute "..."`

  Для тега обнаружен атрибут, который не входит в спецификацию HTML для этого тега. Замените данный атрибут аналогичным по функциональности кодом.

- `... proprietary attribute value "..."`

  Для атрибута обнаружено значение, которое не входит в спецификацию HTML для этого атрибута.

- `... dropping value "..." for repeated attribute "..."`

  В теге обнаружено несколько одинаковых атрибутов.

- `... unexpected or duplicate quote mark`

  При написании значения тега пропущена открывающая или закрывающая кавычка. Например: `<a href=http://test.ru">`.

- `attribute with missing trailing quote mark`

  Для значения тега количество открывающих и закрывающих кавычек не совпадает. Например: `<a href="http://test.ru"">`.

- `id and name attribute value mismatch`

Для тега одновременно указан атрибут `id` и атрибут `name`. Причем значения для них различаются.

- `replacing <...> by <...>`

  Данная ошибка возникает в следующих случаях:

  - неправильный порядок тегов;
  - имеется лишний закрывающий тег;
  - имеется парный открывающий тег, но для него отсутствует закрывающий.

- `... anchor "..." already defined`

  В документе обнаружено несколько тегов с одинаковым значением атрибута `name`.

Приложение 2

Популярные спецсимволы

В табл. П2.1–П2.10 приведены коды некоторых популярных спецсимволов.

Таблица П2.1. Основные спецсимволы

Имя	Код	Символ	Описание
&para;	&#182;	¶	Символ абзаца
&sect;	&#167;	§	Параграф
&copy;	&#169;	©	Знак Copyright
&reg;	&#174;	®	Знак зарегистрированной торговой марки
&trade;	&#8482;	™	Знак торговой марки
&deg;	&#176;	°	Градус
&plusmn;	&#177;	±	Плюс-минус
&frac14;	&#188;	¼	Дробь — одна четверть
&frac12;	&#189;	½	Дробь — одна вторая
&frac34;	&#190;	¾	Дробь — три четверти
&times;	&#215;	×	Знак умножения
&divide;	&#247;	÷	Знак деления
&fnof;	&#402;	*f*	Знак функции
&permil;	&#8240;	‰	Знак промилле

Таблица П2.2. Греческие символы

Имя	Код	Символ	Описание
&Alpha;	&#913;	A	Греческая заглавная буква «альфа»
&Beta;	&#914;	B	Греческая заглавная буква «бета»
&Gamma;	&#915;	Γ	Греческая заглавная буква «гамма»
&Delta;	&#916;	Δ	Греческая заглавная буква «дельта»
&Epsilon;	&#917;	E	Греческая заглавная буква «эпсилон»
&Zeta;	&#918;	Z	Греческая заглавная буква «дзета»
&Eta;	&#919;	H	Греческая заглавная буква «эта»
&Theta;	&#920;	Θ	Греческая заглавная буква «тета»
&Iota;	&#921;	I	Греческая заглавная буква «иота»
&Kappa;	&#922;	K	Греческая заглавная буква «каппа»
&Lambd;	&#923;	Λ	Греческая заглавная буква «лямбда»
&Mu;	&#924;	M	Греческая заглавная буква «мю»
&Nu;	&#925;	N	Греческая заглавная буква «ню»
&Xi;	&#926;	Ξ	Греческая заглавная буква «кси»
&Omicron;	&#927;	O	Греческая заглавная буква «омикрон»
&Pi;	&#928;	Π	Греческая заглавная буква «пи»
&Rho;	&#929;	P	Греческая заглавная буква «ро»
&Sigma;	&#931;	Σ	Греческая заглавная буква «сигма»
&Tau;	&#932;	T	Греческая заглавная буква «тау»
&Upsilon;	&#933;	Y	Греческая заглавная буква «ипсилон»

Имя	Код	Символ	Описание
Φ	Φ	Φ	Греческая заглавная буква «фи»
Χ	Χ	Χ	Греческая заглавная буква «хи»
Ψ	Ψ	Ψ	Греческая заглавная буква «пси»
Ω	Ω	Ω	Греческая заглавная буква «омега»
α	α	α	Греческая строчная буква «альфа»
β	β	β	Греческая строчная буква «бета»
γ	γ	γ	Греческая строчная буква «гамма»
δ	δ	δ	Греческая строчная буква «дельта»
ε	ε	ε	Греческая строчная буква «эпсилон»
ζ	ζ	ζ	Греческая строчная буква «дзета»
η	η	η	Греческая строчная буква «эта»
θ	θ	θ	Греческая строчная буква «тета»
ι	ι	ι	Греческая строчная буква «иота»
κ	κ	κ	Греческая строчная буква «каппа»
λ	λ	λ	Греческая строчная буква «лямбда»
μ	μ	μ	Греческая строчная буква «мю»
ν	ν	ν	Греческая строчная буква «ню»
ξ	ξ	ξ	Греческая строчная буква «кси»
ο	ο	ο	Греческая строчная буква «омикрон»
π	π	π	Греческая строчная буква «пи»
ρ	ρ	ρ	Греческая строчная буква «ро»
ς	ς	ς	Греческая строчная буква «сигма»
σ	σ	σ	Греческая строчная буква «сигма»
τ	τ	τ	Греческая строчная буква «тау»
υ	υ	υ	Греческая строчная буква «ипсилон»
φ	φ	φ	Греческая строчная буква «фи»
χ	χ	χ	Греческая строчная буква «хи»
ψ	ψ	ψ	Греческая строчная буква «пси»
ω	ω	ω	Греческая строчная буква «омега»

Таблица П2.3. Символы различных алфавитов

Имя	Код	Символ	Описание
¨	¨	¨	Знак диереза
¡	¡	¡	Перевернутый восклицательный знак
¿	¿	¿	Перевернутый вопросительный знак
À	À	À	Прописная «А» с тупым ударением
Á	Á	Á	Прописная «А» с острым ударением
Â	Â	Â	Прописная «А» с циркумфлексом

Продолжение ⇩

Таблица П2.3 *(продолжение)*

Имя	Код	Символ	Описание
Ã	Ã	Ã	Прописная «A» с тильдой
Ä	Ä	Ä	Прописная «A» с диерезой
Å	Å	Å	Прописная «A» с кружком
Æ	Æ	Æ	Прописная лигатура «AE»
Ç	Ç	Ç	Прописная «C» с седилем
È	È	È	Прописная «E» с тупым ударением
É	É	É	Прописная «E» с острым ударением
Ê	Ê	Ê	Прописная «E» с циркумфлексом
Ë	Ë	Ë	Прописная «E» с диерезой
Ì	Ì	Ì	Прописная «I» с тупым ударением
Í	Í	Í	Прописная «I» с острым ударением
Î	Î	Î	Прописная «I» с циркумфлексом
Ï	Ï	Ï	Прописная «I» с диерезой
Ð	Ð	Ð	Прописная буква исландского алфавита Eth
Ñ	Ñ	Ñ	Прописная «N» с тильдой
Ò	Ò	Ò	Прописная «O» с тупым ударением
Ó	Ó	Ó	Прописная «O» с острым ударением
Ô	Ô	Ô	Прописная «O» с циркумфлексом
Õ	Õ	Õ	Прописная «O» с тильдой
Ö	Ö	Ö	Прописная «O» с диерезой
Ø	Ø	Ø	Прописная «O» перечеркнутое
Ù	Ù	Ù	Прописная «U» с тупым ударением
Ú	Ú	Ú	Прописная «U» с острым ударением
Û	Û	Û	Прописная «U» с циркумфлексом
Ü	Ü	Ü	Прописная «U» с диерезой
Ý	Ý	Ý	Прописная «Y» с острым ударением
Þ	Þ	Þ	Буква «торн»
ß	ß	ß	Двойная «s» (эсцет)
à	à	à	Строчная «a» с тупым ударением
á	á	á	Строчная «a» с острым ударением
â	â	â	Строчная «a» с циркумфлексом
ã	ã	ã	Строчная «a» с тильдой
ä	ä	ä	Строчная «a» с диерезой
å	å	å	Строчная «a» с кружком
æ	æ	æ	Строчная лигатура «ae»
ç	ç	ç	Строчная «c» с седилем
è	è	è	Строчная «e» с тупым ударением
é	é	é	Строчная «e» с острым ударением

Имя	Код	Символ	Описание
ê	ê	ê	Строчная «e» с циркумфлексом
ë	ë	ë	Строчная «e» с диерезой
ì	ì	ì	Строчная «i» с тупым ударением
í	í	í	Строчная «i» с острым ударением
î	î	î	Строчная «i» с циркумфлексом
ï	ï	ï	Строчная «i» с диерезой
ð	ð	ð	Строчная буква исландского алфавита Eth
ñ	ñ	ñ	Строчная «n» с тильдой
ò	ò	ò	Строчная «o» с тупым ударением
ó	ó	ó	Строчная «o» с острым ударением
ô	ô	ô	Строчная «o» с циркумфлексом
õ	õ	õ	Строчная «o» с тильдой
ö	ö	ö	Строчная «o» с диерезой
ø	ø	ø	Строчная «o» перечеркнутое
ù	ù	ù	Строчная «u» с тупым ударением
ú	ú	ú	Строчная «u» с острым ударением
û	û	û	Строчная «u» с циркумфлексом
ü	ü	ü	Строчная «u» с диерезой
ý	ý	ý	Строчная «y» с острым ударением
þ	þ	þ	Строчная буква исландского алфавита «торн»
ÿ	ÿ	ÿ	Строчная «y» с диерезой
℘	℘	℘	Рукописная «P»
ℵ	ℵ	ℵ	Алеф
Ÿ	Ÿ	Ÿ	Прописная «Y» с диерезой
š	š	š	Строчная «S» с гачеком
Š	Š	Š	Прописная «S» с гачеком
œ	œ	œ	Строчная лигатура «oe»
Œ	Œ	Œ	Прописная лигатура «OE»

Таблица П2.4. Стрелки

Имя	Код	Символ	Описание
←	←	←	Стрелка влево
↑	↑	↑	Стрелка вверх
→	→	→	Стрелка вправо
↓	↓	↓	Стрелка вниз
↔	↔	↔	Стрелка влево-вправо
↵	↵	↵	Возврат каретки
⇐	⇐	⇐	Двойная стрелка влево

Продолжение ⇩

Таблица П2.4 *(продолжение)*

Имя	Код	Символ	Описание
⇑	⇑	⇑	Двойная стрелка вверх
⇒	⇒	⇒	Двойная стрелка вправо
⇓	⇓	⇓	Двойная стрелка вниз
⇔	⇔	⇔	Двойная стрелка влево-вправо
	◄	◄	Влево
	▲	▲	Вверх
	►	►	Вправо
	▼	▼	Вниз

Таблица П2.5. Прочие символы

Имя	Код	Символ	Описание
♠	♠	♠	Знак масти «пики»
♣	♣	♣	Знак масти «трефы»
♥	♥	♥	Знак масти «червы»
♦	♦	♦	Знак масти «бубны»
"	"	"	Двойная кавычка
&	&	&	Амперсанд
<	<	<	Знак «меньше»
>	>	>	Знак «больше»
•	•	•	Маркер списка
‾	‾	‾	Надчеркивание
‡	‡	‡	Двойной кинжал
†	†	†	Кинжал
˜	˜	~	Тильда
ˆ	ˆ	^	Циркумфлекс
¦	¦	¦	Вертикальная черта
ª	ª	ª	Показатель женского рода
º	º	º	Показатель мужского рода
·	·	·	Средняя точка
¸	¸	¸	Седиль

Таблица П2.6. Знаки пунктуации

Имя	Код	Символ	Описание
…	…	…	Многоточие …
′	′	′	Одиночный штрих — минуты и футы
″	″	″	Двойной штрих — секунды и дюймы

Таблица П2.7. Общая пунктуация

Имя	Код	Символ	Описание
&ndash;	&#8211;	–	Тире
&mdash;	&#8212;	—	Длинное тире
&lsquo;	&#8216;	‘	Левая одиночная кавычка
&rsquo;	&#8217;	’	Правая одиночная кавычка
&sbquo;	&#8218;	,	Нижняя одиночная кавычка
&ldquo;	&#8220;	“	Левая двойная кавычка
&rdquo;	&#8221;	”	Правая двойная кавычка
&bdquo;	&#8222;	„	Нижняя двойная кавычка
&laquo;	&#171;	«	Левая двойная угловая кавычка
&raquo;	&#187;	»	Правая двойная угловая кавычка
&rsaquo;	&#8250;	›	Закрывающая угловая кавычка
&lsaquo;	&#8249;	‹	Открывающая угловая кавычка
&shy;	&#173;		Мягкий перенос
&acute;	&#180;	´	Острое ударение

Таблица П2.8. Математические операторы

Имя	Код	Символ	Описание
&image;	&#8465;	ℑ	Мнимая часть числа
&real;	&#8476;	ℜ	Действительная часть числа
&forall;	&#8704;	∀	Квантор всеобщности
&part;	&#8706;	∂	Знак дифференциала
&exist;	&#8707;	∃	Квантор существования
&empty;	&#8709;	∅	Пустое множество
&nabla;	&#8711;	∇	Набла
&isin;	&#8712;	∈	Принадлежит множеству
&notin;	&#8713;	∉	Не принадлежит множеству
&ni;	&#8715;	∋	Является членом
&prod;	&#8719;	∏	*N*-арное произведение
&sum;	&#8721;	∑	*N*-арная сумма
&minus;	&#8722;	−	Знак «минус»
&lowast;	&#8727;	*	Оператор «звездочка»
&radic;	&#8730;	√	Радикал
&prop;	&#8733;	∝	Пропорционально
&infin;	&#8734;	∞	Бесконечность
&ang;	&#8736;	∠	Угол
&and;	&#8743;	∧	Логическое И
&or;	&#8744;	∨	Логическое ИЛИ

Продолжение ⇩

Таблица П2.8 *(продолжение)*

Имя	Код	Символ	Описание
∩	∩	∩	Пересечение
∪	∪	∪	Объединение
∫	∫	∫	Интеграл
∴	∴	∴	Следовательно
∼	∼	∼	Оператор «тильда»
≅	≅	≅	Приблизительно равно
≈	≈	≈	Асимптотически равно
≠	≠	≠	Не равно
≡	≡	≡	Тождественно равно
≤	≤	≤	Меньше или равно
≥	≥	≥	Больше или равно
⊂	⊂	⊂	Подмножество
⊃	⊃	⊃	Надмножество
⊄	⊄	⊄	Не подмножество
⊆	⊆	⊆	Подмножество или равно
⊇	⊇	⊇	Надмножество или равно
⊕	⊕	⊕	Прямая сумма
⊗	⊗	⊗	Векторное произведение
⊥	⊥	⊥	Перпендикулярно
⋅	⋅	·	Оператор «точка»
⌈	⌈	⌈	Левый верхний угол
⌉	⌉	⌉	Правый верхний угол
⌊	⌊	⌊	Левый нижний угол
⌋	⌋	⌋	Правый нижний угол
⟨	〈	<	Левая угловая скобка
⟩	〉	>	Правая угловая скобка
¬	¬	¬	Знак отрицания
²	²	²	Вторая степень
³	³	³	Третья степень
µ	µ	µ	Знак «микро»
¹	¹	¹	Единица в верхнем индексе

Таблица П2.9. Пробелы

Имя	Код	Описание
		Короткий пробел
		Длинный пробел
		Узкий пробел

Имя	Код	Описание
&zwnj;	&#8204;	Разделитель нулевой ширины
&zwj;	&#8205;	Соединитель нулевой ширины
		Неразрывный пробел

Таблица П2.10. Денежные знаки

Имя	Код	Символ	Описание
&pound;	&#163;	£	Фунт стерлингов
&euro;	&#8364;	€	Евро
&cent;	&#162;	¢	Цент
&curren;	&#164;	¤	Знак денежной единицы
&yen;	&#165;	¥	Иена

Приложение 3

Список тегов, версии браузеров и спецификации HTML, в которых работают перечисленные теги

В табл. П3.1 приведен список тегов, а также показаны версии браузеров и спецификации HTML, в которых работают перечисленные теги.

При этом используются следующие сокращения:

- IE — браузер Internet Explorer;
- O — Opera;
- FF — Mozilla Firefox;
- GC — Google Chrome;
- S — Safari.

После названия браузера идет его версия.

Таблица П3.1. Совместимость тегов с браузерами

Тег	IE 6	IE 7	O 9	O 10	S 3	FF 3	GC 10	HTML 4	XHTML	HTML5
a	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
abbr	Нет	Нет	Да	Да	Да	Да	Нет	Да	Да	Да
acronym	Да	Да	Да	Да	Да	Да	Да	Да	Да	Нет
address	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
applet	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
area	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
article	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
aside	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
audio	Нет	Нет	Нет	Нет	Да	Нет	Да	Нет	Нет	Да
b	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
base	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
basefont	Да	Да	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
bdo	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
bgsound	Да	Да	Да	Да	Нет	Нет	Нет	Нет	Нет	Нет
big	Да	Да	Да	Да	Да	Да	Да	Да	Да	Нет
blink	Нет	Нет	Да	Да	Нет	Да	Нет	Нет	Нет	Нет
blockquote	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
body	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
br	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
button	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
canvas	Нет	Нет	Нет	Да	Да	Нет	Да	Нет	Нет	Да
caption	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
center	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
cite	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
code	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да

Продолжение ⇩

Таблица П3.1 *(продолжение)*

Тег	IE 6	IE 7	O 9	O 10	S 3	FF 3	GC 10	HTML 4	XHTML	HTML5
col	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
colgroup	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
command	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Да
comment	Да	Да	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
datalist	Нет	Нет	Да	Да	Нет	Нет	Нет	Нет	Нет	Да
dd	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
del	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
details	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Да
dfn	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
dir	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
div	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
dl	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
dt	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
em	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
embed	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Да
fieldset	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
figcaption	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
figure	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
font	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
footer	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
form	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
frame	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
frameset	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
h1	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
h2	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
h3	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
h4	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
h5	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
h6	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
head	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
header	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
hgroup	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
hr	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
html	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
i	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
iframe	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Да
img	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
input	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
ins	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да

Тег	IE 6	IE 7	O 9	O 10	S 3	FF 3	GC 10	HTML 4	XHTML	HTML5
isindex	Да	Да	Нет	Нет	Да	Да	Да	Нет	Нет	Нет
kbd	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
keygen	Нет	Нет	Да	Да	Да	Да	Да	Да	Да	Нет
label	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
legend	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
li	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
link	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
listing	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
map	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
mark	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
marquee	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
menu	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Да
meta	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
meter	Нет	Нет	Нет	Нет	Нет	Нет	Да	Нет	Нет	Да
multicol	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет
nav	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
nobr	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
noembed	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
noframes	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
noscript	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
object	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
ol	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
optgroup	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
option	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
output	Нет	Нет	Нет	Да	Нет	Нет	Нет	Нет	Нет	Да
p	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
param	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
plaintext	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
pre	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
progress	Нет	Нет	Нет	Нет	Нет	Нет	Да	Нет	Нет	Да
q	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
rp	Да	Да	Нет	Нет	Нет	Нет	Да	Нет	Нет	Да
rt	Да	Да	Нет	Нет	Нет	Нет	Да	Нет	Нет	Да
ruby	Да	Да	Нет	Нет	Нет	Нет	Да	Нет	Нет	Да
s	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Да
samp	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
script	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
section	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
select	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да

Продолжение ⇩

Таблица П3.1 *(продолжение)*

Тег	IE 6	IE 7	O 9	O 10	S 3	FF 3	GC 10	HTML 4	XHTML	HTML5
small	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
source	Нет	Нет	Нет	Нет	Да	Нет	Да	Нет	Нет	Да
spacer	Нет	Нет	Нет	Нет	Нет	Да	Нет	Нет	Нет	Нет
span	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
strike	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
strong	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
style	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
sub	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
summary	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Да
sup	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
table	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
tbody	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
td	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
textarea	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
tfoot	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
th	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
thead	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
time	Нет	Нет	Да	Да	Нет	Нет	Да	Нет	Нет	Да
title	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
tr	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
track	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Нет	Да
tt	Да	Да	Да	Да	Да	Да	Да	Да	Да	Нет
u	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет
ul	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
var	Да	Да	Да	Да	Да	Да	Да	Да	Да	Да
video	Нет	Нет	Нет	Нет	Да	Нет	Да	Нет	Нет	Да
wbr	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Да
xmp	Да	Да	Да	Да	Да	Да	Да	Нет	Нет	Нет

Приложение 4

Свойства CSS

В табл. 4.1 приведены свойства CSS. Таблица состоит из следующих полей:

- «Свойство» — название свойства CSS;
- «Насл.» — наследуется ли данное свойство;
- «CSS» — версия CSS, в которой впервые появилось данное свойство;
- «IE» — версия браузера Internet Explorer, в которой впервые была реализована поддержка данного свойства;
- «По умолчанию» — значение по умолчанию для данного CSS-свойства;
- «Применяется» — типы тегов, к которым можно применять данное свойство.

Таблица П4.1. Описание свойств CSS

Свойство	Насл.	CSS	IE	По умолчанию	Применяется
background	Нет	CSS 1	6.0	transparent ‖ none ‖ repeat ‖ scroll ‖ 0% 0%	Ко всем
background-attachment	Нет	CSS 1	6.0	scroll	Ко всем
background-clip	Нет	CSS 3	9.0	border-box	Ко всем
background-color	Нет	CSS 1	6.0	transparent	Ко всем
background-image	Нет	CSS 1	6.0	none	Ко всем
background-origin	Нет	CSS 3	9.0	padding-box	Ко всем
background-position	Нет	CSS 1	6.0	0% 0%	К блочным элементам
background-repeat	Нет	CSS 1	6.0	repeat	Ко всем
background-size	Нет	CSS 3	9.0	auto	Ко всем
border	Нет	CSS 1	6.0	Зависит от использования	Ко всем
border-bottom	Нет	CSS 1	6.0	Зависит от использования	Ко всем
border-bottom-color	Нет	CSS 2	6.0	Значение цвета, заданное через color	Ко всем
border-bottom-left-radius	Нет	CSS 3	9.0	0	Ко всем
border-bottom-right-radius	Нет	CSS 3	9.0	0	Ко всем
border-bottom-style	Нет	CSS 2	6.0	Нет	Ко всем
border-bottom-width	Нет	CSS 1	6.0	medium	Ко всем
border-collapse	Нет	CSS 2	6.0	separate	К тегу <table> или к элементам, у которых значение display установлено как table или inline-table
border-color	Нет	CSS 1	6.0	Нет	Ко всем

Свойство	Насл.	CSS	IE	По умолчанию	Применяется
border-image	Нет	CSS 3	-	none	Ко всем элементам, за исключением тех, у которых border-collapse задан как collapse
border-left	Нет	CSS 1	6.0	Зависит от использования	Ко всем
border-left-color	Нет	CSS 2	6.0	Нет	Ко всем
border-left-style	Нет	CSS 2	6.0	Нет	Ко всем
border-left-width	Нет	CSS 1	6.0	medium	Ко всем
border-radius	Нет	CSS 3	9.0	0	Ко всем элементам, за исключением таблиц с border-collapse: collapse
border-right	Нет	CSS 1	6.0	Зависит от использования	Ко всем
border-right-color	Нет	CSS 2	6.0	Нет	Ко всем
border-right-style	Нет	CSS 2	6.0	Нет	Ко всем
border-right-width	Нет	CSS 2	6.0	medium	Ко всем
border-spacing	Да	CSS 2	8.0	0	К таблицам
border-style	Нет	CSS 1	6.0	none	Ко всем
border-top	Нет	CSS 1	6.0	Зависит от использования	Ко всем
border-top-color	Нет	CSS 2	6.0	Нет	Ко всем
border-top-left-radius	Нет	CSS 3	9.0	0	Ко всем
border-top-right-radius	Нет	CSS 3	9.0	0	Ко всем
border-top-style	Нет	CSS 2	6.0	Нет	Ко всем
border-top-width	Нет	CSS 1	6.0	medium	Ко всем
border-width	Нет	CSS 1	6.0	medium	Ко всем
bottom	Нет	CSS 2	6.0	auto	Ко всем
box-shadow	Нет	CSS 3	9.0	none	Ко всем
box-sizing	Нет	CSS 3	8.0	content-box	Ко всем
caption-side	Да	CSS 2	8.0	top	К <caption> или ко всем элементам, у которых значение display установлено как table-caption
clear	Нет	CSS 1	6.0	none	К блочным и плавающим элементам
clip	Нет	CSS 2	6.0	auto	К блочным элементам

Продолжение ⇩

Таблица П4.1 *(продолжение)*

Свойство	Насл.	CSS	IE	По умолчанию	Применяется
color	Да	CSS 1	6.0	Зависит от настроек браузера, обычно черный цвет	Ко всем
column-count	Нет	CSS 3	10.0	auto	К блочным элементам (кроме таблиц), ячейкам и элементам, у которых display установлен как inline-block
column-gap	Нет	CSS 3	10.0	normal	К блочным элементам (кроме таблиц), ячейкам и элементам, у которых display установлен как inline-block
column-rule	Нет	CSS 3	10.0	medium none	К блочным элементам (кроме таблиц), ячейкам и элементам, у которых display установлен как inline-block
column-width	Нет	CSS 3	10.0	auto	К блочным элементам (кроме таблиц), ячейкам и элементам, у которых display установлен как inline-block
columns	Нет	CSS 3	10.0	auto	К блочным элементам (кроме таблиц), ячейкам и элементам, у которых display установлен как inline-block
content	Нет	CSS 2	8.0	Пустая строка	Пустая строка
counter-increment	Нет	CSS 2	8.0	none	Ко всем
counter-reset	Нет	CSS 2	8.0	none	Ко всем
cursor	Да	CSS 2	6.0	auto	Ко всем
direction	Да	CSS 2	6.0	ltr	Ко всем
display	Нет	CSS 2	6.0	inline	Ко всем

Свойство	Насл.	CSS	IE	По умолчанию	Применяется
empty-cells	Да	CSS 2	8.0	show	К <td>, <th> или к элементам, у которых display: table-cell
float	Нет	CSS 1	6.0	none	Ко всем элементам (за исключением позиционированных)
font	Да	CSS 1	6.0	Зависит от использования	Ко всем
font-family	Да	CSS 1	6.0	Шрифт, установленный в браузере по умолчанию. Обычно это Times New Roman	Ко всем
font-size	Да	CSS 1	6.0	medium	Ко всем
font-stretch	Да	CSS 2	9.0	normal	Ко всем
font-style	Да	CSS 1	6.0	normal	Ко всем
font-variant	Да	CSS 1	6.0	normal	Ко всем
font-weight	Да	CSS 1	6.0	normal	Ко всем
height	Нет	CSS 1	6.0	auto	К блочным и заменяемым элементам
left	Нет	CSS 2	6.0	auto	Ко всем
letter-spacing	Да	CSS 1	6.0	normal	Ко всем
line-height	Да	CSS 1	6.0	normal	Ко всем
list-style	Да	CSS 1	6.0	Нет	К тегам <dd>, <dt>, , и , а также ко всем элементам, у которых указано display: list-item
list-style-image	Да	CSS 1	6.0	none	К тегам <dd>, <dt>, , и , а также ко всем элементам, у которых указано display: list-item
list-style-position	Да	CSS 1	6.0	outside	К тегам <dd>, <dt>, , и , а также ко всем элементам, у которых указано display: list-item

Продолжение ⇩

Таблица П4.1 *(продолжение)*

Свойство	Насл.	CSS	IE	По умолчанию	Применяется
list-style-type	Да	CSS 1	6.0	disc (для); decimal (для)	К тегам <dd>, <dt>, , и , а также ко всем элементам, у которых указано display: list-item
margin	Нет	CSS 1	6.0	0	Ко всем
margin-bottom	Нет	CSS 1	6.0	0	Ко всем
margin-left	Нет	CSS 1	6.0	0	Ко всем
margin-right	Нет	CSS 1	6.0	0	Ко всем
margin-top	Нет	CSS 1	6.0	0	Ко всем
max-height	Нет	CSS 2	7.0	none	Ко всем элементам, кроме встроенных и таблиц
max-width	Нет	CSS 2	7.0	none	Ко всем элементам, кроме встроенных и таблиц
min-height	Нет	CSS 2	7.0	0	Ко всем элементам, кроме встроенных и таблиц
min-width	Нет	CSS 2	7.0	0	Ко всем элементам, кроме встроенных и таблиц
opacity	Нет	CSS 3	9.0	1	Ко всем
orphans	Да	CSS 2	8.0	2	К блочным элементам
outline	Нет	CSS 2	8.0	Нет	Ко всем
outline-color	Нет	CSS 2	8.0	invert	Ко всем
outline-style	Нет	CSS 2	8.0	none	Ко всем
outline-width	Нет	CSS 2	8.0	medium	Ко всем
overflow	Нет	CSS 2	6.0	visible	К блочным элементам
overflow-x	Нет	CSS 3	6.0	visible	К блочным элементам
overflow-y	Нет	CSS 3	6.0	visible	К блочным элементам
padding	Нет	CSS 1	6.0	0	Ко всем
padding-bottom	Нет	CSS 1	6.0	0	Ко всем
padding-left	Нет	CSS 1	6.0	0	Ко всем
padding-right	Нет	CSS 1	6.0	0	Ко всем
padding-top	Нет	CSS 1	6.0	0	Ко всем

Свойство	Насл.	CSS	IE	По умолчанию	Применяется
page-break-after	Нет	CSS 2	6.0	auto	К блочным элементам
page-break-before	Нет	CSS 2	6.0	auto	К блочным элементам
page-break-inside	Нет	CSS 2	8.0	auto	К блочным элементам
position	Нет	CSS 2	6.0	static	Ко всем элементам, за исключением генерируемых
quotes	Да	CSS 2	8.0	Зависит от браузера, его настроек и операционной системы. Чаще всего используются кавычки вида "/"	Ко всем
resize	Нет	CSS 3	-	none	К <textarea>
right	Нет	CSS 2	6.0	auto	Ко всем
tab-size	Да	CSS 3*	–	8	К блочным элементам
table-layout	Нет	CSS 2	6.0	auto	К тегу <table> или к элементу, у которого значение display установлено как table или inline-table
text-align	Да	CSS 1	6.0	left	Ко всем
text-align-last	Да	CSS 3	6.0	start	К блочным элементам
text-decoration	Нет	CSS 1	6.0	none	Ко всем
text-indent	Да	CSS 1	6.0	0	К блочным элементам
text-overflow	Нет	CSS 3*	6.0	clip	К блочным элементам
text-shadow	Да	CSS 2	10.0	none	Ко всем
text-transform	Да	CSS 1	6.0	none	Ко всем
top	Нет	CSS 2	6.0	auto	Ко всем
unicode-bidi	Нет	CSS 2	6.0	normal	Ко всем
vertical-align	Нет	CSS 1	6.0	baseline	К встроенным элементам или ячейкам таблицы
visibility	Да	CSS 2	6.0	visible	Ко всем

Продолжение ⇩

Таблица П4.1 *(продолжение)*

Свойство	Насл.	CSS	IE	По умолчанию	Применяется
white-space	Да	CSS 1	6.0	normal	К блочным элементам
widows	Да	CSS 2	8.0	2	К блочным элементам
width	Нет	CSS 1	6.0	auto	К блочным элементам
word-spacing	Да	CSS 1	6.0	normal	Ко всем
word-wrap	Да	CSS 3*	6.0	normal	Ко всем
writing-mode	Да	CSS 3	6.0	Нет	Ко всем элементам и генерируемому контенту
z-index	Нет	CSS 2	6.0	auto	К любым позиционированным элементам

* Возможно, данное свойство не будет включено в CSS 3.

Клименко Р. А.

Веб-мастеринг на 100%

Заведующая редакцией	*К. Галицкая*
Руководитель проекта	*Д. Виницкий*
Ведущий редактор	*Е. Каляева*
Литературный редактор	*Е. Каляева*
Художник	*Л. Адуевская*
Корректор	*Е. Павлович*
Верстка	*А. Барцевич*

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2;
95 3005 — литература учебная.

Подписано в печать 24.09.12. Формат 70×100/16. Усл. п. л. 41,280. Тираж 2000. Заказ 0000.

Отпечатано с готовых диапозитивов в ГППО «Псковская областная типография».
180004, Псков, ул. Ротная, 34.

ВАМ НРАВЯТСЯ НАШИ КНИГИ? ЗАРАБАТЫВАЙТЕ ВМЕСТЕ С НАМИ!

У Вас есть свой сайт?
Вы ведете блог?
Регулярно общаетесь на форумах? Интересуетесь литературой, любите рекомендовать хорошие книги и хотели бы стать нашим партнером?
ЭТО ВПОЛНЕ РЕАЛЬНО!

СТАНЬТЕ УЧАСТНИКОМ ПАРТНЕРСКОЙ ПРОГРАММЫ ИЗДАТЕЛЬСТВА «ПИТЕР»!

Зарегистрируйтесь на нашем сайте в качестве партнера по адресу ***www.piter.com/ePartners***

Получите свой персональный уникальный номер партнера

Выбирайте книги на сайте ***www.piter.com,*** *размещайте информацию о них на своем сайте, в блоге или на форуме и добавляйте в текст ссылки на эти книги (на сайт* ***www.piter.com****)*

ВНИМАНИЕ! *В каждую ссылку необходимо добавить свой персональный уникальный номер партнера.*

С этого момента получайте 10% от стоимости каждой покупки, которую совершит клиент, придя в интернет-магазин «Питер» по ссылке с Вашим партнерским номером. А если покупатель приобрел не только эту книгу, но и другие издания, Вы получаете дополнительно по 5% от стоимости каждой книги.

Деньги с виртуального счета Вы можете потратить на покупку книг в интернет-магазине издательства «Питер», а также, если сумма будет больше 500 рублей, перевести их на кошелек в системе Яндекс.Деньги или Web.Money.

Пример партнерской ссылки:
http://www.piter.com/book.phtml?978538800282 – обычная ссылка
http://www.piter.com/book.phtml?978538800282**&refer=0000** – партнерская ссылка, где 0000 – это ваш уникальный партнерский номер

Подробно о Партнерской программе ИД «Питер» читайте на сайте WWW.PITER.COM

ИЗДАТЕЛЬСКИЙ ДОМ
ПИТЕР®
WWW.PITER.COM

**ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
предлагают профессиональную и популярную литературу по различным направлениям: история и публицистика, экономика и финансы, менеджмент и маркетинг, компьютерные технологии, медицина и психология.**

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: voronej@piter.com

Екатеринбург: ул. Бебеля, д. 11а
тел./факс: (343) 378-98-41, 378-98-42; e-mail: office@ekat.piter.com

Нижний Новгород: тел.: 8 960 187-85-50; e-mail: nnovgorod@piter.com

Новосибирск: Комбинатский пер., д. 3
тел./факс: (383) 279-73-92; e-mail: sib@nsk.piter.com

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 229-68-09; e-mail: samara@piter.com

УКРАИНА

Киев: Московский пр., д. 6, корп. 1, офис 33
тел./факс: (044) 490-35-69, 490-35-68; e-mail: office@kiev.piter.com

Харьков: ул. Суздальские ряды, д. 12, офис 10
тел./факс: (057) 7584145, +38 067 545-55-64; e-mail: piter@kharkov.piter.com

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163
тел./факс: (517) 208-80-01, 208-81-25; e-mail: minsk@piter.com

---

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок
тел./факс: (812) 703-73-73; e-mail: spb@piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов
тел./факс издательства: (812) 703-73-72, (495) 974-34-50

Заказ книг для вузов и библиотек
тел./факс: (812) 703-73-73, доб. 6250; e-mail: uchebnik@piter.com

---

Заказ книг по почте: на сайте www.piter.com; по тел.: (812) 703-73-74, доб. 6225
