Пётр Ташков

на 100%

ВЕБ-МАСТЕРИНГ

HTML, CSS, JavaScript, PHP, CMS, AJAX, раскрутка



Эта книга поможет вам:

- проводить тонкую настройку сайта
- моделировать динамические веб-страницы
- создавать графические элементы сайтов
- сопровождать и продвигать сайты в Интернете

ПИТЕР

Ташков П. А.

Веб-мастеринг на 100 %: HTML, CSS, JavaScript, PHP, CMS, AJAX, раскрутка

Серия «На 100 %»

Заведующий редакцией	*А. Громаковский*
Ведущий редактор	*Е. Каляева*
Литературный редактор	*Ю. Кравцова*
Художник	*Л. Адуевская*
Корректоры	*Н. Терех, Ю. Цеханович*
Верстка	*О. Махлина*

ББК 32.988.02-018
УДК 004.72

Ташков П. А.

Т25 Веб-мастеринг на 100 %: HTML, CSS, JavaScript, PHP, CMS, AJAX, раскрутка. — СПб.: Питер, 2010. — 512 с.: ил. — (Серия «На 100%»).

ISBN 978-5-49807-826-7

Эта книга адресуется всем, кто хочет научиться быстро и эффективно решать задачи веб-мастеринга и стать специалистом по созданию веб-порталов профессионального уровня. В книге описываются все средства, которые входят в арсенал настоящего веб-мастера, — языки HTML, JavaScript, PHP, AJAX и Perl, дается введение в базы данных MySQL, а также излагаются основные приемы работы с системами управления содержимым сайта (CMS) Joomla! и Drupal. С помощью этих средств вы сможете создавать полноценные веб-сайты любого назначения, например интерактивные интернет-магазины или непрерывно обновляемые новостные порталы. Наконец, рассказывается о проверке работоспособности созданного веб-сайта, излагаются методы и рекомендации по размещению сайта на платных и бесплатных хостингах, способы его раскрутки и регистрации в поисковиках и каталогах. Прочитав эту книгу, вы станете настоящим веб-мастером, готовым к работе над любыми проектами.

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-5-49807-826-7

© ООО Издательство «Питер», 2010

ООО «Лидер», 194044, Санкт-Петербург, Б. Сампсониевский пр., 29а.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Подписано в печать 11.06.10. Формат 70×100/16. Усл. п. л. 45,15. Тираж 3000. Заказ 0000.

Отпечатано по технологии CtP в ОАО «Печатный двор» им. А. М. Горького.
197110, Санкт-Петербург, Чкаловский пр., 15.

Краткое содержание

Введение 10
Соглашения, принятые в книге 12
От главы коллектива авторов 13
От издательства 13

Глава 1. HTML и CSS 14

Глава 2. Работа с JavaScript 141

Глава 3. Apache, Perl и MySQL 262

Глава 4. Знакомство с основными системами управления содержимым сайта (CMS) — Joomla! и Drupal 315

Глава 5. Создание серверных сценариев на языке PHP 6 364

Глава 6. Веб-приложения на языке PHP 397

Глава 7. Технология AJAX 427

Глава 8. Проверка работоспособности сайта 451

Глава 9. Размещение сайта и выбор хостинга 469

Глава 10. Раскрутка сайта 481

Оглавление

Введение 10
Соглашения, принятые в книге 12
Шрифты 12
Нумерованные и маркированные списки 12
Врезки 12
Листинги 13
От главы коллектива авторов 13
От издательства 13

Глава 1. HTML и CSS 14
Структура языка HTML 15
Тег 15
Атрибут 16
Структура HTML-документа 16
Тег ‹!DOCTYPE› 16
Тег ‹HTML› 17
Тег ‹HEAD› 18
Тег ‹TITLE› 19
Тег ‹META› 21
Тег ‹STYLE› 27
Тег ‹LINK› 32
Тег ‹SCRIPT› 33
Тег ‹BASE› 34
Тег ‹BODY› 36
Комментарии 39
Ввод и оформление текста 39
Заголовки 40
Абзацы 42
Обрывы строк 43
Списки 46
Ссылки 55
Внешние ссылки 55
Внутренние ссылки 56
Общие моменты при создании внешних и внутренних ссылок 58

Форматирование текста 60
Теги логического форматирования 60
Теги физического форматирования 65
Теги форматирования больших блоков текста 69
Вложение тегов 73
Работа с таблицами 73
Что такое таблица 74
Создание тела таблицы 74
Ячейки таблицы 75
Границы таблицы 76
Ширина и высота таблицы и ячеек 79
Группировка строк и столбцов 82
Выравнивание таблицы и содержимого ячеек 83
Объединение ячеек таблицы 84
Установка фона или рисунка 86
Создание вложенных таблиц 87
Изображения 88
Встраивание изображений 88
Размер изображения 89
Выравнивание изображения 90
Рамка вокруг изображения 93
Альтернативный текст 94
Формы 96
Тег ‹FORM› 97
Создание полей формы 99
Общие принципы CSS 109
Селекторы 110
Псевдоклассы и псевдоэлементы 115
Форматирование текста средствами CSS 120
Выделение текста цветом 120
Шрифты 122
Форматирование текста 125
Оформление документа CSS 127
Фон 127
Поля и отступы 129
Границы 131
Работа с блоками 133
Резюме 140

Глава 2. Работа с JavaScript 141
Область применения JavaScript 142
Создание сценариев 143
Встраивание сценариев 144
Ваш первый сценарий 148
Синтаксис и основные средства языка JavaScript 150
Работа с информацией 150
Переменные и типы данных 153
Выражения 162
Условия и циклы 168
Функции 178
Встроенные объекты JavaScript 184
Объектная модель 191
Веб-браузер и HTML-документ 191
Родительские и дочерние объекты 192
Объекты веб-браузера 194
Объектная модель документа 209
Работа с формами 220
Форма как объект 220
Свойства, методы и события формы 226
Примеры сценариев 231
Динамический HTML 237
Управление текстом 238
Графика на веб-страницах 246
Слои 252
Доступ к стилям с помощью JavaScript 256
Пример создания динамической игры 257
Резюме 261

Глава 3. Apache, Perl и MySQL 262
Сервер Apache 263
Для чего нужен Apache 263
Установка Apache 264
Основные настройки Apache 267
Язык Perl 270
Установка Perl и подключение его к Apache 270
Основы языка и синтаксис 274

Примеры использования Perl для взаимодействия с пользователем 278
Механизм поиска 282
Получение информации о посетителях 284
СУБД MySQL 286
Введение в СУБД MySQL 286
Установка и настройка программы.
Инструменты графического интерфейса. Командная строка 290
SQL-команды: синтаксис команд, примеры запросов
из командной строки, создание базы данных 298
Работа с таблицами: добавление и удаление записей,
выбор с условиями 302
Операторы языка SQL 305
Настройка взаимодействия MySQL и Perl 307
Установка проекта «Денвер» и работа с ним 307
Обращение к базе SQL из программы Perl 311
Резюме 313

Глава 4. Знакомство с основными системами управления содержимым сайта (CMS) — Joomla! и Drupal 315
CMS Joomla! 316
Установка 318
Администрирование 326
Русификация 327
Пользователи и группы 329
Компоненты 332
Модули 335
Шаблоны 340
Плагины 342
CMS Drupal 344
Установка 345
Возможные проблемы при установке 349
Администрирование 350
Резюме 362

Глава 5. Создание серверных сценариев на языке PHP 6 364
Создание и запуск первого сценария PHP 365
Комментарии внутри кода PHP 368
Переменные 369

Типы данных PHP 370
Boolean 371
String 371
Integer 371
Float 372
Array 372
Resource 374
NULL 374
Константы в PHP 374
Операторы PHP 376
Операторы присвоения 376
Математические операторы 376
Комбинированные операции присваивания 378
Строковые операторы 378
Операторы инкремента и декремента 379
Битовые операторы 380
Операторы сравнения 380
Логические операторы 381
Приоритеты операторов 382
Условные операторы 383
Оператор switch 385
Операторы циклов 387
Работа с функциями в PHP 391
Синтаксис функций PHP 392
Передача аргументов 393
Параметры по умолчанию 393
Видимость переменных 394
Резюме 396

Глава 6. Веб-приложения на языке PHP 397
Работа с формами 399
Загрузка и обработка файлов 406
Подключение и работа с базами данных MySQL 410
Подключение к серверу MySQL и выбор базы данных 410
Работа с базой данных MySQL 412
Сеансы 418
Работа с cookie 420

Отправка почты 423
Резюме 426

Глава 7. Технология AJAX 427
Как работает AJAX 429
Знакомство с XML 430
Первое веб-приложение с использованием AJAX 432
Работа с данными XML 441
Работа с MySQL 445
Резюме 450

Глава 8. Проверка работоспособности сайта 451
Проверка актуальности ссылок на сайте 452
Рабочее окно программы Web Link Validator 453
Проверка отдельных ссылок и закладок 454
Проверка ссылок всего веб-сайта 457
Оптимизация кода веб-страниц 463
Резюме 468

Глава 9. Размещение сайта и выбор хостинга 469
Выбор хостинга 470
Обзор популярных хостингов 471
Сервис «Джино» 474
Загрузка данных на сайт 479
Резюме 480

Глава 10. Раскрутка сайта 481
Поисковые машины 483
Каталоги 490
Рейтинги 495
Баннеры 500
Элементы привлечения посетителей 505
Гостевые книги 506
Форумы 506
Чаты 507
Почтовые рассылки 508
Ссылки 509
Резюме 512

Введение

Эта книга написана для тех, кто хочет создать собственный сайт в Интернете, отвечающий всем требованиям к профессиональному веб-порталу. Такой сайт должен быть красивым, удобным в использовании и навигации, свободным от всякого рода проблем, мешающих его просмотру. К тому же он должен попадать в начало списка с результатами поиска с помощью крупнейших поисковиков, то есть сайт должен быть посещаемым. Использования существующих программ веб-дизайна может оказаться недостаточно: тонкая настройка содержимого веб-страниц и их внутренней организации может быть выполнена только путем самостоятельного написания кода HTML-страницы и встроенных в них сценариев. Для этого вам нужно изучить язык гипертекстовой разметки HTML, научиться создавать сценарии на языках JavaScript и PHP, уметь работать с каскадными таблицами стилей CSS, а также владеть принципами раскрутки сайтов. Обо всем этом вы узнаете из книги, которую сейчас держите в руках. С ее помощью вы научитесь создавать эффективно функционирующие, красиво оформленные, рейтинговые сайты профессионального уровня.

В главе 1 вы освоите базовые средства языков HTML и CSS, научитесь создавать документы HTML, оформленные согласно всем требованиям, предъявляемым к статическим веб-страницам. Особое внимание в этой главе уделяется языку CSS, который ныне является основным инструментом оформления содержимого сайтов.

В главе 2 излагается техника построения динамических веб-страниц с помощью внедрения в код HTML сценариев JavaScript, которые привносят интерактивность в содержимое сайта, необходимую для его полноценного функционирования.

В главе 3 вы научитесь работать с мощными средствами создания веб-приложений — с сервером Apache, языком Perl и базами данных MySQL. Вы освоите установку этих средств на компьютер, запуск сервера Apache, подключение к нему базы данных MySQL и технику обращения из сценариев Perl к базе MySQL с помощью Apache и запросов на языке SQL. Таким образом, вы сможете создавать веб-порталы, предоставляющие своим посетителям различные ресурсы по запросам к базе данных. Эти веб-порталы удобны тем, что позволяют пользователям быстро находить необходимую информацию.

В главе 4 описывается работа с «движками» сайтов — специальными программами по управлению содержимым сайта (системами CMS). Такие системы стали весьма популярным средством работы с сайтами, содержимое которых требуется непрерывно обновлять и редактировать (сайты новостей, интернет-магазины). Вы осво-

ите работу с наиболее популярными программами CMS — Drupal и Joomla!, научитесь устанавливать их на компьютер, настраивать их работу и администрировать.

В главе 5 описывается язык PHP, ставший основным средством создания веб-приложений коммерческого плана. Рассматриваются основные конструкции языка: переменные, циклы, сравнения, присвоения, а также использование функций.

В главе 6 используются сведения, приведенные в главе 5, для построения примеров веб-приложений, реализующих работу с формами, хранение информации в базах данных MySQL, отправку почты и сохранение файлов Cookie.

В главе 7 описывается технология AJAX, позволяющая организовывать асинхронные вызовы серверного сценария для оптимизации загрузки данных на клиентское приложение и улучшения качества взаимодействия пользователя с различными сервисами Интернета. AJAX широко используется в современных веб-приложениях, реализующих функции интернет-магазинов, просмотр графических данных большого объема, поиск информации в Интернете. Рассмотрен общий принцип организации асинхронных запросов, применение AJAX совместно со сценариями PHP, работы с базами данных MySQL.

В главе 8 описываются методы поиска ошибок на веб-сайтах. Наверное, многие пользователи посещали сайты со ссылками, которые никуда не ведут, с пустыми рамками рисунков и неработающими элементами навигации. Такие ошибки могут просто свести на нет эффективность работы сайта. Однако найти их не всегда легко. Для решения данной задачи были разработаны специальные программы диагностики и поиска ошибок, которые и рассматриваются в этой главе.

В главе 9 описываются методы размещения сайтов на различных ресурсах Интернета, которых сейчас предлагается великое множество. Выбору оптимального варианта и методов работы с предоставляемыми средствами выгрузки и управления сайтом как раз и посвящается эта глава.

В главе 10 рассматриваются принципы раскрутки сайта, без знания которой вы не сможете получить никакой отдачи от разработанного веб-сайта. Сейчас в Интернете размещаются сотни миллионов сайтов, и занять высокое место в рейтингах поисковиков и каталогов очень непросто. Для достижения этой цели существуют специальные методики раскрутки, которые и рассматриваются в этой главе.

Освоив технику разработки веб-сайтов, вы станете настоящим веб-мастером, которому по плечу любые задачи, решаемые специалистами IT-фирмы при разработке веб-сайтов любого назначения.

Соглашения, принятые в книге

Чтобы ваша работа с книгой была более эффективной, в ней используются специальные приемы выделения важных фрагментов текста.

Шрифты

Для выделения важных фрагментов текста применяются следующие шрифты:

- *курсивом* выделены определения и термины;
- шрифтом для элементов интерфейса обозначен текст, который можно увидеть на экране компьютера (например, заголовки диалоговых окон, названия команд меню, кнопок, параметров, переключателей, имена папок и т. п.);
- `шрифтом для команд` выделены имена файлов, команды и другой текст, который нужно вводить с клавиатуры.

Нумерованные и маркированные списки

Чтобы описать несколько однотипных объектов, дать советы для решения какой-нибудь проблемы или предложить варианты выполнения определенного действия, в книге применяются *маркированные списки*. Из рекомендаций, оформленных маркированным списком, вы можете выбрать одну или несколько в произвольном порядке.

Нумерованным списком оформлены пошаговые инструкции для выполнения каких-либо действий. Для достижения результата нужно последовательно проделать описанные шаги, не пропуская их и не переставляя местами.

Врезки

Для особого выделения одного или нескольких абзацев текста в книге используются врезки, которые имеют следующее назначение.



ПРИМЕЧАНИЕ

Такие врезки содержат уточнения, дополнения или комментарии, которые относятся к излагаемому материалу.



ВНИМАНИЕ

В данных врезках приводится информация, на которую следует обратить особое внимание ввиду ее важности. Наиболее часто подобные врезки применяются для предупреждения о возможных ошибочных действиях пользователя.

СОВЕТ



В этих врезках описаны эффективные приемы выполнения практических действий.

Листинги

В листингах приводятся коды программ, необходимые для выполнения некоторых действий при создании сайтов в Интернете.

От главы коллектива авторов

Высказать замечания и пожелания, задать вопросы по этой книге вы можете по адресу AlexanderZhadaev@sigmaplus.mcdir.ru или посетив нашу домашнюю страничку www.sigmaplus.mcdir.ru (там вы найдете также дополнительные материалы по книге, сможете принять участие в форуме или пообщаться в чате).

Александр Жадаев

От издательства

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

Мы будем рады узнать ваше мнение!

Все приведенные в книге листинги (коды программ) можно загрузить с сайта издательства «Питер» по адресу http://www.piter.com/download.

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

Глава 1

HTML и CSS

Структура языка HTML

Структура HTML-документа

Ввод и оформление текста

Ссылки

Форматирование текста

Работа с таблицами

Изображения

Формы

Общие принципы CSS

Форматирование текста средствами CSS

Оформление документа CSS

Резюме

В этой главе излагается, что такое язык HTML и как с его помощью можно создавать простые веб-страницы. Вы также узнаете об огромных возможностях по управлению содержимым HTML-документов, которые предоставляет CSS.

Структура языка HTML

Для создания страниц в Интернете служит язык гипертекстовой разметки (HTML — HiperText Markup Language). Каждый сайт, конечно, индивидуален, однако существуют общие правила построения HTML-страниц, следовать которым нужно обязательно.

Существуют общие правила записи HTML-документов и общие понятия, используемые при создании сайтов. Вначале рассмотрим понятия.

Тег

Основой языка HTML является тег. Этот элемент несет в себе определенную информацию, может описывать документ в общем или способ форматирования текста. Все содержимое веб-страниц задается и редактируется с помощью тегов.

Тег помещается в угловые скобки `<ТЕГ>`. Иногда необходимо задать парный закрывающий тег, который пишется так: `</ТЕГ>`. В некоторых случаях закрывающий тег вообще не требуется, а иногда его можно пропустить. Однако для корректной обработки документа рекомендуется всегда использовать закрывающий тег. Информация, заключенная между открывающимся и закрывающимся тегом, называется его *контейнером*.

Кратко функции тегов можно описать так: открывающий тег включает форматирование, а закрывающий — выключает его.

Примером использования закрывающегося тега является тег `<P>`, который обозначает абзац: `<P>Текст абзаца</P>`, при этом закрывающий тег не является обязательным, хотя желателен. Закрывающего тега не требует тег `<IMG>`, который добавляет картинку на сайт.

Тег нужен для того, чтобы сообщить браузеру, какой элемент должен быть на месте тега или какого типа информация заключена внутри тега. Кроме того, теги показывают, как следует отображать эту информацию.

Атрибут

Атрибуты сообщают браузеру, каким образом должен отображаться тот или иной элемент страницы. Атрибуты позволяют сделать более разнообразным внешний вид информации, добавляемой с помощью одинаковых тегов. Следует отметить, что в некоторых случаях использование тега без атрибутов не дает результатов.

Например, в одном абзаце нужно выровнять текст по левому краю, а в другом — по правому. Чтобы задать выравнивание абзаца, используется атрибут `ALIGN` тега `<P>`.

```
<P align="left">Выравнивание по левому краю</P>
<P align="right">Выравнивание по правому краю</P>
```

Значение атрибута задается после знака равенства, как правило, в кавычках. Значения атрибутов различаются в зависимости от тега.

Теги и их атрибуты являются основой языка HTML, но для правильного отображения страницы в браузерах важно также по строгим правилам правильно создать структуру документа. Существуют теги, без которых HTML-документ не может обойтись, потому что именно они определяют его структуру.

Структура HTML-документа

Для создания структуры документа и хранения служебной информации о нем предусмотрено много тегов. Рассмотрим их по порядку.

Тег <!DOCTYPE>

Этот тег записывается в начале HTML-файла. Он описывает создаваемый документ и предоставляет основную информацию для браузера.

В листинге 1.1 приводится пример заполнения информацией тега `<!DOCTYPE>`. Для большинства сайтов текст внутри этого тега будет таким.

Листинг 1.1. Описание документа в теге <DOCTYPE>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Эта строчка дает браузеру общую информацию об HTML-документе. Вряд ли вам придется заполнять эту строку иначе, однако рассмотрим ее по частям и выясним, какую информацию о документе несут данные из тега <!DOCTYPE>.

- `html` — показывает, что для создания документа используется язык HTML.
- `PUBLIC` — указывает на то, что при создании документа использовалась общепринятая версия HTML.
- `"-//W3C//DTD XHTML 1.0 Transitional//EN"` — отражает, какая версия языка используется, а также какие элементы и атрибуты можно применять.
- `"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"` — адрес URL-документа, в котором описываются элементы и атрибуты, используемые в соответствии с версией языка. В этом теге содержится общая служебная информация о странице. Он не влияет на то, как страница будет выглядеть на экране, более того, без него страница не сможет нормально загрузиться. Использование этого тега подразумевается стандартом HTML.

После того как мы ввели общую информацию о странице, надо разобраться с ее структурой.

Тег ‹HTML›

Главным структурным тегом страницы является тег <HTML>, все содержимое страницы записывается внутри его контейнера. Все, что находится за пределами тегов <HTML> и </HTML>, не воспринимается браузером как код HTML и никак им не обрабатывается.

В листинге 1.2 представлен пример использования тега <HTML>.

Листинг 1.2. Применение тега ‹html›

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<!-- Содержимое документа -->
</html>
```

Тег <HTML> обязательно должен использоваться с закрывающим тегом, который указывает на окончание документа.

В теге <HTML> могут применяться следующие атрибуты:

- `xmlns` — задает пространство имен для xml-документа, поскольку наш документ является разновидностью xml-документа;

- `lang` — задает язык документа;
- `xml:lang` — тоже задает язык документа;
- `title` — задает всплывающую подсказку для страницы.

В листинге 1.3 приведен пример использования тега <HTML> с атрибутами.

Листинг 1.3. Тег ‹HTML› с атрибутами

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" title="Моя страничка" lang="ru"
xml:lang="ru">
<!-- Содержимое документа -->
</html>
```

На рис. 1.1 представлен результат использования атрибутов тега <HTML>.

На рис. 1.1 вы видите всплывающую подсказку, которая появляется при наведении указателя мыши на страницу, открытую в браузере. Эта подсказка кратко описывает сайт. Остальные атрибуты тега не влияют на отображение страницы.

После того как вы создали блок для документа, внутри него можно сформировать заглавную часть документа.

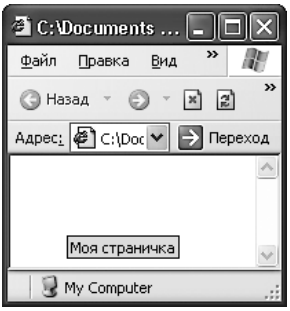


Рис. 1.1. Использование атрибутов тега ‹HTML›

Тег ‹HEAD›

Тег <HEAD> используется для служебных целей, введенная в нем информация не отображается в окне браузера, однако он содержит множество данных, которые указывают браузеру, как следует обрабатывать страницы.

Тег <HEAD> требует закрывающий тег </HEAD>. Между ними располагаются другие теги, которые несут служебную информацию о странице. Теги, находящиеся внутри блока тегов <HEAD>, играют очень важную роль: данные, которые в них содержатся, помогают браузеру в обработке страницы, а поисковым системам — в индексации документа.

Теги, которые можно использовать внутри блока тегов <HEAD>, представлены в табл. 1.1.

Теги в табл. 1.1 описываются кратко. Некоторые из них мы рассмотрим далее более подробно.

Таблица 1.1. Теги, используемые внутри тегов ‹head›

Тег	Описание
<title>	Задает заголовок окна, требует закрывающего тега
<meta>	Задает различную служебную информацию
<script>	Позволяет добавлять сценарии
<link>	Задает ссылку на таблицы стилей
<style>	Позволяет добавить стили для страницы
<base>	Задает базовый адрес документа
<basefont>	Задает для страницы шрифт по умолчанию
<bgsound>	Фоновый звук на странице

Тег <HEAD> не имеет никаких атрибутов, он выполняет исключительно структурирующие функции, показывая обработчику часть документа, в которой хранится служебная информация.

В листинге 1.4 представлен пример записи тега <HEAD>.

Листинг 1.4. Тег ‹HEAD›

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<!--Содержимое заголовка-->
</head>
<!--Содержимое документа-->
</html>
```

Рассмотрим подробнее теги, которые располагаются внутри блока тегов <HEAD>. Как можно увидеть из их краткого описания в табл. 1.1, они содержат информацию и данные, единые для всей страницы.

Начнем с тега, определяющего заголовок страницы.

Тег ‹TITLE›

Тег <TITLE> задает название страницы, которое будет отображаться в строке с названием приложения окна браузера. Этот тег требует закрывающий тег </TITLE>. Текст, содержащийся между этими тегами, отображается в качестве заголовка страницы.

Помимо основной функции (рассказать посетителю, какую информацию содержит страница), этот тег решает несколько косвенных задач. Например, при сохранении страницы в разделе «Избранное» в качестве названия ссылки будет использоваться текст, записанный в теге <TITLE>. Некоторые поисковые системы используют текст, содержащийся в этом теге, для поиска и выводят его в качестве заголовка результата поиска. Следовательно, корректное заглавие может привлечь внимание посетителей к сайту.

ПРИМЕЧАНИЕ



По тексту заголовка пользователь получает дополнительную информацию: что это за сайт, на котором он находится, и как называется текущая страница. Не следует думать, что достаточно в документе указать логотип сайта и проигнорировать заголовок: посетитель может свернуть окно, и тогда заголовок будет отображаться на кнопке панели задач. По нему можно будет легко сориентироваться, с каким сайтом работать.

Большинство браузеров поддерживают возможность сохранения веб-страницы на компьютер. В этом случае имя сохраненного файла совпадает с названием заголовка документа. Если в тексте заголовка содержатся символы, недопустимые в имени файла (\ / : * ? " ‹ › |), то они проигнорируются или заменятся дозволенными символами.

При сохранении в разделе браузера «Избранное» адрес текущей страницы с ее заголовком помещается в список ссылок. Как правило, этот список хранится в виде отдельных файлов, потому к их именам также относится вышеописанное правило.

В листинге 1.5 показан пример использования тега <TITLE>.

Листинг 1.5. Использование тега ‹TITLE›

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Пример создания заголовка страницы</title>
</head>
<!--Содержимое документа-->
</html>
```

На рис. 1.2 представлен результат выполнения листинга 1.5: текст заголовка отображается в заголовке окна.

Мы определились с заголовком новой страницы, теперь посмотрим, какая служебная информация может содержаться внутри тега <HEAD>.

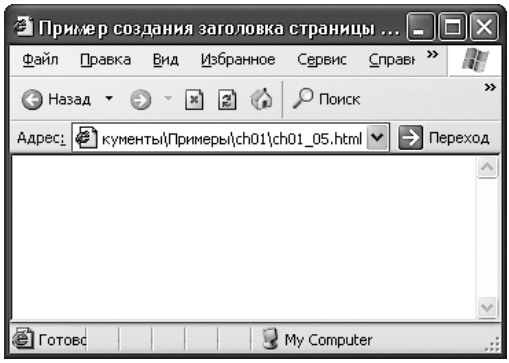


Рис. 1.2. Вид заголовка страницы

Тег <META>

Тег <META> используется для хранения дополнительной информации о странице. Эту информацию используют браузеры для обработки страницы, а поисковые системы — для ее индексации.

В блоке <HEAD> можно использовать несколько тегов <META>, потому что в зависимости от используемых атрибутов они могут нести разную информацию.

Рассмотрим атрибуты тега <META>.

Атрибут CONTENT используется вместе с атрибутами HTTP-EQUIV и NAME. Он задает значение свойства, которое указывается с их помощью.

Атрибут HTTP-EQUIV устанавливает служебные свойства документа, например его кодировку, дает браузеру информацию о том, как нужно обрабатывать документ.

В табл. 1.2 представлены возможные значения атрибута HTTP-EQUIV.

Таблица 1.2. Возможные значения атрибута HTTP-EQUIV

Значение атрибута	Описание
Content-Type	Определяет тип содержимого документа. Этот параметр желательно указывать всегда
Expires	Задает время действия документа. После даты, указанной для этого свойства, документ считается устаревшим
Pragma	Определяет тип кэширования вашего документа, то есть можно запретить браузеру сохранять страницу в кэш

Продолжение

Таблица 1.2 *(продолжение)*

Значение атрибута	Описание
Refresh	Позволяет задать параметры автоматической загрузки документа в то же окно браузера, что и текущий документ
Content-Language	Задает язык содержимого, является аналогом атрибута lang тега html
Content-Script-Type	Типы сценариев, используемых на сайте
Content-Style-Type	Задает типы таблиц стилей

Рассмотрим подробнее применение описанных выше атрибутов.

В листинге 1.6 представлен пример того, как можно задать свойства обработки страницы с помощью атрибута `HTTP-EQUIV`.

Листинг 1.6. Применение тега HTTP-EQUIV

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" title="Моя страничка" lang="ru"
xml:lang="ru">
<head>
<title>Использование тега META</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251"
/>
<meta http-equiv="refresh" content="10; URL=http://www.test.ru" />
<meta http-equiv="pragma" content="no-cache" />
<meta http-equiv="expires" content="Sun, Oct 2010 10:09:00 GMT+3" />
</head>
</html>
```

Как видно из этого примера, значение атрибута `HTTP-EQUIV` указывает на переменную, значение которой указано с помощью атрибута `CONTENT`.

Значение атрибута `Content-Type`, использованное в примере, будет одинаковым для всех сайтов в кириллической кодировке. Рекомендуется всегда указывать его, иначе браузер будет отображать текст на вашей странице некорректно.

Применение параметра `Refresh` полезно в случае, когда страницу перенесли в другое место или когда у страницы много адресов. В таком случае вместо создания множества сайтов по разным адресам можно организовать переброску посетителя на основной сайт. Например, в листинге 1.6 задается загрузка сайта www.test.ru через 10 с после загрузки страницы.



СОВЕТ

Не все браузеры поддерживают автоматическую пересылку, поэтому на всякий случай на страницу, с которой идет переадресация, надо оставлять текстовую ссылку на страницу переадресации.

Рассмотрим возможные значения атрибута `Content` для каждого из представленных свойств (табл. 1.3).

Таблица 1.3. Значения атрибута CONTENT для различных значений атрибута HTTP-EQUIV

Атрибут http-equiv	Значения атрибута content	Описание
Content-Type	ISO-8859-1	Latin-1
	Windows-1251	Кириллица (Windows)
	KOI8-r	Кириллица (КОИ8-Р)
	cp866	Кириллица (DOS)
	Windows-1252	Западная Европа (Windows)
	Windows-1250	Центральная Европа (Windows)
Expires	Sun, Oct 2010 10:09:00 GMT+3	Дата должна указываться в стандарте [RFC850] на английском языке
Pragma	no-cache	Не кэшировать, можно использовать, если страница получается в результате работы сценария, тогда нет смысла оставлять ее в кэше
Refresh	время (в секундах)	Время до автоматической перезагрузки страницы
	время (в секундах), URL	Время до автоматической перезагрузки страницы и адрес страницы, которая по истечении этого срока загрузится в текущее окно
Content-Language	de	Немецкий
	el	Греческий
	en	Английский
	en-GB	Английский, британский
	en-US	Английский, американский
	en-cockney	Английский, кокни
	es	Испанский
	fr	Французский
	it	Итальянский
	i-navajo	Навахо
	ja	Японский
	he	Иврит

Продолжение ⇩

Таблица 1.3 *(продолжение)*

Атрибут http-equiv	Значения атрибута content	Описание
Content-Language	nl	Голландский
	ru	Русский
	pt	Португальский
	zh	Китайский
Content-Script-Type	text/javascript	JavaScript
	text/perlscript	PerlScript
	text/tcl	TCL
	text/vbscript	VBScript
Content-Style-Type	text/css	Язык таблиц стилей CSS

В табл. 1.3 приводится большинство значений атрибута CONTENT, которые могут вам пригодиться. Поначалу вы можете использовать лишь свойство `Content-Type`, так как его нужно указывать обязательно, а остальные параметры включать по мере необходимости.

Атрибут NAME, так же как и атрибут `HTTP-EQUIV`, содержит служебную информацию о документе, однако в нем записывается информация другого плана, например данные об авторе и содержимом документа. Эти данные не влияют на обработку документа браузером, однако предоставляют информацию для поисковых систем.

В табл. 1.4 приводятся возможные значения атрибута NAME.

Таблица 1.4. Возможные значения атрибута NAME

Значение атрибута	Описание
Keyword	Задает ключевые слова, используемые на странице. По этим словам осуществляют поиск поисковые системы
Description	Описание содержимого документа
Author	Задает автора документа
Document-State	Задает необходимость переиндексации страницы
Resource-Type	Задает тип ресурса. Возможные типы мы рассмотрим далее, однако надо учитывать, что если тип документа отличается от «Document», то поисковые системы не будут его индексировать
Revisit	Период переиндексации документа (в днях)
Robots	Указания для роботов и поисковых машин
Subject	Задает тематику документа для поисковых машин
URL	Пересылает робота индексации на другую страницу (например, если у вас несколько одинаковых сайтов с разными адресами, а необходимо, чтобы проиндексирован был только главный)

Как видно из табл. 1.4, большинство атрибутов отвечает за индексацию страницы в поисковиках. Это очень важный момент, когда вы создаете сайт: вы хотите сделать его посещаемым, и в этом помогает тег <META>.

Параметры, задаваемые в теге <META>, помогают поисковым роботам проассоциировать ваш сайт с определенной тематикой. Когда пользователь введет в поисковой строке слово, заданное в качестве ключевого слова для вашей страницы, среди результатов отобразится ссылка на нее.

ПРИМЕЧАНИЕ



Роботы — это специальные программы, которые обращаются к различным сайтам и заносят их содержимое в базы поисковых сайтов. У каждого поисковика свои программы-роботы и своя логика поиска и хранения данных в базах, поэтому часто одинаковые запросы в разных поисковых системах дают разные результаты.

В последние годы роботы игнорируют теги META или, точнее, просто «принимают их к сведению». Они не влияют на ранжирование сайтов.

В листинге 1.7 приводится пример использования параметров тега <META> для управления индексацией страницы.

Листинг 1.7. Параметры тега ‹META› для индексации в поисковиках

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" title="Моя страничка" lang="ru"
xml:lang="ru">
<head>
<title>Использование тега META</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<meta http-equiv="pragma" content="no-cache" />
<meta http-equiv="expires" content="Sun, 10 Oct 2010 10:09:00 GMT+3" />
<meta name="author" content="Автор сайта – Вася Пупкин" />
<meta name="description" content="Использование тега <meta> для индексации
сайта в поисковых системах" />
<meta name="document-state" content="Dynamic" />
<meta name="keywords" content="тег, мета, индексация, поиск" lang="ru" />
<meta name="keywords" content="tags, meta, index, search" lang="en" />
<meta name="Resource-Type" content="Document" />
<meta name="Revisit" content="2" />
<meta name="Robots" content="noindex,follow" />
```

```
<meta name="URL" content="http://www.test.ru" />
</head>
</html>
```



СОВЕТ

В качестве ключевых слов надо использовать только те слова, которые имеют прямое отношение к тексту, расположенному на странице. Не стоит повторять ключевые слова по нескольку раз. Если используете словосочетания, не применяйте в них союзы и предлоги: они не несут смысловой нагрузки, а только занимают место. Обычно поисковые роботы распознают первые 200–250 символов, определенные в параметре keyword, порядок слов зависит от их важности. Наиболее важные слова располагаются в начале списка.

При подборе ключевых слов постарайтесь предположить, какие сочетания и слова будет использовать пользователь при поиске страницы вашей тематики, и используйте их в ключевых словах. Для разных страниц сайта, даже если их тематика одинаковая, используйте разные сочетания ключевых слов, тогда шанс, что пользователь найдет именно ваш сайт, возрастет.

Помните, что сейчас поисковые системы не осуществляют поиск по ключевым словам из этого тега — они просматривают всю страницу, чтобы определить ее тематику. Поэтому старайтесь, чтобы ключевые слова соответствовали содержанию страницы, тогда у вашего сайта будет шанс оказаться в числе первых при поиске.

Параметр `Robots` дает некоторые управляющие команды роботам. В приведенном в листинге 1.7 примере данный атрибут указывает на то, что текущую страницу индексировать не нужно, однако следует пройти по ссылкам на странице и проиндексировать остальную часть сайта.

В табл. 1.5 приводятся команды, которые можно использовать для управления роботами.

Таблица 1.5. Значения атрибута content для свойства Robots

Значение	Описание
Index	Эта страница должна индексироваться
Noindex	Эта страница не должна индексироваться
Follow	Индексировать страницы по ссылкам с этой страницы
Nofollow	Не индексировать страницы по ссылкам с этой страницы
All	Эта страница должна индексироваться, индексировать страницы по ссылкам с этой страницы
None	Эта страница не должна индексироваться, не индексировать страницы по ссылкам с этой страницы

Встретив строку `URL`, робот прекратит индексацию текущей страницы и перейдет на страницу с адресом www.test.ru.

Итак, важность тега <META> очевидна: он отвечает за многие параметры страницы. Далеко не все его возможности будут необходимы вам постоянно, однако о том, что вы можете сделать с его помощью, следует знать.

Тег ‹STYLE›

Внутри блока этого тега задаются стили, используемые на странице. Этих тегов на странице может быть несколько.

Чтобы понять, как использовать этот тег, надо познакомиться с таким понятием, как таблицы стилей, потому что именно их синтаксис применяется для описания стилей внутри тега <STYLE>.

Каскадные таблицы стилей (CSS) используют для того, чтобы отделить HTML-код страницы от ее оформления. Другими словами, внутри таблицы описывается оформление различных тегов, а в HTML-коде — только применяемый стиль. Это очень удобный метод, с помощью которого вы можете менять оформление страницы, цвет фона, шрифт, не перебирая огромное количество HTML-кодов, а просто заменив цвет в таблице стилей.

Тег <STYLE> позволяет записывать внутри себя код в формате CSS. Чтобы браузер распознал формат кода, внутри этого блока используется атрибут TYPE и в качестве его значения используется MIME-тип: text/css.

Чтобы подключить к тегу какой-либо стиль, надо использовать атрибут CLASS и в качестве его значения установить название стиля, который вы хотите применить к этому тегу.

Изначально можно создавать стили для конкретного тега или общий стиль, который можно применить к любому тегу.

Можно задать разные стили для вывода на экран и для вывода на печать. Это удобно, если в качестве фона вы используете темные тона или рисунок. Совершенно не обязательно оставлять подобное декорирование документа при выводе на печать, тем более что это потребует больше краски для печати.

Чтобы задать определенное форматирование для разных случаев, используется атрибут MEDIA.

Значения атрибута MEDIA следующие:

- print — стили для вывода на печать;
- screen — стили для вывода на экран;
- all — стили для вывода на любое устройство.

Рассмотрим синтаксис CCS на примере кода из листинга 1.8.

Листинг 1.8. Использование стилей

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<style type="text/css" media="screen" >
p.style {
color:#CC0000;
background:#9999CC;
word-spacing:20px;
}
.style2 {
color:#66FFFF;
background:#990000;
word-spacing:10px;
}
</style>
<style type="text/css" media="print" >
p.style {
color:#000000;
background:#FFFFFF;
word-spacing:20px;
}
.style2 {
color:#000000;
background:#FFFFFF;
word-spacing:10px;
}
</style>
<title>Использование стилей</title>
</head>
<body>
<p class="style">
Текст со стилем style
```

```
</p>
<p class="style2">
Текст со стилем style2
</p>
<table border="1" >
<tr>
<td class="style2">
Для ячейки применен стиль style2
</td>
</tr>
</table>
</body>
</html>
```

В примере создано два стиля. Стиль `style` можно применять только для тега `<P>`, чтобы обозначить перед названием стиля название тега. Второй стиль — `style2` — можно применять для любого объекта. В листинге 1.8 стиль `style2` применяется также для ячейки таблицы.

Результат обработки листинга 1.8 представлен на рис. 1.3.

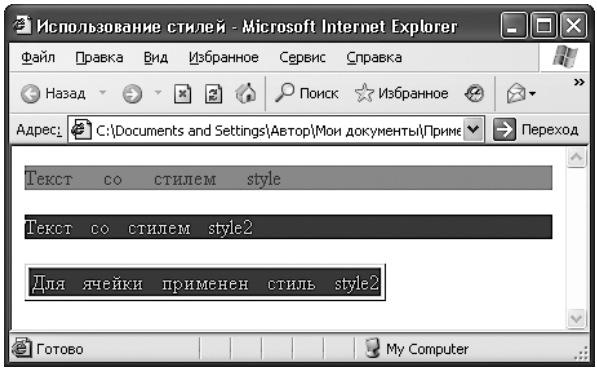


Рис. 1.3. Использование стилей

Для вывода на принтер используется черный шрифт с белым фоном, что позволяет сэкономить чернила в принтерах посетителей.

Применить к тегу стиль можно и другим способом: задать его в качестве значения атрибута STYLE (он доступен для всех тегов). В этом атрибуте в формате CSS через точку с запятой прописываются значения разных свойств тега.

В листинге 1.9 представлен пример задания свойств тегов с помощью атрибута STYLE.

Листинг 1.9. Использование атрибута STYLE

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Использование стилей</title>
</head>
<body>
<p style="color:#CC0000; background:#9999CC; word-spacing:20px;">
Текст с атрибутом style="color:#CC0000; background:#9999CC;
word-spacing:20px;"
</p>
<p style="color:#66FFFF; background:#990000; word-spacing:10px;">
Текст с атрибутом style="color:#66FFFF; background:#990000;
word-spacing:10px;"
</p>
<table border="1" >
<tr>
<td style="color:#66FFFF; background:#990000; word-spacing:10px;">
Описание ячейки таблицы с атрибутом style="color:#66FFFF; background:#990000;
word-spacing:10px;"
</td>
</tr>
</table>
</body>
</html>
```

Здесь применяются такие же стили, как и в предыдущем примере, однако параметры задаются с помощью атрибута STYLE.

На рис. 1.4 показан результат обработки кода из листинга 1.9.

Результаты обработки кодов на рис. 1.3 и 1.4 абсолютно одинаковые, однако задавать стили в начале документа удобнее, так как при необходимости их легко найти и исправить.

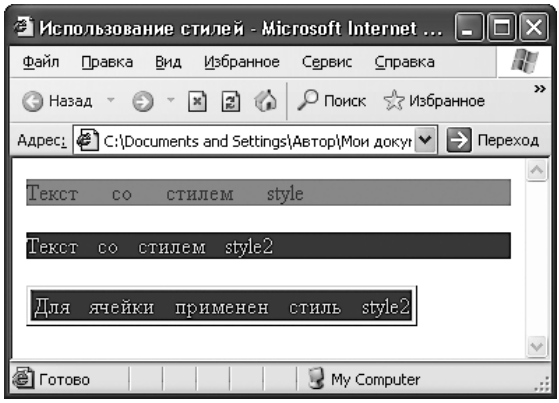


Рис. 1.4. Установка стилей через атрибут STYLE

Существует еще один способ задать стили для документа: записать их в отдельный файл CSS. При этом синтаксис записи в данный файл такой же, как и при использовании тега `<STYLE>`, однако этот способ более универсальный.

Подключить файл со стилями к документу можно двумя способами. При первом способе используется тег `<STYLE>` (код с его использованием представлен в листинге 1.10). Внутри блока этого тега записывается следующая строка: `@import URL("путь к файлу с таблицами");`.

Листинг 1.10. Импорт файлов с таблицами стилей

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<style>
@import URL("test.css");
</style>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251"
/>
<title>Использование стилей</title>
</head>
</html>
```

Такая конструкция позволяет применять к тегам документа стили, описанные в файле `test.css`. Помните, что конструкция `@import` должна быть первой в таблице стилей. В противном случае многие браузеры ее просто проигнорируют.

Прикрепить файл с таблицами стилей к документу можно и другим способом. Рассмотрим его в следующем подразделе.

Тег <LINK>

Использование тега <LINK> — это второй способ подключения файла с таблицами стилей к документу. Данный тег не требует закрывающего тега. Его возможности намного шире, нежели просто подключение таблиц стилей. Он определяет отношения между текущей страницей и другими документами. Поскольку отношения могут быть разными, то и тегов <LINK> на странице возможно несколько.

Основным атрибутом этого тега является HREF. Он указывает путь к объекту, для которого описывается тип связи. Этот атрибут является обязательным.

Второй атрибут — TYPE. Он устанавливает параметры объекта, с которым определяется тип связи. При связывании таблиц стилей этот атрибут принимает значение text/css.

У данного тега есть два взаимообратных атрибута — REL и REV. Первый определяет отношение одного документа к другим, а второй — отношение другого документа к текущему. Эти атрибуты могут принимать различные фиксированные значения, которые обозначают типы отношений.

В листинге 1.11 представлены примеры использования тега <LINK>.

Листинг 1.11. Использование тега <LINK>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<link href="test.css" rel="stylesheet" type="text/css" />
<link href="copyrihgt.html" rel="copyright" />
<link href="help.html" rev="help" />
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Использование тега LINK</title>
</head>
</html>
```

В примере из листинга 1.11 тег <LINK> использован для прикрепления к текущему документу таблиц стилей, для указания файла, в котором содержится информация

об авторском праве на данный документ, и определения файла со справочной информацией.

В табл. 1.6 приводятся некоторые возможные значения атрибута REL. Чтобы узнать об этих значениях подробнее, можно обратиться, например, к справочникам в Интернете.

Таблица 1.6. Некоторые значения атрибута REL

Значение	Описание
Top	Титульная страница набора документов либо основа структуры объектов
Contents	Ссылка на оглавление данного документа
Index	Документ, который можно использовать для индексного поиска по текущему документу
Glossary	Словарь терминов данного документа
Copyright	Авторские права
Next	Следующий документ в очереди просмотра
Previous	Предшествующий документ в очереди просмотра
Search	Поисковая страница
Help	Документ, призванный помочь пользователю (появляется по ссылке)

Мы выяснили, какие возможности имеет тег <LINK> относительно работы с различными документами, в частности с таблицами стилей. Более подробно на стилях остановимся позже, а сейчас рассмотрим тег, который пригодится при работе с другими важными объектами создания сайтов.

Тег ‹SCRIPT›

Тег <SCRIPT> позволяет присоединять к документу сценарии. Данный тег требует закрывающего тега, при этом текст сценария может располагаться либо в блоке этого тега, либо во внешнем файле. Если текст сценария расположен во внешнем файле, то он подключается с помощью атрибутов тега <SCRIPT>.

Рассмотрим атрибуты этого тега. Основными здесь можно считать два атрибута — TYPE и LANGUAGE. Первый задает MIME-тип содержимого страницы, а второй — язык, на котором написан сценарий. Рекомендуется задавать оба параметра одновременно на случай, если браузер один из них не воспримет. При этом если браузер понял значение атрибута TYPE, то значение атрибута LANGUAGE не воспринимается.

При подключении внешнего файла со сценариями в теге <SCRIPT> применяется атрибут SRC. Значением этого атрибута является место расположения файла

со сценариями. Это удобно в том случае, когда для различных страниц надо использовать одинаковые сценарии.

У тега <SCRIPT> есть также вспомогательный атрибут — DEFER. Он запрещает загружать сценарий до окончания полной загрузки страницы и таким образом позволяет оптимизировать время загрузки страницы.

В листинге 1.12 приводятся примеры подключения сценариев к HTML-документу.

Листинг 1.12. Использование тега ‹SCRIPT›

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script defer="defer" language="javascript" type="text/javascript">
<!--текст сценария-->
</script>
<script language="javascript" type="text/javascript" src="test.js">
</script>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Использование тега SCRIPT</title>
</head>
</html>
```

Количество подключаемых сценариев не ограничено, однако помните, что на их обработку требуется время, поэтому не стоит загружать браузер больше необходимого.

Если в документе создаются относительные ссылки на файлы, то при перемещении текущего документа файлы оказываются недоступными. В языке HTML есть инструменты для решения этой проблемы. Рассмотрим их в следующем подразделе.

Тег ‹BASE›

Тег <BASE> служит для того, чтобы задать базовый адрес, относительно которого вычисляются все относительные адреса. Это поможет избежать проблем при переносе вашей страницы в другое место (то есть все ссылки будут работать, как и прежде).

ПРИМЕЧАНИЕ



Абсолютный адрес документа указывает полный путь к нему, начиная с корневого каталога. Например, путь C:\Test\test.jpg используется для обозначения пути к файлам, находящимся на вашем компьютере, а http://www.test.ru/mytests/test.html — для документов, находящихся в Интернете. Если путь к файлу очень длинный и вводить его каждый раз неудобно, то можно задать относительный адрес. Например, чтобы из документа, находящегося по адресу C:\Test\MyTests\test.html, получить доступ к документу C:\Test\test.jpg, достаточно в нем набрать ..\test.jpg, при этом знак «..» означает переход на более высокий уровень иерархии каталогов.

Основным атрибутом тега <BASE> является HREF. В качестве значения этого атрибута используется адрес базовой папки, относительно которой будут вычисляться относительные адреса.

Еще одним полезным атрибутом данного тега является TARGET. Он определяет, в какое окно будут загружаться ссылки, встречающиеся в документе. Этот атрибут может принимать четыре значения.

Значения атрибута TARGET следующие:

- _top — отменяет все фреймы и загружает страницу в полное окно браузера;
- _blank — загружает страницу в новое окно;
- _self — загружает страницу в текущее окно;
- _parent — загружает страницу во фрейм-родитель.

В теге <BASE> можно использовать только один из указанных атрибутов.

В листинге 1.13 представлен пример использования тега <BASE>.

Листинг 1.13. Использование тега ‹BASE›

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<base href="http://www.test.ru/" target="_blank" />
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Использование тега BASE</title>
</head>
</html>
```

В примере из листинга 1.13 все относительные адреса будут браться от адреса http://www.test.ru/. Другими словами, если в тексте документа будет строка

с адресом `../IMG/test.jpg`, то браузер воспримет ее как адрес `http://www.test.ru/IMG/test.jpg`. При этом все ссылки будут открываться в новых окнах, если среди параметров самой ссылки не указать другой вариант.

Мы разобрались со всеми тегами, которые можно использовать внутри тега <HEAD>. Теперь перейдем к рассмотрению тела документа.

Тег ‹BODY›

Внутри блока тегов <BODY> располагается документ. Находящийся внутри этих тегов текст будет отображаться браузером. Все теги, отвечающие за форматирование документа, помещают в тег <BODY>, то есть внутри него будет находиться тело документа.

Атрибуты тега <BODY> применяются для того, чтобы установить общие для всего документа свойства, и в этом отношении возможности данного тега достаточно велики: можно задать цвет ссылок, параметры фона и т. д.

Сначала рассмотрим атрибуты тега <BODY>, управляющие отображением ссылок. Для удобства пользования страницей посетителями следует задавать разные цвета для посещенных, непосещенных и активных ссылок (то есть ссылок, на которые пользователь навел мышку). При этом надо следить, чтобы они не сливались с цветом фона страницы.

Атрибуты тега <BODY>, отвечающие за цвет ссылок, следующие:

- `alink` — задает цвет активной ссылки;
- `vlink` — задает цвет посещенной ссылки;
- `link` — задает цвет непосещенной ссылки.

Цвета ссылок можно задать в hex-формате или с помощью ключевых слов. Ключевые слова рационально использовать, если вы хотите применять стандартные цвета (так как их немного). Если же вы собираетесь задавать нестандартные цвета, то следует воспользоваться hex-форматом.

В примере из листинга 1.14 представлены оба варианта записи.

Листинг 1.14. Цвет ссылок

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
<base href="http://www.test.ru/" target="_blank" />
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Использование тега BODY</title>
</head>
<body alink="#00FF00" vlink="red" link="#330000">
</body>
</html>
```

В примере для активной и непосещенной ссылки цвет указывается с помощью цифр и букв, которые задают код цвета в шестнадцатеричном формате. Эти цифры и буквы записываются в качестве значения соответствующего атрибута после знака #. Код нужного вам цвета можно определить с помощью любого современного графического редактора.

Для посещенной ссылки цвет задается ключевым словом. Использовать их несложно. Для обозначения цвета с помощью ключевых слов используются стандартные названия цветов на английском языке — `blue`, `red`, `black` и т. д.

Тег `<BODY>` предоставляет большие возможности для управления фоном страницы. Рассмотрим атрибуты, отвечающие за это.

Для начала надо задать фоновый цвет страницы. Это делается с помощью атрибута `BGCOLOR`, значение которого — «цвет фона». Потом указывается фоновый рисунок страницы с помощью атрибута `BACKGROUND`, значение которого — «адрес файла с рисунком для фона». Старайтесь задавать цвет фона, близкий к цветовой гамме фонового рисунка. Это окажется полезным, если у пользователя не загрузится фоновый рисунок или же он будет загружаться медленно. Когда цвет фона близок к цвету рисунка, не будет дисбаланса в общем виде страницы.

Есть еще один атрибут. Он управляет прокруткой фона. Атрибут `BGPROPERTIES` определяет, будет ли фон прокручиваться вместе с текстом или нет. Он может принимать только одно значение — `Fixed`. Оно используется, если требуется зафиксировать фон документа, чтобы тот не прокручивался с текстом. В противном случае нужно удалить этот атрибут. При выборе параметров прокрутки фона надо действовать очень осторожно и следить за тем, чтобы при прокрутке не нарушалась читаемость текста.

В листинге 1.15 представлен пример использования атрибутов тега `<BODY>`, отвечающих за фон документа.

Листинг 1.15. Параметры фона

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<base href="http://www.test.ru/" target="_blank" />
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Использование тега BODY</title>
</head>
<body background="test.jpg" bgcolor="#0033CC" bgproperties="fixed">
</body>
</html>
```

В примере из листинга 1.15 файл `test.jpg` оформлен в синих тонах, поэтому фоновый цвет взят из синей гаммы, прокрутка фона вместе с текстом запрещена.

После того как мы разобрались с фоном, можно задать положение содержимого страницы относительно границ окна браузера.

Атрибуты тега `<BODY>`, отвечающие за отступ от границ окна, следующие:

- `bottommargin` — расстояние от нижнего края окна браузера до содержимого страницы;
- `leftmargin` — расстояние от левого края окна браузера до содержимого страницы;
- `topmargin` — расстояние от верхнего края окна браузера до содержимого страницы.

Расстояние для всех атрибутов устанавливается в пикселах.

В листинге 1.16 представлен пример того, как можно указать расстояние до содержимого страницы с помощью атрибутов тега `<BODY>`.

Листинг 1.16. Установка расстояния от границ окна до содержимого страницы

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<base href="http://www.test.ru/" target="_blank" />
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
```

```
<title>Использование тега BODY</title>
</head>
<body topmargin="10" leftmargin="20" bottommargin="10">
</body>
</html>
```

В примере расстояния сверху и снизу будут по 10 пикселов, а слева — 20 пикселов.

У тега <BODY> есть еще два атрибута, которые могут вам пригодиться. Атрибут TEXT отвечает за цвет текста в документе. Следите, чтобы цвет текста не сливался с фоном страницы.

Атрибут SCROLL определяет, будет ли иметь окно документа вертикальную полосу прокрутки. У него всего два значения — Yes и No, которые, соответственно, разрешают или запрещают прокрутку. Как правило, запрещать прокрутку не следует, потому что документ, который умещается на вашем экране, у посетителя с низким разрешением монитора может не влезть на экран и без полосы прокрутки он не увидит часть документа.

Мы рассмотрели возможности и функции тега <BODY>. Все параметры, заданные в этом теге, определяют общий вид документа.

Когда мы создаем большие сайты, то со временем можем забыть, что именно делает та или иная часть кода страницы. Избежать этого помогут комментарии.

Комментарии

Комментарии могут располагаться в любом месте страницы. В комментариях можно задавать пояснения к коду страницы. Это удобно, если код большой или когда с текстом работает несколько человек.

Комментарии заключаются в тег <!-- текст комментария -->. Текст, расположенный внутри этого тега, браузером не отображается.

Комментарии удобно использовать при создании HTML-документов, так как в них можно заключить временно ненужные фрагменты кода.

Ввод и оформление текста

Текстовая информация, которая будет отображаться на сайте, вводится внутри тега <BODY>. Просто размещать текст внутри тега можно, но не нужно: как правило, его

необходимо оформлять, например разделять на абзацы или создавать заголовки. Для оформления текста HTML предоставляет много возможностей: для любого абзаца или заголовка можно задать уникальный внешний вид, можно создать списки с маркерами любого типа, менять цвет, размер и шрифт текста. В общем, создатель сайта может оформить текст своего сайта очень красочно.

Рассмотрим общие принципы структурного форматирования текста, а затем перейдем непосредственно к управлению его внешним видом. Для начала разберем, как создать заголовок сайта.

Заголовки

Заголовок — важный элемент сайта, который помогает систематизировать текст. В HTML можно создавать заголовки разных уровней, что позволяет легко выделять смысловые темы и подтемы. Текст в заголовках влияет на индексацию сайта поисковыми системами, так как многие роботы при поиске обращают внимание именно на содержимое заголовков сайтов.

В HTML можно создавать заголовки шести уровней. Самым важным считается заголовок первого уровня, а наименее малозначимым — шестого.

Заголовки создаются тегами `<H1>`, `<H2>`, `<H3>`, `<H4>`, `<H5>` и `<H6>`. По умолчанию заголовок самого верхнего уровня выделяется самым крупным шрифтом. Чем ниже уровень заголовка, тем меньше шрифт.

Для заголовка любого уровня можно задать выравнивание по горизонтали с помощью атрибута `ALIGN`.

Значения атрибута `ALIGN` следующие:

- `left` — по левому краю;
- `right` — по правому краю;
- `center` — по центру;
- `justify` — по ширине (для заголовков длиннее строки).

Для заголовков можно задать атрибут `TITLE`, значение которого — «текст всплывающей подсказки» (той всплывающей подсказки, которая появляется при наведении курсора мыши на заголовок).

В листинге 1.17 представлен пример кода для создания разных типов заголовков и для различного вида выравнивания.

Листинг 1.17. Заголовки

```
<html>
<head>
<title>Создание заголовков</title>
</head>
<body>
<h1 align="center">Заголовок 1 уровня</h1>
<h2 align="justify">Заголовок 2 уровня</h2>
<h3 align="left">Заголовок 3 уровня</h3>
<h4 align="right">Заголовок 4 уровня</h4>
<h5>Заголовок 5 уровня</h5>
<h6 title="заголовок 6 уровня">Заголовок 6 уровня</h6>
</body>
</html>
```

На рис. 1.5 показан результат обработки браузером листинга 1.17.

На рис. 1.5 очевидно различие между заголовками разного размера. В принципе, придать тексту подобный вид можно с помощью обычного текста, но поскольку поисковики воспринимают текст заголовков как более приоритетный, то имеет смысл для выделения важной информации использовать именно теги заголовков.

При отображении заголовок всегда начинается с новой строки и после него всегда начинается новая строка (это отделяет заголовок от остального текста).

Для управления другими элементами отображения заголовков, например цветом или шрифтом, используется язык CSS. Поэтому в тегах `<H1…6>` разрешается использовать атрибуты STYLE и CLASS, которые подключают стили.

После того как вы создадите заголовки, можно добавить к ним обычный текст, который надо логически разбить на абзацы.

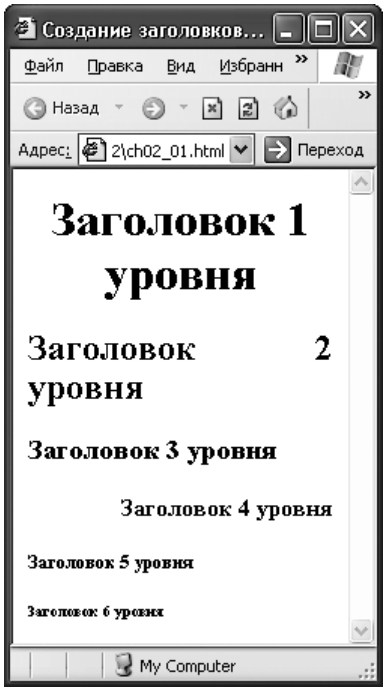


Рис. 1.5. Вид заголовков

Абзацы

Как известно, абзацы делят текст на логические части и на письме отделяются отступом от края листа. В HTML абзацы отбиваются друг от друга на расстояние в одну строку.

Для организации абзацев в HTML предусмотрен тег <P>, который подразумевает наличие закрывающего тега. Этот тег является, наверное, самым востребованным тегом: наибольшее количество текста, представленного на сайтах, находится внутри блока тегов <P>.

Для управления внешним видом текста абзацев используются в основном таблицы стилей. Однако незначительно текст можно отредактировать с помощью HTML.

Для тега <P> можно задать атрибут, определяющий выравнивание, — ALIGN. Этот атрибут может принимать несколько значений.

Значения атрибута ALIGN следующие.

- center — выравнивание по центру. При таком виде выравнивания текст располагается в центре экрана, образуя неровные края. Не стоит использовать это выравнивание для текстов больших объемов, поскольку читать такой текст неудобно;
- left — выравнивание по левому краю. Текст располагается у левого края окна браузера, а справа остаются неровные края. Это самый обычный вид выравнивания, при этом текст достаточно легкий для чтения;
- right — выравнивание по правому краю. Текст прижимается к правому краю экрана, образуя неровные края слева. Такое выравнивание подойдет небольшим эпиграфам. Выровненный по правому краю большой текст читать неудобно;
- justify — выравнивание по ширине. Пробелы между словами автоматически регулируются таким образом, чтобы текст прижимался к левому и правому краям окна. Этот вид выравнивания наилучший для больших объемов текста: ровные края слева и справа придают тексту аккуратный вид.

Кроме выравнивания для абзаца, можно задать текст всплывающей подсказки, которая появляется при наведении указателя мыши на текст. Подсказка создается с помощью атрибута TITLE, значением которого является текст подсказки.

В листинге 1.18 показан пример создания абзацев с разным выравниванием и всплывающими подсказками.

Листинг 1.18. Создание абзацев

```
<html>
<head>
<title>Абзацы</title>
</head>
<body>
<p align="center" title="Абзац, выровненный по центру">Это небольшой заголовочный абзац, выровненный по центру</p>
<p align="right" title="Абзац, выровненный по правому краю">Здесь эпиграф</p>
<p align="left" title="Абзац, выровненный по левому краю">Этот абзац с текстом выровнен по левому краю, поэтому края с правого края будут рваными</p>
<p align="justify" title="Абзац, выровненный по ширине">Этот абзац с основным текстом, выровненный по ширине. Края этого абзаца прилегают к краям окна браузера.</p>
</body>
</html>
```

На рис. 1.6 представлен результат обработки листинга 1.18. Очевидно различие между абзацами с разным выравниванием.

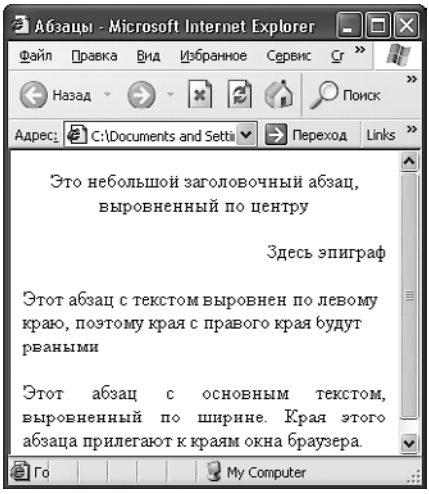


Рис. 1.6. Абзацы

Для дальнейшего оформления текста абзацев применяются каскадные таблицы стилей. Тег <P> позволяет использовать атрибуты STYLE и CLASS.

Обрывы строк

С оформлением абзаца все понятно. Но что делать, если необходимо оборвать строку, не закрывая абзац, например в том же эпиграфе или для записи стихов? В HTML возможно перенести текст на новую строку, не заканчивая абзац. Обычно браузер переносит слова в зависимости от размера окна, и возможность самостоятельно определить место переноса может пригодиться при записи стихов или для отделения различных элементов друг от друга.

Для переноса текста на новую строку служит тег
. Он не требует закрывающего тега, однако рекомендуется записывать его как
, чтобы все программы отображали его корректно.

В листинге 1.19 представлен пример кода для принудительного переноса строки.

Листинг 1.19. Обрыв строки

```
<html>
<head>
<title>Обрыв строки</title>
</head>
<body>
Наша Таня громко плачет.<br />
Уронила в речку мячик.<br />
</body>
</html>
```

На рис. 1.7 представлен результат обработки браузером кода из листинга 1.19.

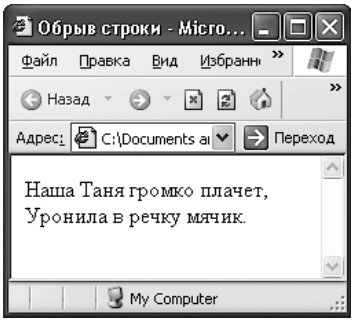


Рис. 1.7. Обрыв строки

На рис. 1.7 вы видите, что при обрыве строки с помощью тега
 не добавляется пустая строка после переноса, как это было бы при использовании тега <P>.

Есть еще один вариант применения тега
. Его используют, когда надо задать обтекание текстом таблицы, изображения или любого другого плавающего элемента страницы (то есть элемента, для которого задан атрибут ALIGN). Для этого применяется атрибут CLEAR.

Значения атрибута CLEAR следующие:

- all — запрещает обтекание элемента с двух сторон;
- left — запрещает обтекание с левой стороны плавающего объекта, расположенного после тега;
- right — запрещает обтекание с правой стороны плавающего объекта, расположенного после тега;
- none — отменяет свойство.

Если обтекание запрещено, то текст, следующий за тегом
, будет отображаться на строке после плавающего объекта.

Иногда нужно выполнить действие, обратное обязательному переносу строки, то есть гарантировать, что текст ни в коем случае не будет переноситься на новую строку. Для создания таких неразрывных строк используется тег <NOBR>, который требует закрывающего тега. Текст, размещенный внутри этих тегов, будет размещаться на одной строке. При необходимости браузер создаст горизонтальную полосу прокрутки.

В некоторых случаях строка может оказаться очень длинной и неудобной для чтения, поэтому внутри тегов <NOBR> можно использовать тег <WBR>, который указывает на место возможного переноса строки.

В листинге 1.20 представлен пример использования тегов <NOBR> и <WBR>.

Листинг 1.20. Запрет переноса строки

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251"
/>
<title>Обрыв строки</title>
</head>
<body>
<nobr>Это очень важная строка, ее ни в коем случае нельзя переносить на
другую строчку, однако в этом месте <wbr> возможно сделать перенос.</nobr>
</body>
</html>
```

На рис. 1.8 показан результат обработки в браузере кода из листинга 1.20.

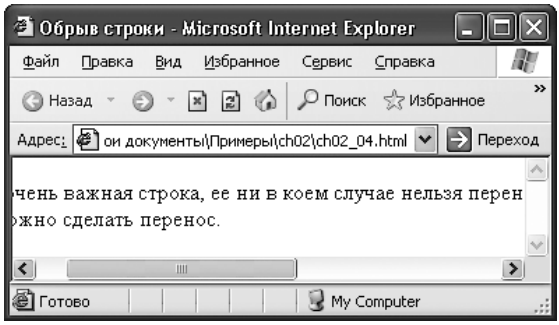


Рис. 1.8. Запрет переносов

Как видно на рис. 1.8, браузер создал горизонтальную полосу прокрутки и перенес строку в том месте, где мы разрешили.

Возможность управлять переносами в браузере позволяет контролировать общий вид страницы, потому нужно внимательно следить за тем, как браузер организовал переносы.

Обрывы страниц позволяют организовать простые списки, однако HTML предлагает специальный инструмент для этого.

Списки

Списки предназначены для организации и группировки данных. Это может пригодиться при создании карты сайта (то есть его оглавления), описания сложных структур и т. д.

В HTML можно выделить несколько типов списков, отличных по внешнему представлению информации:

- маркированный;
- нумерованный;
- список определений.

Маркированный список

Маркированный список — это список, в котором пункты отмечаются с помощью различных символов. Такой список называется также ненумерованным, или неупорядоченным, потому что для его элементов последовательность не важна. Этот список можно использовать для перечисления объектов или их свойств.

Для создания маркированных списков HTML предусматривает тег `<UL>`. Он требует закрывающий тег, а элементы списка находятся внутри блока тегов `<UL>`. Каждый элемент начинается с тега `<LI>`.

У тега `<UL>` есть атрибут TYPE, определяющий вид маркера списка.

Значения атрибута TYPE следующие:

- `circle` — создает маркер в виде круга, белого внутри;
- `square` — создает маркер в виде квадрата;
- `disc` — создает маркер в виде закрашенного черным круга.

Маркер может быть любым наиболее соответствующим, на ваш взгляд, виду страницы.

В листинге 1.21 представлен пример создания маркированных списков с разными маркерами.

Листинг 1.21. Маркированные списки

```
<html
<head>
<title>Списки</title>
<body>
<ul type="circle">
<li>Венера</li>
<li>Земля</li>
<li>Юпитер</li>
<li>Марс</li>
</ul>
<ul type="square" >
<li>Венера</li>
<li>Земля</li>
<li>Юпитер</li>
<li>Марс</li>
</ul>
<ul type="disc" >
<li>Венера</li>
<li>Земля</li>
<li>Юпитер</li>
<li>Марс</li>
</ul>
</body>
</html>
```

На рис. 1.9 показан результат обработки кода из листинга 1.21.

На этом рисунке вы видите различные маркеры списков.

При создании маркированных списков с помощью тега <LI> можно задать вид маркера отдельно для каждого элемента списка.

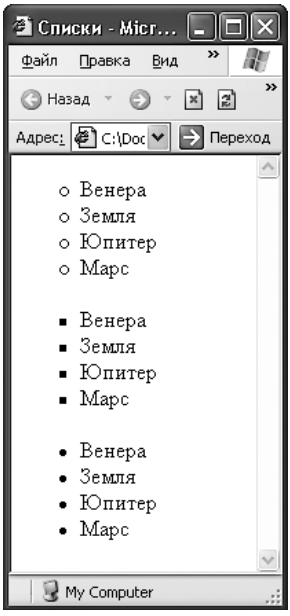


Рис. 1.9. Маркированные списки

В листинге 1.22 представлен пример создания списка с разными маркерами для разных элементов.

Листинг 1.22. Список с разными маркерами

```
<html>
<head>
<title>Списки</title>
<body>
<ul>
<li type="circle">Земля</li>
<li type="circle">Марс</li>
<li type="square">Солнце</li>
<li type="circle">Венера</li>
<li type="disc">Луна</li>
</ul>
</body>
</html>
```

На рис. 1.10 представлен результат обработки браузером кода из листинга 1.22.

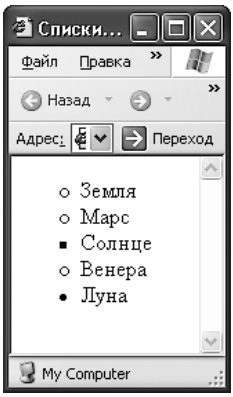


Рис. 1.10. Список с разными маркерами

В примере разные маркеры отмечают объекты разных типов.

Иногда удобнее создавать собственные маркеры для списков, чтобы их внешний вид соответствовал стилю вашего сайта и вашим желаниям. Для этого в HTML есть возможность создать список с графическими маркерами — вместо тега `<LI>` можно подставить картинку с изображением. Тег `<UL>` укажет браузеру, что нужно сделать отступ, а с помощью тега `<BR>` — перевод строки.

В листинге 1.23 представлен пример создания списка с графическими маркерами.

Листинг 1.23. Список с графическими маркерами

```
<html>
<head>
<title>Списки</title>
<body>
<ul>
<img src="marker.jpg" />Венера<br />
```

```
<img src="marker.jpg" />Марс<br />
<img src="marker.jpg" />Земля<br />
</ul>
</body>
</html>
```

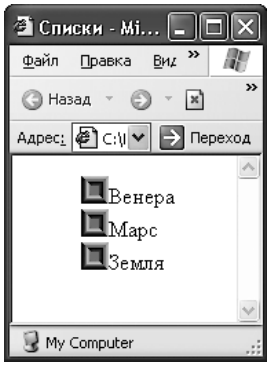


Рис. 1.11. Список с графическими маркерами

На рис. 1.11 показан результат обработки кода из листинга 1.23.

Теперь вы можете использовать картинки для создания симпатичных списков, соответствующих стилю вашего сайта.

Однако применение неупорядоченных списков не всегда удобно. Что же делать, если нужно расписать порядок действий? Например, рецепт приготовления блюда. Для перечисления ингредиентов можно использовать неупорядоченный список, а для описания порядка действий — нумерованный список.

Нумерованный список

Нумерованный список — это список, который применяется тогда, когда порядок следования его пунктов имеет большое значение, например при описании алгоритмов и других пошаговых действий.

Для создания упорядоченных списков применяется тег <OL>, которому требуется закрывающий тег. Все пункты списка находятся внутри блока этих тегов.

Тег <OL> имеет атрибут TYPE, который задает формат символов, используемых для нумерации.

Значения атрибута TYPE следующие:

- A — пункты нумеруются заглавными буквами латинского алфавита;
- a — пункты нумеруются строчными буквами латинского алфавита;
- I — пункты нумеруются заглавными римскими цифрами;
- i — пункты нумеруются строчными римскими цифрами;
- 1 — пункты нумеруются арабскими цифрами.

Вторым атрибутом тега <OL> является атрибут START, который указывает, с какого числа следует начинать нумерацию списка.

В листинге 1.24 показан пример кода для создания упорядоченного списка с разной нумерацией.

Листинг 1.24. Упорядоченные списки

```
<html>
<head>
<title>Списки</title>
<body>
<ol type="1" start="6">
<li>Достать ключ</li>
<li>Вставить ключ в замочную скважину</li>
<li>Повернуть ключ по часовой стрелке на два оборота</li>
<li>Достать ключ из замка</li>
<li>Открыть дверь</li>
</ol>
<ol type="A">
<li>Достать ключ</li>
<li>Вставить ключ в замочную скважину</li>
<li>Повернуть ключ по часовой стрелке на два оборота</li>
<li>Достать ключ из замка</li>
<li>Открыть дверь</li>
</ol>
<ol type="a">
<li>Достать ключ</li>
<li>Вставить ключ в замочную скважину</li>
<li>Повернуть ключ по часовой стрелке на два оборота</li>
<li>Достать ключ из замка</li>
<li>Открыть дверь</li>
</ol>
<ol type="I">
<li>Достать ключ</li>
<li>Вставить ключ в замочную скважину</li>
<li>Повернуть ключ по часовой стрелке на два оборота</li>
<li>Достать ключ из замка</li>
<li>Открыть дверь</li>
```

```
</ol>
<ol type="i">
<li>Достать ключ</li>
<li>Вставить ключ в замочную скважину</li>
<li>Повернуть ключ по часовой стрелке на два оборота</li>
<li>Достать ключ из замка</li>
<li>Открыть дверь</li>
</ol>
</body>
</html>
```

На рис. 1.12 представлен результат обработки кода из листинга 1.24 в браузере.

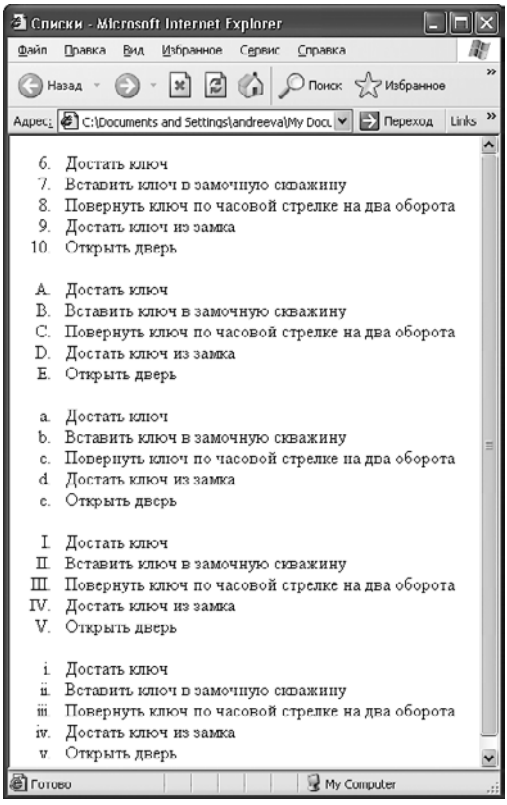


Рис. 1.12. Упорядоченные списки

В примере созданы списки с различными типами нумерации, для списка с арабской нумерацией задан стартовый номер 6.

При создании упорядоченных списков на тег <LI> можно возложить дополнительные функции. Как и в примере с маркированными списками, в теге <LI> можно задать вид нумерации конкретного пункта с помощью атрибута TYPE. Кроме того, атрибут VALUE позволяет задать номер, с которого будет продолжаться нумерация списка.

В примере из листинга 1.25 представлен код для создания списка с разными типами нумерации и различным порядком следования элементов.

Листинг 1.25. Список с разными типами нумерации

```
<html>
<head>
<title>Списки</title>
<body>
<ol type="1" >
<li type="1" value="10">Закрой дверь</li>
<li>Сходи в магазин</li>
<li value="1">Открой дверь</li>
<li>Возьми деньги</li>
<li type="I">Иди налево</li>
<li type="I">Потом поверни направо</li>
<li>Увидишь магазин</li>
</ol>
</body>
</html>
```

На рис. 1.13 вы видите результат обработки листинга 1.25.

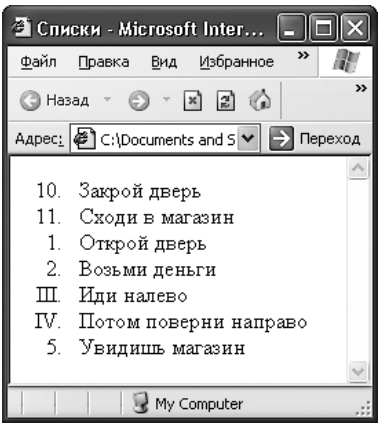


Рис. 1.13. Упорядоченный список с разной нумерацией

Как видно на рис. 1.13, порядок нумерации и тип ее отображения отделены друг от друга, изменение типа чисел не влияет на числовое обозначение пунктов.

Список определений

Бывает, что на сайте нужно разместить список терминов, проще говоря, словарь. Для создания подобных конструкций служит список определений.

Список определений — это особый вид списка, который применяется для форматирования словарей и в случаях, когда необходимо пояснить значения терминов.

Особенность списка определения следует из его функций. Элемент такого списка всегда состоит из двух частей: первая часть задает определяемое слово, а вторая — описание или расшифровку термина. При этом форматирование осуществляется таким образом, что описание термина отображается с отступом от края экрана и, возможно, с пропуском строки от определения.

Для организации списков определений служит тег <DL> — это тег-контейнер, внутри которого находятся определение и описание термина. У этого тега нет атрибутов, кроме стандартных STYLE и CLASS, с помощью которых к определению можно подключить стили.

Чтобы внести информацию внутри тега-контейнера <DL>, надо задать теги <DT> и <DD>.

Тег <DT> используется для того, чтобы задать определение. Никаких особенных атрибутов он не имеет.

Тег <DD> используется для описания термина из тега <DT>. Особых атрибутов в нем также не предусмотрено. Чтобы изменить вид информации, представленной в этом теге, надо использовать таблицы стилей.

В листинге 1.26 представлен пример создания списков определений.

Листинг 1.26. Списки определений

```
<html>
<head>
<title>Списки</title>
<body>
<dl>
<dt>Земля</dt>
<dd>Третья планета Солнечной системы</dd>
<dt>Марс</dt>
<dd>Четвертая планета Солнечной системы</dd>
</dl>
</body>
</html>
```

На рис. 1.14 показан результат обработки кода из листинга 1.26.

На рис. 1.14 видны особенности форматирования списков определений, информацию в таком виде воспринимать гораздо проще.

Списки можно вкладывать друг в друга в виде подсписков. Об этом мы и поговорим в следующем пункте.

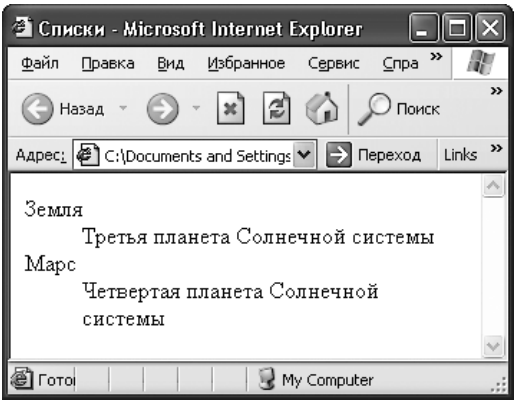


Рис. 1.14. Списки определений

Вложенные списки

Часто возможностей простых списков не хватает. Например, при создании оглавлений никак не обойтись без вложенных пунктов. Теперь рассмотрим, каким образом создаются вложенные списки.

В HTML можно комбинировать и вкладывать друг в друга списки разных типов. При этом надо внимательно следить за тем, где заканчивается вложенный список, а где — список верхнего уровня, иначе возникнут проблемы с отображением информации.

В листинге 1.27 представлен код для создания вложенного списка.

Листинг 1.27. Вложенные списки

```
<html>
<head>
<title>Списки</title>
<body>
<ol>
<li>
Открой дверь
<ul>
<li>Ключом</li>
<li>Отмычкой</li>
</ul>
</li>
<li>
Зайди в квартиру
```

```
<ul>
<li>Тихо шагая</li>
<li>Громко топая</li>
</ul>
</li>
<li>Ляг спать</li>
</ol>
</body>
</html>
```

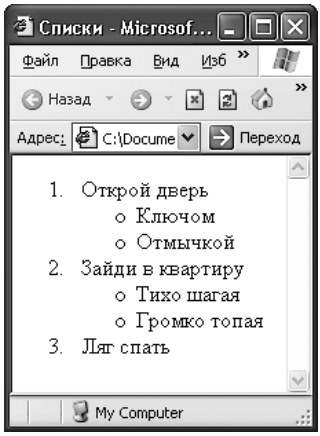


Рис. 1.15. Вложенный список

На рис. 1.15 можно увидеть, как выглядит вложенный список.

Вы видите, что элементы внутренних списков отступают от элементов списка более высокого уровня.

Ссылки

Мы разобрались с большей частью принципов оформления текста. Теперь рассмотрим самое главное, на чем основывается сеть Интернет, — ссылки.

По своей сути Интернет базируется на информации и ссылках на нее. Ссылки связывают документы, разбросанные по всему Интернету, в одну сеть. Ваш сайт может быть записан в памяти разных компьютеров, но для посетителя он будет отображаться как единое целое благодаря ссылкам.

Существует два типа ссылок: внешние и внутренние. Внешние ссылки связывают страницы и помогают осуществлять навигацию. Внутренние ссылки дают возможность передвигаться в рамках одной страницы.

Внешние ссылки

Внешние ссылки — это ссылки на объекты, расположенные вне текущей страницы. Это могут быть другие страницы сайта, картинки, мультимедиа-приложения и иные сайты.

Основой внешних ссылок является URL-адрес объекта, на который вы хотите сослаться.

Для создания ссылок в HTML служит тег <A>, который требует закрывающий тег. В блоке этих тегов располагается текст или другой объект, который будет выделяться как ссылка. На самом деле для того, чтобы сообщить человеку, что в каком-то месте сайта расположена ссылка, совершенно не обязательно писать адрес страницы. Язык HTML дает возможность «замаскировать» адрес под текст ссылки, который описывает, что увидит пользователь, перейдя по ней. Отдельно находится адрес страницы, на которую вы попадете при щелчке кнопкой мыши на ссылке.

Между тегами <A> и </A> располагается текст ссылки, а адрес документа для перехода записывается в теге <A> в качестве значения атрибута HREF. Таким образом, пользователь сможет быстро и легко перемещаться по сайту. Ему абсолютно все равно, где находится документ, на который указывает ссылка, он видит только текст описания.

В листинге 1.28 представлен пример создания ссылок на HTML-страницу и картинку.

Листинг 1.28. Создание ссылок

```
<html>
<head>
<title>Ссылки</title>
<body>
<a href="aboutme.html">Обо мне</a><br />
<a href="myfoto.jpg">Мое фото</a>
</body>
</html>
```

На рис. 1.16 показан пример отображения ссылок в браузере.

Текст ссылки, как правило, отображается подчеркнутым, а при наведении на ссылку курсор меняет вид. Когда посетитель сайта щелкнет кнопкой мыши на ссылке, он перейдет на страницу, которая указывается в качестве адреса ссылки.

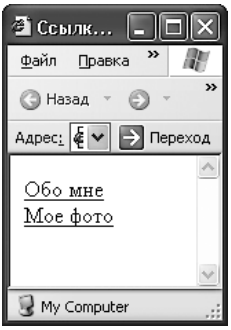


Рис. 1.16. Ссылки

Внутренние ссылки

Внутренние ссылки — это ссылки, организующие переходы внутри одного HTML-документа. Они применяются в случае, когда текста на одной странице очень много. Для простоты навигации можно создать ссылки, при щелчке кнопкой мыши на которых пользователь автоматически перейдет к нужной части документа.

Чтобы создать такую ссылку, сначала нужно определить место, к которому приводит ссылка. Это делается с помощью атрибута NAME тега <A>. Необходимый отрывок текста заключается в блок тега <A>. Однако совершенно не обязательно брать внутрь этого тега текст — можно просто установить его на месте, к которому браузер должен переходить при щелчке кнопкой мыши на ссылке.

В качестве значения атрибута NAME можно взять любое имя. Желательно, чтобы оно характеризовало текущее место — так вам самим будет проще пользоваться метками.

Затем надо создать ссылку на эту метку. Ссылка на внутреннюю метку создается так же, как и ссылка на внешний документ, только вместо URL-адреса желаемой страницы надо ввести адрес метки в таком виде: `#met1`, где `met1` — имя метки. При щелчке кнопкой мыши на ссылке браузер автоматически перейдет к месту, указанному меткой.

В листинге 1.29 представлен пример создания внутренних ссылок.

Листинг 1.29. Внутренние ссылки

```
<html>
<head>
<title>Ссылки</title>
<body>
<a name="met2">Это метка для перехода 2</a><br />
<a href="#met1">Кликните для перехода к метке 1</a><br />
<a href="#met2">Кликните для перехода к метке 2</a><br />
<a name="met1">Это метка для перехода 1</a>
</body>
</html>
```

На рис. 1.17 показано, как внутренние ссылки представляются в браузере.

Как вы видите, внутренние ссылки при отображении ничем не отличаются от внешних ссылок, а текст, отмеченный как метка, не выделяется.

Синтаксис внутренних ссылок такой же, как и у внешних, поэтому атрибуты тега <A> одинаковы для обоих типов.

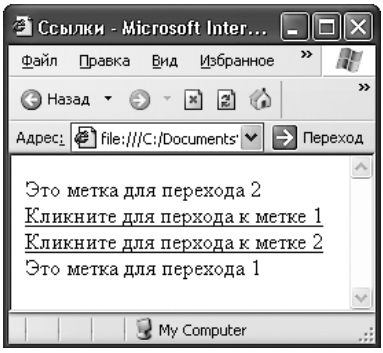


Рис. 1.17. Внутренние ссылки

Общие моменты при создании внешних и внутренних ссылок

Все немногочисленные атрибуты тега <A> можно применять при создании как внешних, так и внутренних ссылок.

Тег <A> имеет два вспомогательных атрибута.

Первый атрибут — TARGET. Он указывает на то, в каком окне должен открываться документ, к которому ведет ссылка.

Значения атрибута TARGET следующие:

- _blank — страница открывается в новом окне браузера;
- _parent — страница загружается во фрейм-родитель;
- _self — страница загружается в текущее окно;
- _top — страница загружается в полное окно браузера.

Этот атрибут очень полезен, потому что иногда ссылку удобнее открыть в новом окне, например, когда она уводит на другой сайт, а вы не хотите, чтобы посетитель забыл о вашем сайте. Если страница откроется в новом окне, то это гарантирует, что посетитель снова взглянет на вашу страницу. Однако не стоит забывать, что многие браузеры блокируют такие окна, особенно если они открываются без ведома пользователя.

Вторым вспомогательным атрибутом является TITLE, который позволяет создать всплывающую подсказку для вашей ссылки. Это очень удобно, так как дает пользователю дополнительную информацию о том, что его ждет при щелчке кнопкой мыши на ссылке. Значением атрибута является текст всплывающей подсказки.

В листинге 1.30 представлен пример создания ссылок с рассмотренными выше атрибутами.

Листинг 1.30. Ссылки с дополнительными атрибутами

```
<html>
<head>
<title>Ссылки</title>
<body>
<a href="aboutme.html" target="_blank" title="Пройдя по ссылке, вы сможете
узнать обо мне">Обо мне</a>
</body>
</html>
```

На рис. 1.18 показан пример отображения ссылки со всплывающей подсказкой.

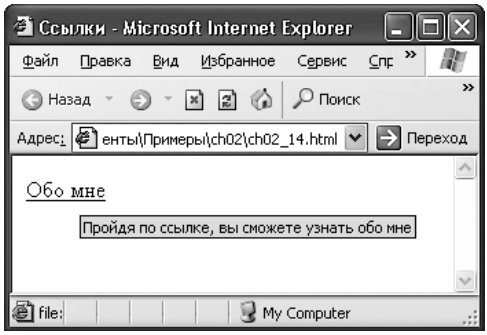


Рис. 1.18. Ссылки с дополнительными атрибутами

Всплывающая подсказка помогает посетителю страницы решить, нужно ли ему переходить по ссылке.

Любая ссылка, как внешняя, так и внутренняя, может приводить к изображению. Другими словами, новая страница будет открываться при щелчке кнопкой мыши на картинке. Можно, например, создать миниатюрные копии фотографий, при щелчке кнопкой мыши на которые будет открываться фотография большего размера.

Для создания ссылки-изображения достаточно поместить картинку внутрь тега `<A>`.

В листинге 1.31 представлен пример создания ссылки-изображения.

Листинг 1.31. Ссылки-изображения

```
<html>
<head>
<title>Ссылки</title>
<body>
<a href="mypetfoto.html" target="_
blank" title="Пройдя по ссылке, вы
сможете увидеть фото моего зверя"><img
src="mypetfotomini.jpg" /></a>
</body>
</html>
```

На рис. 1.19 показывается, как браузер отображает ссылки-изображения.



Рис. 1.19. Ссылка-изображение

Вокруг картинки создается рамка, которая дает посетителю понять, что перед ним не просто изображение, а ссылка.

Вы можете создать ссылку любого вида на каждый объект. Следите за тем, чтобы ссылки имели понятные названия, соответствующие содержанию документов,

появляющихся при щелчке кнопкой мыши на них. Это гарантирует, что посетители будут довольны сайтом и в дальнейшем снова зайдут на него.

Вы можете задавать внешний вид текста на странице. Рассмотрим, как это делается.

Форматирование текста

Для чего нужно форматировать текст? Это делается для того, чтобы сайт был выдержан в одном стиле. Как правило, вид и цвет текста, принятые по умолчанию, не отвечают стилю большинства сайтов. Поэтому текст приходится форматировать самостоятельно, для чего HTML предоставляет огромные возможности.

Существует множество тегов для форматирования текста. Их можно разделить на две группы: логические и физические. Отличие этих групп друг от друга принципиальное.

Логические теги сообщают браузеру о том, какой тип информации в них содержится, например важный текст или цитата. Браузер решает, как отобразить такой текст. Существуют стандарты для отображения определенных тегов, но главное в этих тегах — их смысловое значение. По сути, теги разбивают документ на логические части, при этом они не обязывают браузер отображать текст внутри них каким-либо конкретным образом.

Физические теги просто говорят браузеру, как должен выглядеть тот или иной блок текста, не уточняя смысл и значимость его содержимого. Иначе говоря, такой тег указывает браузеру нарисовать букву красным цветом, полужирным шрифтом или курсивом, не уточняя, почему эта буква должна выглядеть именно так.

В некотором роде воздействие на внешний вид текста у многих тегов одинаковое. Например, для выделения текста полужирным шрифтом можно использовать как логические, так и физические теги.

Разберем подробнее теги форматирования текста. Начнем с рассмотрения логических тегов.

Теги логического форматирования

Теги логического форматирования определяют не внешний вид текста, а его тип, в зависимости от которого браузер применяет тот или иной вид внешнего форматирования. Все рассматриваемые ниже теги являются контейнерами и требуют закрывающий тег.

Некоторые из этих тегов могут вообще не влиять на отображение текста, поэтому при рассмотрении тегов логического форматирования будем делать упор на то, как они определяют значение текста, а не на то, как они его форматируют.

Тег ‹ABBR›

Тег <ABBR> определяет текст как аббревиатуру с помощью атрибута TITLE. Вы можете задать всплывающую подсказку с расшифровкой аббревиатуры. При этом поисковые роботы индексируют именно полный вариант расшифровки, определенный атрибутом TITLE.

Пример:

```
<abbr title="Научно-исследовательский институт">НИИ</abbr>
```

Тег ‹ACRONYM›

Тег <ACRONYM> указывает, что текст является акронимом.



ПРИМЕЧАНИЕ

Акронимы — это устоявшиеся сокращения, например СНГ, США и т. п.

Атрибут TITLE позволяет задать расшифровку акронима.

Пример:

```
<acronym>СНГ</acronym>
```

Тег ‹CITE›

Тег <CITE> отмечает небольшую цитату или сноску, взятую из другого источника. Такой текст обычно отображается курсивом.

Пример:

```
<cite>Здесь указан источник информации </cite>
```

Тег ‹CODE›

Тег <CODE> указывает на программный код, который может содержать, например, переменные, функции или небольшие куски программы. Такой текст обычно выводится моноширинным шрифтом.

Пример:

```
Зададим функцию <code> func(int a, char b);</code>
```

Тег <DEL>

Тег <DEL> помечает текст как удаленный. Он может использоваться при внесении изменений в документы. Этот тег имеет два атрибута: CITE, содержащий URL-адрес документа, в котором описаны причины удаления фрагмента, и DATETIME, включающий дату и время удаления фрагмента в формате ГГГГ-ММ-ДДТчч:мм:ссTZD.

TZD — это указатель часового пояса. Он может принимать два типа значений. Первый тип значений — это указатель Z, указывающий, что время задается относительно нулевого часового пояса (иначе говоря, он обозначает *время UTC* — Coordinated Universal Time, универсальное время). Второй тип значений имеет формат +чч.мм или -чч.мм, который задает сдвиг вперед (+) или назад (–) указанного времени относительно времени UTC. Браузеры отображают такой текст как зачеркнутый.

Пример:

```
<del cite="whydel.html" datetime="2007-10-06T10:11:45+3.00"> Неактуальный фрагмент</del>
```

Тег <DFN>

Тег <DFN> выделяет текст как определение. Этот тег можно использовать, если новый термин встречается в тексте впервые и тут же дается его определение. Браузер отображает такой текст курсивом.

Пример:

```
<dfn>Определение</dfn>-описание
```

Тег <EM>

Тег <EM> выделяет важные фрагменты текста. Браузер отображает такой текст курсивом.

Пример:

```
<em>Важно</em>
```

Тег <INS>

Тег <INS> отмечает текст как вставку, применяется для выделения изменений, вносимых в документ. У этого тега есть два атрибута: CITE, содержащий URL-адрес документа, в котором описаны причины добавления фрагмента, и DATETIME, включающий дату и время добавления в формате ГГГГ-ММ-ДДТчч:мм:ссTZD (TZD — это часовой пояс). Браузеры такой текст отображают как подчеркнутый.

Пример:

```
<ins cite="whyadd.html" datetime="2007-10-06T10:11:45+3.00"> Новый фрагмент</ins>
```

Тег ‹KBD›

Тегом <KBD> помечается текст, вводимый пользователем с клавиатуры. Браузеры такой текст отображают моноширинным шрифтом.

Пример:

```
Введите слово <kbd>дом</kbd>
```

Тег ‹Q›

Тег <Q> обозначает текст как цитату, используется для добавления коротких цитат в текст. Обычно такой текст отображается курсивом, некоторые браузеры заключают его в кавычки.

Пример:

```
Цитата: <q>Как сказал поэт</q>
```

Тег ‹SAMP›

Тег <SAMP> определяет выделенный текст как пример, обычно используется для выделения результатов работы программы. Браузер выделяет этот текст моноширинным шрифтом.

Пример:

```
<samp>Образец</samp>
```

Тег ‹STRONG›

Тег <STRONG> предназначен для постановки акцента на тексте. Такой текст браузеры выделяют полужирным шрифтом.

Пример:

```
<strong>Очень важный фрагмент</strong>
```

Тег ‹VAR›

Тег <VAR> используется для выделения переменных из программ. Браузер отмечает такой текст курсивом.

Пример:

```
Введите переменную <var>X</var>
```

В листинге 1.32 показан код страницы с различными видами логического форматирования текста.

Листинг 1.32. Логическое форматирование текста

```
<html>
<head>
<title>Теги логического форматирования</title>
<body>
<abbr title="Научно-исследовательский институт">НИИ</abbr><br />
<acronym>СНГ</acronym><br />
<cite>Здесь указан источник информации</cite><br />
Зададим функцию <code>func(int a, char b);</code><br />
<del cite="whydel.html" datetime="2007-10-06T10:11:45+3.00">Неактуальный фрагмент</del><br />
<dfn>Определение</dfn>-описание<br />
<em>Важно</em><br />
<ins cite="whyadd.html" datetime="2007-10-06T10:11:45+3.00">Новый фрагмент</ins><br />
Введите слово <kbd>дом</kbd><br />
Цитата: <q>Как сказал поэт</q><br />
<samp>Образец</samp></br>
<strong>Очень важный фрагмент</
strong> <br />
Введите переменную <var>X</var><br />
</body>
</html>
```

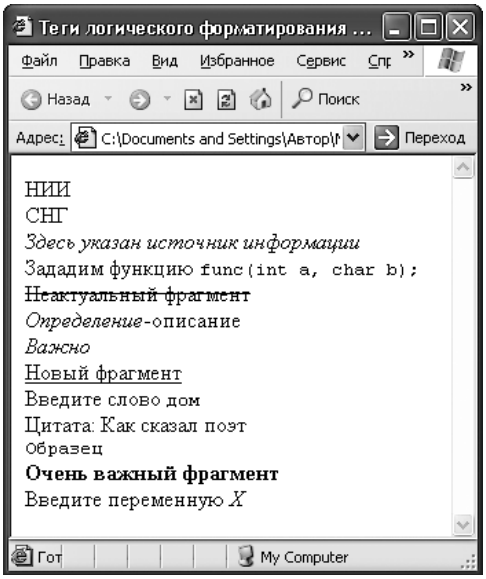


Рис. 1.20. Логическое форматирование текста

На рис. 1.20 показан результат обработки браузером кода из листинга 1.32.

На рис. 1.20 видно, что не все теги логического форматирования изменяют вид текста. Это объясняется тем, что они предназначены не для этого. Различные браузеры могут по-разному выводить

различные теги, поэтому перед тем как выложить сайт в Интернете, проверьте его вид в различных программах.

Вышеупомянутые теги используются относительно редко.

Теги физического форматирования

Теги физического форматирования сообщают браузеру, как должен выглядеть текст внутри тега.

Тег <B>

Тег <B> задает полужирное написание шрифта.

Пример:

```
<b>Полужирный шрифт</b>
```

Тег <I>

Тег <I> отображает выделенный текст курсивом.

Пример:

```
<i>Курсив</i>
```

Тег <TT>

Тег <TT> задает моноширинное написание текста.

Пример:

```
<tt>Моноширинный шрифт</tt>
```

Тег <U>

Тег <U> отображает текст подчеркнутым шрифтом.

Пример:

```
<u>Подчеркнутый</u>
```

Тег <S>

Тег <S> зачеркивает текст горизонтальной линией.

Пример:

```
<s>Зачеркнутый</s>
```

Тег <STRIKE>

Тег <STRIKE> также зачеркивает текст горизонтальной линией.

Пример:

```
<strike>Снова зачеркнутый</strike>
```

Тег <BIG>

Тег <BIG> отображает текст внутри тега шрифтом большего размера относительно остального текста.

ПРИМЕЧАНИЕ



В языке HTML размеры шрифта измеряются в условных единицах (от 1 до 7). По умолчанию принят размер 3.

При вложении тегов размер шрифта будет каждый раз увеличиваться на единицу.

Пример:

```
Шрифт <big>побольше <big>Еще больше</big></big>
```

Тег <SMALL>

Тег <SMALL> отображает выделенный текст шрифтом меньшего размера относительно остального текста. При вложении тегов шрифт будет уменьшаться на единицу с каждым вложением.

Пример:

```
Шрифт <small>поменьше</small>
```

Тег <SUB>

Тег <SUB> задает подстрочное написание символов, то есть текст располагается ниже уровня базовой строки и становится меньшего размера. Это удобно при вводе формул.

Пример:

```
<sub>Подстрочный</sub> шрифт
```

Тег <SUP>

Тег <SUP> задает надстрочное написание символа, то есть он располагается выше базовой линии и становится меньшего размера. Этот тег также используют при написании формул.

Пример:

```
<sup>Надстрочный</sup> шрифт
```

Тег ‹SPAN›

Тег <SPAN> выделяет часть текста и определяет для нее особые параметры отображения с помощью таблиц стилей. Этот тег используется для выделения небольших частей текста.

Пример:

```
<span style="background-color:#00FFFF">Текст с фоном</span>
```

В листинге 1.33 представлен код страницы с различными видами физического форматирования текста.

Листинг 1.33. Физическое форматирование

```
<html>
<head>
<title>Теги физического форматирования</title>
<body>
<b>Полужирный шрифт</b><br />
<i>Курсив</i><br />
<tt>Моноширинный шрифт</tt><br />
<u>Подчеркнутый</u><br />
<s>Зачеркнутый</s><br />
<strike>Снова зачеркнутый</strike><br />
Шрифт <big>побольше <big>Еще больше</big></big><br />
Шрифт <small>поменьше</small><br />
<sub>Подстрочный</sub> шрифт<br />
<sup>Надстрочный</sup> шрифт<br />
<span style="background-color:#00FFFF">Текст с фоном</span>
</body>
</html>
```

На рис. 1.21 показан результат обработки браузером кода из листинга 1.33.

Как вы видите, любой тег физического форматирования изменяет вид текста, так как именно это и является их функцией.

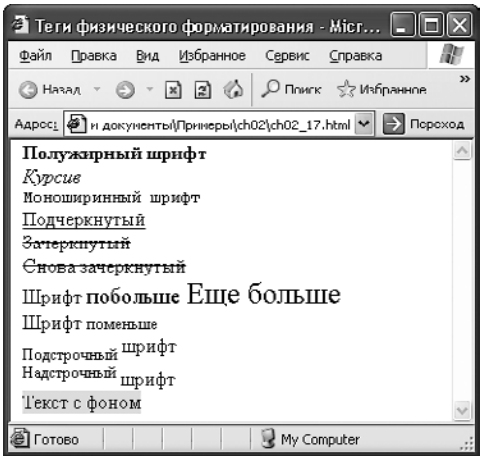


Рис. 1.21. Физическое форматирование текста

Как вы могли заметить, зачастую действие логических и физических тегов дублируется. Рекомендуется все же использовать теги логического форматирования, так как они определяют суть фрагмента текста.

В табл. 1.7 кратко описаны аналоги рассмотренных физических тегов среди логических тегов и показано, какие теги являются взаимозаменяемыми.

Таблица 1.7. Теги форматирования текста

Физический тег	Логический тег	Действие на текст
B	STRONG	Полужирный
I	EM, DFN, VAR, CITE	Курсив
TT	CODE, SAMP, KBD	Моноширинный
U	INS	Подчеркнутый
S, STRIKE	DEL	Зачеркнутый
BIG	нет	Увеличение шрифта
SMALL	нет	Уменьшение шрифта
SUB	нет	Подстрочный
SUP	нет	Надстрочный

Как видно из таблицы, для многих тегов физического форматирования можно найти логические аналоги, которые намного лучше описывают суть выделенного текста.

Все вышеприведенные теги применяются для форматирования небольших блоков текста. Далее рассмотрим теги, которые можно применять для изменения внешнего вида крупных текстовых блоков.

Теги форматирования больших блоков текста

Теги, которые мы рассмотрим в этом подразделе, позволяют форматировать большие блоки текста. Они определяют параметры отображения и расположения текста, заключенного в них.

Сначала рассмотрим тег, напрямую отвечающий за параметры шрифта.

Тег ‹FONT›

Тег <FONT> задает параметры шрифта для текста внутри своего блока. Для форматирования предпочтительней использовать таблицы стилей, однако некоторые простые документы допускают и такое определение параметров текста.

Атрибуты тега <FONT> задают такие параметры текста, как шрифт, размер и цвет текста внутри тега.

За шрифт отвечает атрибут FACE, его значением должно быть название шрифта. Это название должно быть знакомо компьютеру пользователя, иначе будет использоваться шрифт по умолчанию. Для решения проблемы несоответствия или отсутствия шрифтов можно задать несколько допустимых типов, введя их через запятую в качестве значения атрибута FACE.

За размер шрифта отвечает атрибут SIZE. Размер шрифта задается в относительных размерах (например, 2 или 6). По умолчанию используется размер 3. Можно задать размер шрифта относительно остального текста. Для этого следует указать +, если нужно, чтобы размер шрифта на данном участке был больше основного текста, или –, если требуется обратное форматирование. После знака надо указать количество пунктов, на которое текст должен быть больше или меньше.

За цвет шрифта отвечает атрибут COLOR. Его значением должно быть либо ключевое слово, обозначающее имя цвета, либо код цвета в формате #RRGGBB. HTML-код цвета можно увидеть в большинстве графических редакторов.

В листинге 1.34 показан пример кода для определения параметров текста с помощью тега <FONT>.

Листинг 1.34. Использование тега ‹FONT›

```
<html>
<head>
<title>Тег FONT</title>
<body>
```

```
<font color="#FF0000" face="Verdana, Arial, Helvetica, sans-serif" size="+2">
Текст красного цвета с размером +2
</font><br />
<font color="#FF00FF" face="Verdana, Arial, Helvetica, sans-serif" size="-4">
Текст фиолетового цвета с размером -4
</font><br />
</body>
</html>
```

На рис. 1.22 вы видите результат обработки кода из листинга 1.34.

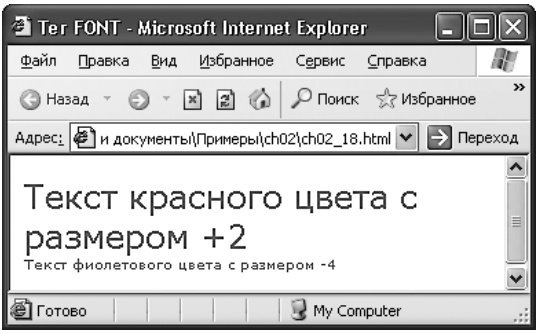


Рис. 1.22. Использование тега ‹FONT›

В листинге 1.34 задан список похожих шрифтов. Браузер просмотрит по порядку список и выведет текст тем шрифтом, который найдет первым. Последним вариантом является не шрифт, а семейство шрифтов sans-serif. Если браузер не найдет ни одного из перечисленных шрифтов, он возьмет известный ему шрифт из этого семейства.

Мы разобрались с тем, как можно задать параметры шрифта. Однако уже отмечалось, что этот метод не лучший, так как форматирование удобнее задавать с помощью таблиц стилей.

Тег ‹DIV›

Тег <DIV> служит для выделения больших блоков текста для форматирования с помощью таблиц стилей. Это означает, что вы включаете необходимый блок текста внутрь тега <DIV> и задаете его параметры либо в атрибуте STYLE, либо подключаете класс из таблицы стилей с помощью атрибута CLASS.

HTML вам позволит сделать лишь следующее: выровнять текст в блоке с помощью атрибута ALIGN и создать всплывающую подсказку для блока с использованием атрибута TITLE.

В листинге 1.35 представлен пример кода для выделения текста с помощью тега <DIV>.

Листинг 1.35. Использование тега ‹DIV›

```
<html>
<head>
<title>Тег DIV</title>
<body>
<div style="cursor:crosshair" align="center">
Текст внутри этого тега выровнен
по центру, а с помощью стилей
задается вид курсора при наве-
дении на этот блок.
</div>
</body>
</html>
```

На рис. 1.23 показан результат обработки кода из листинга 1.35.

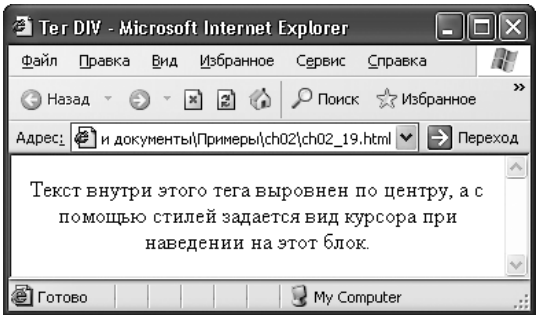


Рис. 1.23. Использование тега ‹DIV›

Тег ‹PRE›

Как форматировать текст, вы уже поняли. Но что делать, если имеется отформатированный нужным образом текст и вы не хотите ничего в нем менять?

Тег <PRE> служит для ввода текста без форматирования, то есть с сохранением пробелов и переносов строк.



ПРИМЕЧАНИЕ

Обычно в HTML несколько пробелов подряд воспринимаются как один пробел.

При этом текст выводится моноширинным шрифтом. Тег <PRE> часто используется для вывода кодов программ. В этот тег можно включать большинство тегов форматирования текста.

В листинге 1.36 приводится пример использования тега <PRE>.

Листинг 1.36. Использование тега ‹PRE›

```
<html>
<head>
<title>Тег PRE</title>
<body>
<pre>
```

```
Здесь
 можно
  расположить
   код
    программы
</pre>
</body>
</html>
```

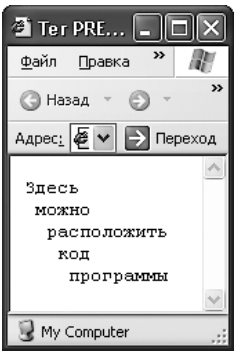


Рис. 1.24. Использование тега ‹PRE›

На рис. 1.24 вы видите, как в браузере выглядит текст, расположенный в теге <PRE>.

Тег ‹BLOCKQUOTE›

Это еще один тег, предназначенный для форматирования больших объемов текста. Он отвечает за правильность расположения цитат и размеров отступов.

Тег <BLOCKQUOTE> определяет выделенный текст как цитату и применяется для описания больших цитат. Он задает для текста отступы сверху, снизу и слева. В этот тег могут включаться теги форматирования текста.

В листинге 1.37 представлен пример выделения цитаты с использованием тега <BLOCKQUOTE>.

Листинг 1.37. Ввод больших цитат

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Тег BLOCKQUOTE</title>
<body>
<blockquote>
Здесь можно расположить важную и большую
цитату.
</blockquote>
</body>
</html>
```

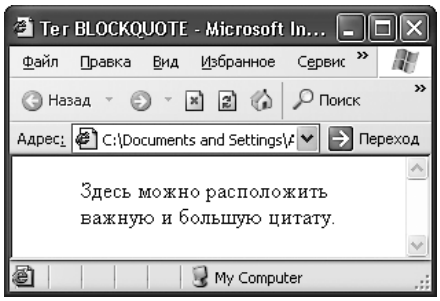


Рис. 1.25. Использование тега ‹BLOCKQUOTE›

На рис. 1.25 показано, как выглядит текст из тега <BLOCKQUOTE> в браузере.

Мы рассмотрели, как действует каждый вариант форматирования текста отдельно, однако HTML позволяет вкладывать теги, при этом объединяя их действие.

Вложение тегов

Язык HTML позволяет вкладывать теги форматирования друг в друга. При этом их действия суммируются. Например, если вложить тег <I> в тег <B>, то мы получим текст, написанный полужирным курсивом. Необходимо следить за правильным закрытием тегов: тег, открытый раньше, должен закрываться позже.

В листинге 1.38 представлены примеры правильного и неправильного вложения тегов.

Листинг 1.38. Вложение тегов

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Вложение тегов</title>
<body>
<del><b><i>Это неправильное вложение тегов</b></i></del><br />
<ins><b><i>Это правильное вложение тегов</i></b></ins>
</body>
</html>
```

Неправильную запись некоторые браузеры могут отображать некорректно.

Мы рассмотрели особенности структурного форматирования документа и форматирования самого текста, особенности логического и физического форматирования текста, принципы работы с большими блоками текста и с вложением тегов. Теперь вы можете задавать вид текста на своем сайте так, как вам угодно.

Работа с таблицами

В технической литературе и в других документах таблицы используют, чтобы расположить информацию в простом и понятном виде. Сейчас мы рассмотрим, как использовать таблицы в веб-сайтах.

В языке HTML таблицы используются в двух случаях: для представления данных, разбитых по строкам и столбцам, и как средство форматирования веб-страниц, в том

числе задания взаимного расположения элементов страницы. Ячейки таблицы могут содержать любые HTML-элементы: заголовки, списки, абзацы, фигуры, графику, а также элементы форм. Фактически весь сайт можно расположить в одной большой таблице.

Что такое таблица

Таблица состоит из ячеек, образующихся при пересечении строк и столбцов (рис. 1.26). Рассмотрим кратко элементы таблицы.

- Ячейка — это основной элемент таблицы. Ячейка формируется пересечением строки и столбца.
- Строка — это набор ячеек, расположенных одна за другой горизонтально.
- Столбец — это набор ячеек, расположенных одна над другой вертикально.
- Граница — это линия, которая окружает каждую ячейку и таблицу в целом.

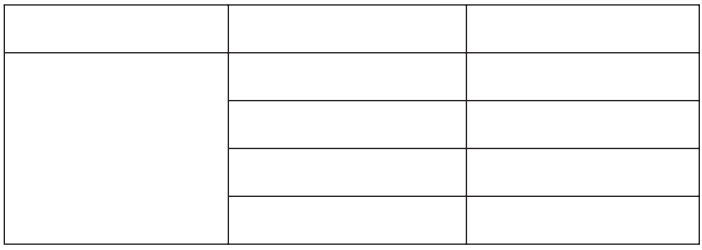


Рис. 1.26. Пример стандартной таблицы

В стандартной таблице каждая строка и каждый столбец содержат одинаковое количество ячеек. Однако таблицы могут очень сильно отличаться от стандартной. В таблице, изображенной на рис. 1.27, одна ячейка объединяет четыре строки, а другая — три столбца.

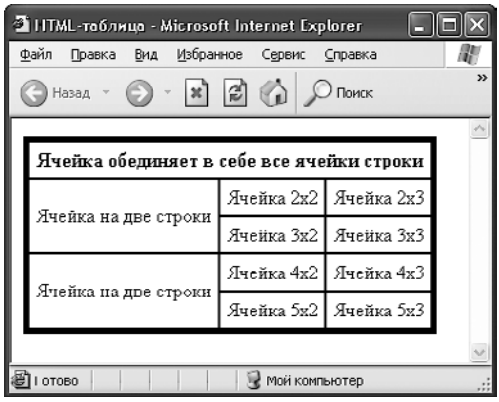


Рис. 1.27. Пример таблицы

Создание тела таблицы

Мы уже рассматривали, как создавать таблицы. Каждая таблица должна начинаться тегом `<table>` и завершаться тегом `</table>`:

```
<body>
    <table>
    </table>
</body>
```

Все прочие элементы таблицы должны быть вложенными в тег <table>.

Ячейки таблицы

Теперь приступим к созданию ячеек таблицы. Для начала нужно задать необходимое число строк, а затем разделить строки столбцами на ячейки. Количество пар тегов <tr> и </tr> определяет количество горизонтальных строк в таблице. Количество пар тегов <td> и </td>, расположенных между тегами соответствующей строки, определяет количество ячеек (столбцов) в пределах строки:

```
<body>
    <table>
    <tr><td>  </td><td>  </td></tr>
    <tr><td>  </td><td>  </td></tr>
    </table>
</body>
```

Тег <TH> позволяет создать заголовок — специальную ячейку, которая будет выделяться полужирным. Для этого текст заголовка помещают внутрь тега <TH>. Например, запись <TH> Текст заголовка </TH> приведет к форматированию текста внутри тега полужирным шрифтом.

Рассмотрим пример HTML-таблицы, которая состоит из трех строк и трех столбцов, причем ячейки первой строки являются заголовками соответствующих столбцов (листинг 1.39).

Листинг 1.39. Код простой HTML-таблицы, состоящей из трех строк и трех столбцов

```
<html>
<head>
<title>Простая HTML-таблица</title>
<head>
<body>
<table>
<caption>Это заголовок таблицы</caption>
```

```
<tr><th>Заголовок 1</th><th>Заголовок 2</th><th>Заголовок 3</th></tr>
<tr><td>Ячейка 2x1 </td><td>Ячейка 2x2 </td><td>Ячейка 2x3 </td></tr>
<tr><td>Ячейка 3x1 </td><td>Ячейка 3x2 </td><td>Ячейка 3x3 </td></tr>
</table>
</body>
</html>
```

На рис. 1.28 вы видите, как выглядит эта таблица в окне браузера.

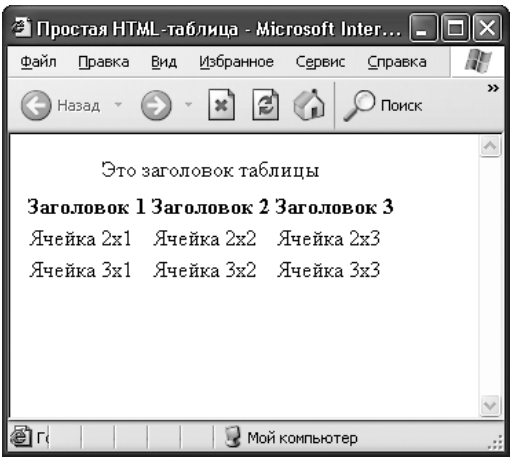


Рис. 1.28. Пример простой HTML-таблицы

Границы таблицы

В рассмотренном выше примере таблица и ячейки не имеют видимых границ. Граница таблицы создается при помощи атрибута `border` тега `<table>`. Ширина границы указывается в пикселах. Если атрибут `border` не указан, то таблица выводится без видимой рамки. Атрибут `bordercolor` позволяет задать цвет границы таблицы, указав код цвета. Добавим к уже созданной таблице границу черного цвета шириной четыре пиксела. Для этого заменим строку

```
<table>
```

на

```
<table border="4" bordercolor="#000000">
```

Границу таблицы (и другие элементы сайта) разные браузеры отображают немного по-разному. Опытные веб-программисты стараются следить, чтобы созданная веб-страница в различных браузерах выглядела одинаково. У каждого браузера свои капризы, поэтому создать сайт, в котором все везде выглядит одинаково, — задача сложная. На рис. 1.29 показывается, как эту таблицу с границей отображает браузер Internet Explorer.

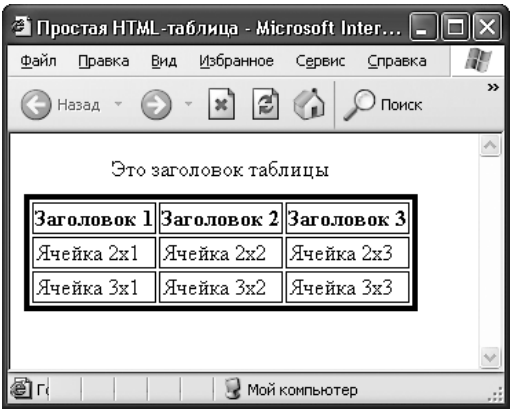


Рис. 1.29. Отображение таблицы с границей в окне Internet Explorer

На рис. 1.29 видно, что между границами ячеек и внешней границей таблицы есть свободное пространство. Оно легко регулируется с помощью атрибутов `cellspacing` и `cellpadding` тега `<table>`. Атрибут `cellspacing` определяет расстояние между ячейками таблицы (в пикселах), а атрибут `cellpadding` — между содержимым ячейки и ее границей.

В листинге 1.40 приводится код таблицы, у которой расстояние между содержимым ячеек и границей составляет 10 пикселов, а расстояния между ячейками нет.

Листинг 1.40. Код HTML-таблицы с нестандартными расстояниями между ячейками и границей

```
<html>
<head>
<title>Простая HTML-таблица</title>
<head>
<body>
<table border="4" bordercolor="#000000" cellspacing="0" cellpadding="10">
<caption>Это заголовок таблицы</caption>
<tr><th>Заголовок 1</th><th>Заголовок 2</th><th>Заголовок 3</th></tr>
<tr><td>Ячейка 2x1 </td><td>Ячейка 2x2</td><td>Ячейка 2x3</td></tr>
<tr><td>Ячейка 3x1 </td><td>Ячейка 3x2</td><td>Ячейка 3x3</td></tr>
</table>
</body>
</html>
```

На рис. 1.30 показывается вид этой таблицы в окне браузера.

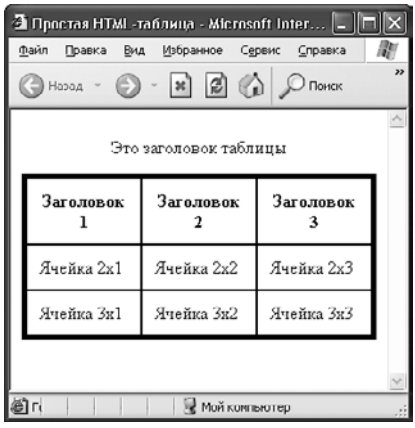


Рис. 1.30. Таблица с нестандартными расстояниями между ячейками и границей

Отображение внешней границы таблицы задается с помощью атрибута `frame`, а внутренними границами — с использованием атрибута `rules`. Атрибут `frame` может принимать следующие значения:

- `Above` — отображается только верхняя линия границы;
- `Below` — отображается только нижняя линия границы;
- `Box` или `Border` — отображается только внешняя граница таблицы;

- `Hsides` — отображаются только горизонтальные линии границы;
- `Lhs` — отображается только левая линия границы;
- `Rhs` — отображается только правая линия границы;
- `Void` — внешняя граница таблицы не отображается;
- `Vsides` — отображаются только вертикальные линии границы.

`Frame` и `rules` — это новые атрибуты в HTML, которые поддерживаются только последними версиями браузеров. Атрибут `rules` может иметь следующие значения:

- `All` — отображается граница вокруг каждой ячейки;
- `Cols` — отображаются только вертикальные границы между столбцами;
- `Groups` — отображаются только вертикальные границы между группами столбцов или горизонтальные границы между группами строк;
- `None` — границы между ячейками не отображаются;
- `Rows` — отображаются только горизонтальные границы между группами строк.

В листинге 1.41 приведен пример использования атрибутов `frame` и `rules`.

Листинг 1.41. Код HTML-таблицы с частичным отображением линий внешней и внутренней границ

```
<html>
<head>
<title>Простая HTML-таблица</title>
<head>
<body>
<table border="4" bordercolor="#000000" cellspacing="0" cellpadding="10"
frame= Hsides rules= Cols>
<caption>Склонение слова пиксел по падежам </caption>
<tr><th>Падеж</th><th>Пример   </th></tr>
<tr><td>Им. </td><td>пиксел </td></tr>
<tr><td>Рд. </td><td>пиксела    </td></tr>
<tr><td>Дт. </td><td>пикселу    </td></tr>
<tr><td>Вт. </td><td>пиксел </td></tr>
<tr><td>Тв. </td><td>пикселом   </td></tr>
<tr><td>Пр. </td><td>о пикселе  </td></tr>
</table>
</body>
</html>
```

На рис. 1.31 показано, как эта таблица выглядит в окне браузера.

Как вы видите, у этой таблицы отображаются горизонтальные линии внешних и вертикальные линии внутренней границ, так как именно эти значения мы задали атрибутам `frame` и `rules`.

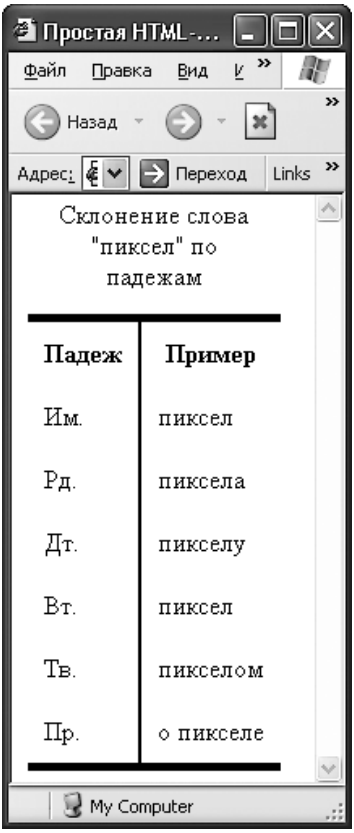


Рис. 1.31. HTML-таблица с частичным отображением линий внешней и внутренней границ

Ширина и высота таблицы и ячеек

Ширина таблицы задается атрибутом `width` тега `<table>`. Значение можно задавать как в абсолютных единицах (`width="250"`), так и в относительных (`width="80%"`). Например, задав ширину в 600 пикселов, можно быть уверенным, что таблица поместится в окне браузера при любом разрешении монитора. При относительном указании значения ширина задается в процентах от ширины окна браузера или от ширины ячейки другой таблицы, в которую вставлена эта таблица. Таким же образом задается высота таблицы при помощи атрибута `height`.

Все вышесказанное относится и к ячейкам таблицы. Не обязательно задавать размеры каждой ячейки. При изменении ширины или высоты какой-либо ячейки все соседние ячейки в пределах столбца будут отображаться с учетом нового значения.

При задании чрезмерно малых величин ширины и высоты таблицы браузер определяет минимальные значения, которые позволяют нормально отображать данные.

На рис. 1.32 и 1.33 изображаются таблицы одинакового содержания, но разной ширины и высоты.

Ширина первой таблицы составляет 80 % от размера окна браузера, а первый столбец этой таблицы занимает 50 % от всей ширины таблицы. Высота первой строки равна 100 пикселам. Вторая таблица квадратная, ширина стороны составляет 400 пикселов.

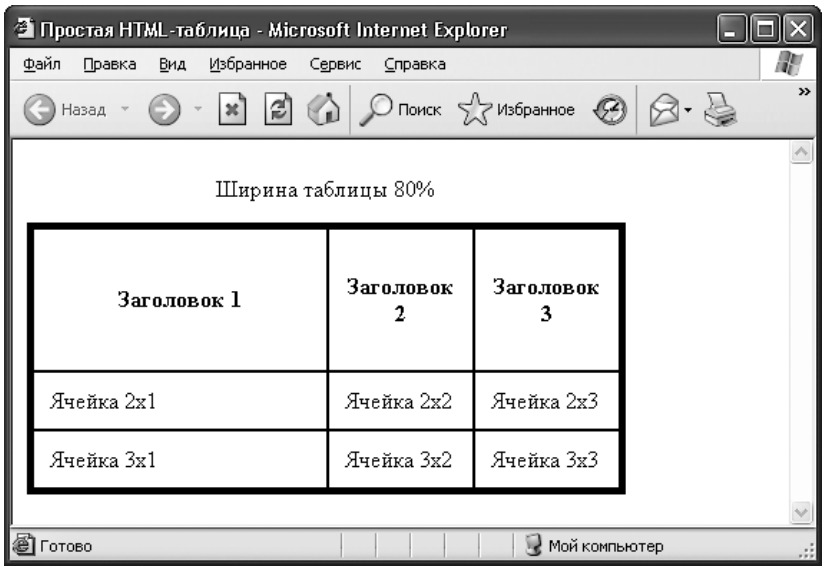


Рис. 1.32. Ширина таблицы составляет 80 % от размера окна браузера

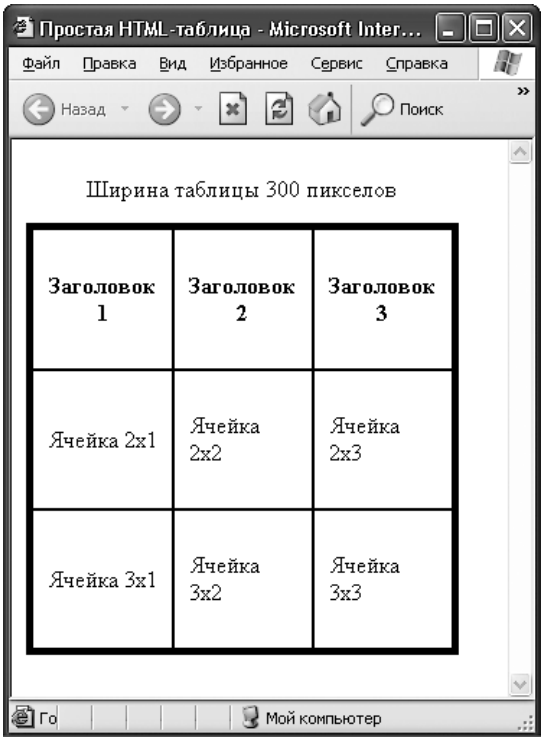


Рис. 1.33. Ширина и высота таблицы равна 300 пикселам

В листингах 1.42 и 1.43 приводятся коды страниц с этими таблицами.

Листинг 1.42. Код HTML-таблицы шириной 80 % от размера окна браузера

```
<html>
<head>
<title>Простая HTML-таблица</title>
<head>
<body>
<table border="4" bordercolor="#000000" cellspacing="0" cellpadding="10"
width="80%">
<caption>Ширина таблицы 80%</caption>
<tr><th height="100" width="50%">Заголовок 1</th>
    <th>Заголовок 2</th><th>Заголовок 3</th></tr>
<tr><td>Ячейка 2x1 </td><td>Ячейка 2x2</td><td>Ячейка 2x3</td></tr>
<tr><td>Ячейка 3x1 </td><td>Ячейка 3x2</td><td>Ячейка 3x3</td></tr>
</table>
</body>
</html>
```

Листинг 1.43. Код HTML-таблицы шириной 300 пикселов

```
<html>
<head>
<title>Простая HTML-таблица</title>
<head>
<body>
<table border="4" bordercolor="#000000" cellspacing="0" cellpadding="10"
width="300" height="300">
<caption>Ширина таблицы 300 пикселов</caption>
<tr><th>Заголовок 1</th><th>Заголовок 2</th><th>Заголовок 3</th></tr>
<tr><td>Ячейка 2x1 </td><td>Ячейка 2x2</td><td>Ячейка 2x3</td></tr>
<tr><td>Ячейка 3x1 </td><td>Ячейка 3x2</td><td>Ячейка 3x3</td></tr>
</table>
</body>
</html>
```

Группировка строк и столбцов

В стандарте HTML 4 появились теги для группировки (не объединения, а именно группировки) строк и столбцов таблицы. При этом группа строк и столбцов характеризуется некоторыми общими свойствами.

Для группировки столбцов таблицы служат теги `<colgroup>` и `<col>`. Тег `<colgroup>` создает структурную группу столбцов, которая выделяет множество логически однородных ячеек. К примеру, одна структурная группа может охватывать ячейки заголовков столбцов, а другая — ячейки, содержащие данные. Тег `<col>` предназначен для формирования неструктурных групп столбцов, которые делят таблицу на разделы, не имеющие отношения к структуре. Это удобно в том случае, когда не все столбцы содержат информацию одного типа.

Полезным атрибутом тегов `<colgroup>` и `<col>` является атрибут `span=n`, который распространяет свойства, заданные этими тегами на *n*-столбцов в группе.

```
<table>
<col span=2 style="color:red">
<tr><td> Этот текст будет красным цветом <td></tr>
<tr><td> И этот текст будет красным цветом <td></tr>
<tr><td> А этот текст будет черным цветом <td></tr>
</table>
```

Для группировки строк таблицы служат теги `<thead>`, `<tbody>` и `<tfoot>`. Данные теги существенно упрощают компоновку и форматирование таблиц. Для создания группы заголовков для строк таблицы используется тег `<thead>`. Этот тег допускается использовать в пределах таблицы только один раз. Для создания одной или нескольких групп строк таблицы, содержащих основные данные, применяется тег `<tbody>`. Тег `<tfoot>` позволяет создать группу строк для представления информации о суммах или итогах, располагаемую в нижней части таблицы. Этот тег также можно использовать в пределах таблицы только единожды. Вовсе не обязательно создавать группы строк таблицы всех трех типов:

```
<Thead>
<tr><th>    </th><th>    </th><th>    </th></tr>
</Thead>
</Tbody>
<tr><td>    </td><td>    </td><td>    </td></tr>
<tr><td>    </td><td>    </td><td>    </td></tr>
<Tbody>
```

Предположим, требуется создать таблицу, в которой внутренние линии не отображаются, а показывается только линия, отделяющая заголовки столбцов от основного текста (тела таблицы). Чтобы создать такую таблицу, нужно выполнить следующие действия.

1. Сгруппировать строки нужным образом.
2. Указать видимость границы между группами строк.

Выравнивание таблицы и содержимого ячеек

Для выравнивания элементов таблиц по горизонтали и вертикали в тегах `<table>`, `<tr>`, `<th>` и `<td>` используются атрибуты `align` и `valign`.

Атрибут `align` применяется ко всем тегам таблицы и определяет горизонтальное выравнивание во всей таблице:

- `<table align = left/right/center>` — общее горизонтальное выравнивание таблицы на странице по левому/правому краю или по центру;
- `<tr align = left/right/center>` — общее горизонтальное выравнивание элементов строки по левому/правому краю или по центру;
- `<th align = left/right/center>` — горизонтальное выравнивание заголовка таблицы по левому/правому краю или по центру (по умолчанию — по центру);
- `<td align = left/right/center/char>` — горизонтальное выравнивание данных в ячейке по левому/правому краю или центру по заданному символу (по умолчанию — по левому краю).

Выравнивание по заданному символу удобно, когда нужно, к примеру, выровнять несколько чисел по десятичной точке. В этом случае нужно добавить атрибут `char` и в его значении указать символ точки в кавычках.

Атрибут `valign` также применяется ко всем элементам таблицы и определяет вертикальное выравнивание по всей таблице:

- `<table valign = bottom/middle/top>` — общее вертикальное выравнивание элементов таблицы по низу/центру/верху (по умолчанию — по центру);
- `<tr valign = bottom/middle/top>` — общее вертикальное выравнивание элементов строки по низу/центру/верху;
- `<th valign bottom/middle/top>` — вертикальное выравнивание заголовка таблицы по низу/центру/верху;

- `<td valign = bottom/middle/top>` — вертикальное выравнивание данных в ячейке по низу/центру/верху.

В листинге 1.44 приводится пример использования вышеописанных атрибутов выравнивания таблицы и содержимого ячеек в различных комбинациях.

Листинг 1.44. Пример использования атрибутов выравнивания таблицы и содержимого ячеек

```
<html>
<head>
<title>Простая HTML-таблица</title>
<head>
<body>
<table border="4" bordercolor="#000000" cellspacing="0" cellpadding="0"
width="400" height="150" align=center>
<caption>Наименование товара</caption>
<tr><th>Товар </th><th>Код</th><th>Количество</th><th>Цена    </th> </tr>
<tr valign=bottom align=center>
<td>Клей  </td><td>028</td><td>190 шт  </td><td>12,2 руб</td></tr>
<tr valign=bottom align=center>
<td>Скотч </td><td>058</td><td>120 шт  </td><td>4,6 руб </td></tr>
<tr valign=bottom align=center>
<td>Ластик</td><td>986</td><td>100 шт  </td><td>2,3 руб </td></tr>
</table>
</body>
</html>
```

На рис. 1.34 показывается, как эта таблица выглядит в окне браузера.

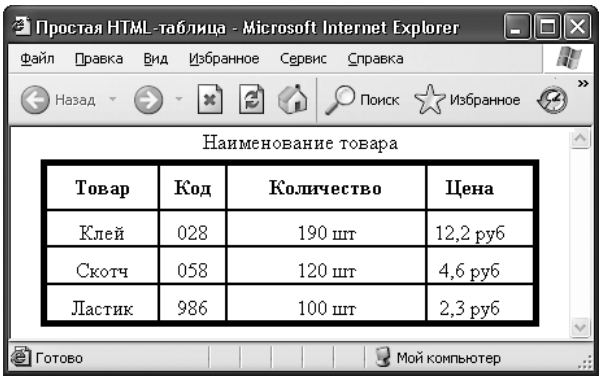


Рис. 1.34. Пример использования атрибутов выравнивания таблицы и содержимого ячеек

Объединение ячеек таблицы

В практике встречается большое количество таблиц, в которых одна ячейка объединяет несколько ячеек по высоте или

ширине. В HTML ячейки объединяются при помощи атрибутов `colspan` и `rowspan`. Атрибут `colspan` определяет количество ячеек, на которые простирается данная ячейка по горизонтали, а `rowspan` — по вертикали.

На рис. 1.35 вы видите таблицу с объединенными ячейками.

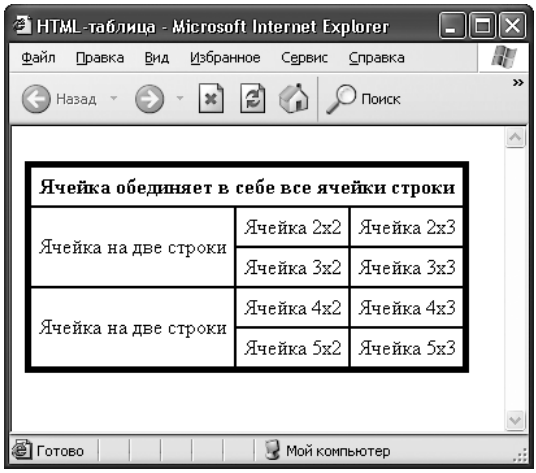


Рис. 1.35. Таблица с объединенными ячейками

Заголовок таблицы находится в ячейке, объединяющей в себе все три ячейки строки. Таблица содержит еще две ячейки, каждая из которых объединяет две ячейки по вертикали. Код этой таблицы приводится в листинге 1.45.

Листинг 1.45. Пример использования атрибутов объединения ячеек

```
<html>
<head>
<title>HTML-таблица</title>
<head>
<body>
<table border="4" bordercolor="#000000" cellspacing="0" cellpadding="5" >
<tr align=center><th colspan=3>Ячейка объединяет в себе все ячейки строки
</th></tr>
<tr align=center><td rowspan=2>Ячейка на две строки </td>
    <td>Ячейка 2x2</td><td>Ячейка 2x3</td></tr>
<tr align=center>  <td>Ячейка 3x2</td><td>Ячейка 3x3</td></tr>
<tr align=center><td rowspan=2>Ячейка на две строки</td>
    <td>Ячейка 4x2</td><td>Ячейка 4x3</td></tr>
```

```
<tr align=center>  <td>Ячейка 5x2</td><td>Ячейка 5x3</td></tr>
</table>
</body>
</html>
```

Установка фона или рисунка

В HTML можно устанавливать фоновый цвет или рисунок. Атрибут `bgcolor` позволяет изменять цвет содержимого ячейки, строки, столбца, группы строк, группы столбцов, а также таблицы целиком.

С помощью конструкции `<td bgcolor="#000000">`, чередующейся с конструкцией `<td></td>`, можно создать таблицу, изображенную на рис. 1.36.

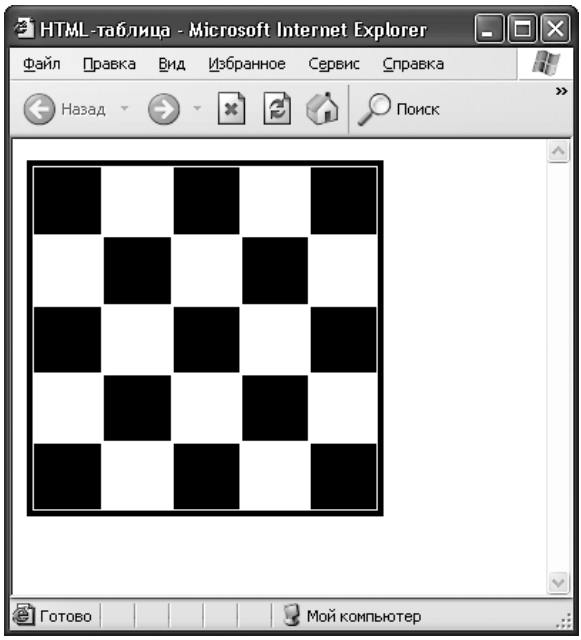


Рис. 1.36. Пример использования атрибута bgcolor

Атрибут `background` позволяет задать графический фон ячейки или таблицы целиком, при этом нужно указать путь к изображению:

```
<td background="img/fon.gif">
```

Если габариты изображения превышают размеры ячейки или таблицы, для которых оно предназначено, браузер обрезает изображение.

Создание вложенных таблиц

Иногда необходимо создать ячейки, границы которых не должны совпадать (рис. 1.37).

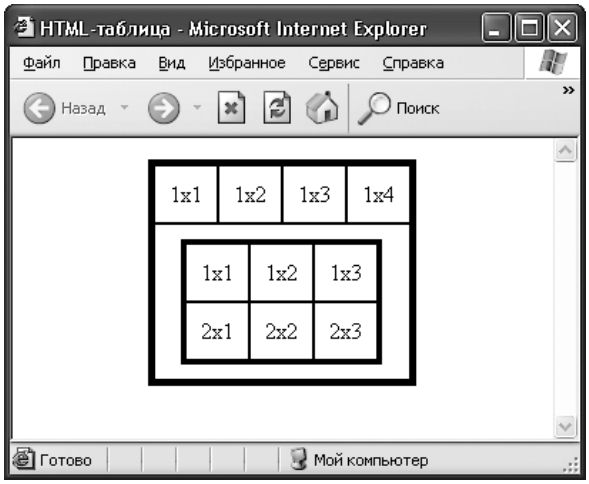


Рис. 1.37. Вложенная HTML-таблица

В этом случае атрибут `colspan` бессилен.

Команды, управляющие созданием и отображением таблиц, допускают вложение одного тега `<table>` в другой, поэтому в ячейке можно создать новую таблицу с независимой структурой.

Использование вложенных таблиц — один из самых распространенных приемов при дизайне сайта. На рис. 1.38 вы можете увидеть, как использовались таблицы при создании сайта www.ya.ru.

Таблицы являются важнейшей составляющей любой веб-страницы. Умелое обращение с ними позволяет жестко связать текстовые блоки документа с графикой и другими объектами. Использование таблиц позволяет странице загружаться быстрее, а также разбивать большие рисунки на несколько маленьких. Чтобы не нарушить целостность рисунка, его части следует расположить в ячейках таблицы таким образом, чтобы они образовывали целый рисунок.

Применение таблиц позволяет разделить HTML-страницу на функциональные части: заголовок страницы, место для рекламы, панель инструментов и т. д. Подобное деление страницы позволит быстро создать индивидуальный «скелет» сайта, который в дальнейшем можно использовать для создания других страниц.

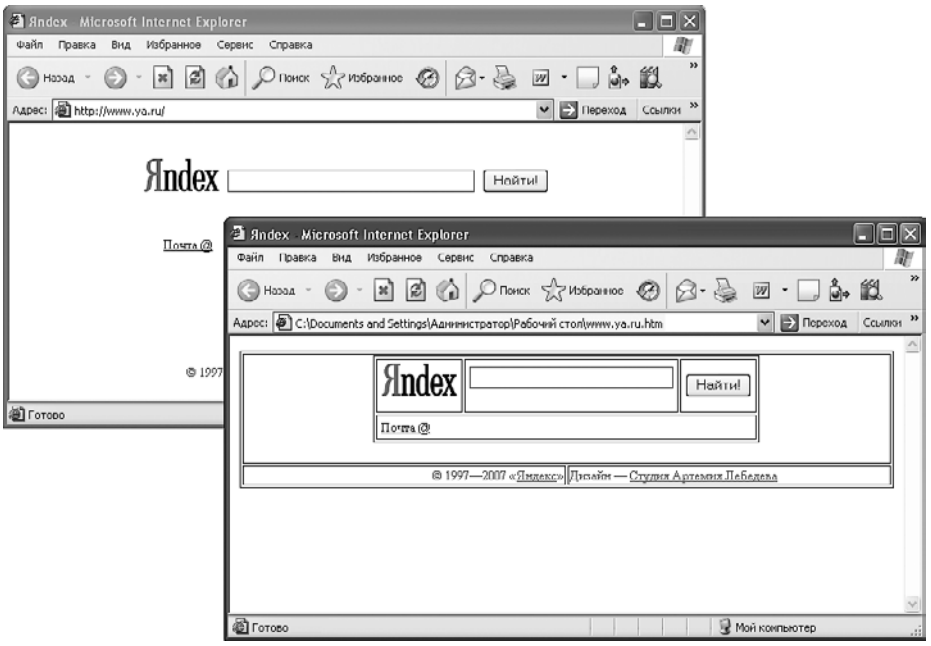


Рис. 1.38. Использование таблиц в дизайне сайтов

Изображения

Мы уже говорили, что изображения добавляются в HTML-код с помощью тега `<IMG>`. Рассмотрим его подробнее.

Встраивание изображений

Современные скорости соединений позволяют размещать большое количество графики на странице. Однако надо знать меру: не стоит злоупотреблять рисунками, в то же время сайт не должен выглядеть пустым. Грамотное и уместное применение изображений поможет сделать сайт красивым, интересным и удобным в использовании.

Для встраивания изображений в HTML-документ применяется тег `<IMG>`. Его обязательный атрибут — атрибут `SRC`, значением которого является адрес встраиваемого изображения.

Простейший вариант записи для включения картинки выглядит следующим образом:

```
<IMG SRC="image.jpg">
```

При такой записи размер картинки на экране будет соответствовать ее реальному размеру.

ПРИМЕЧАНИЕ



Адрес изображения может указываться либо полностью, например, когда картинка находится на другом сервере (http://www.mypage.ru/IMG/myfoto.jpg), либо относительно местоположения вашего документа, например, если картинка лежит во вложенной папке (IMG/myfoto.jpg). Во втором случае для указания директории, находящейся «выше» в иерархии каталогов, используются символы ../. Переход из папки DOC, находящейся в одном каталоге с папкой IMG, будет выглядеть так: ../IMG/myfoto.jpg.

Размер изображения

Если необходимо изменить размер изображения, то лучше воспользоваться специальными программами. В случае, когда размер изображения изменяется незначительно, допустимо применять атрибуты тега <IMG>.

Размер картинки изменяется с помощью атрибутов WIDTH и HEIGHT. Их значения можно указать в пикселах или процентах от размера окна (в этом случае после размера ставится знак %).

Вы можете указать только один атрибут, и тогда второй будет вычисляться автоматически таким образом, чтобы сохранить пропорции рисунка. Искажение пропорций тоже не приводит к улучшению качества изображения, как видно на рис. 1.39.

В листинге 1.46 представлен код встраивания изображения с заданными размерами (результат его обработки вы видите на рис. 1.39).

Листинг 1.46. Задание размеров изображения

```
<html>
<head>
<title>Встраивание изображения</title>
</head>
<body>
```

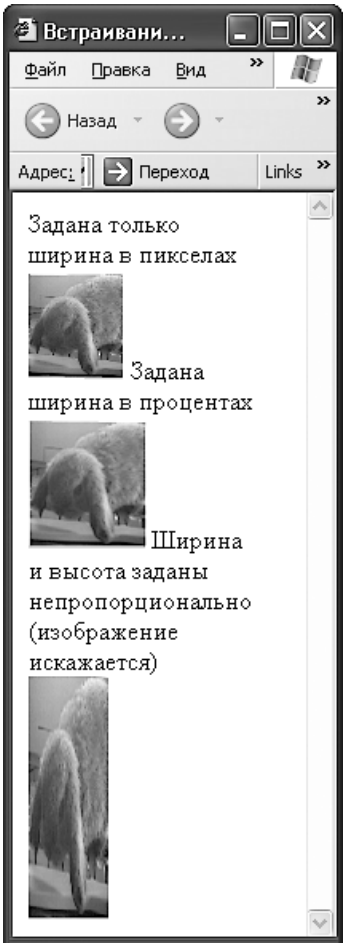


Рис. 1.39. Задание размеров изображения

```
Задана только ширина в пикселах <br />
<img src="image.jpg" width="60" />
Задана ширина в процентах<br />
<img src="image.jpg" width="50%" />
Ширина и высота заданы непропорционально (изображение искажается)<br />
<img src="image.jpg" width="50" height="150" />
</body>
</html>
```

В листинге 1.46 для первого изображения задается только ширина, а высота вычисляется автоматически, для второго изображения размер определяется в процентах, а для третьего изображения устанавливаются неверные размеры, которые искажают картинку.

Если изменять размеры не нужно, то следует задать размеры картинки, соответствующие реальности, или вовсе их не указывать. Это позволит избежать искажений и ускорит обработку картинки браузером.



СОВЕТ

Если вам нужно значительно изменить размер картинки, то используйте специальные программы для работы с изображениями. Они проделают эту операцию, исказив картинку минимально.

Помимо размеров картинки, на внешний вид сайта влияет расположение изображений.

Выравнивание изображения

За расположение изображения на странице отвечает атрибут тега `<IMG>` — `ALIGN`. Он позволяет выравнивать изображения с правой, левой стороны окна или относительно элементов строки.

Атрибут `ALIGN` может принимать много значений, которые позволяют установить картинку именно там, где надо. Возможные значения этого атрибута приводятся ниже.

Горизонтальное выравнивание:

- `left` — по левому краю;
- `right` — по правому краю.

Вертикальное выравнивание:

- `top` — выравнивание верхней границы картинки по самому высокому элементу строки;
- `texttop` — выравнивание верхней границы картинки по самому высокому элементу текста;
- `middle` — середина изображения выравнивается по базовой линии строки;
- `absmiddle` — середина изображения выравнивается по середине строки;
- `baseline` и `bottom` — нижняя граница изображения выравнивается по базовой линии строки;
- `absbottom` — нижняя граница изображения выравнивается по нижней границе текущей строки.

ПРИМЕЧАНИЕ



Базовая линия строки — это линия, на которой располагаются все элементы. Некоторые буквы выступают за эту линию, например буква «р». Ее выносной элемент заканчивается ниже базовой линии и является самым нижним элементом строки. Заглавные буквы, наоборот, выступают над этой линией.

В листинге 1.47 приводятся примеры выравнивания картинок относительно текста по вертикали.

Листинг 1.47. Выравнивание картинки по вертикали

```
<html>
<head>
<title>Встраивание изображения</title>
</head>
<body>
<img src="image.jpg" width="70" align="top"/>Выравнивание по самому верхнему элементу в строке<br />
<img src="image.jpg" width="70" align="absbottom"/>Нижняя граница изображения выравнивается по нижней границе текущей строки<br />
<img src="image.jpg" width="70" align="bottom"/>Нижняя граница изображения выранивается по базовой линии строки<br />
<img src="image.jpg" width="70" align="middle"/>Середина изображения выравнивается по базовой линии строки<br /> </body>
</html>
```

На рис. 1.40 представлен результат обработки кода из листинга 1.47. Очевидна разница между выравниваниями по базовой линии и по границам строки.

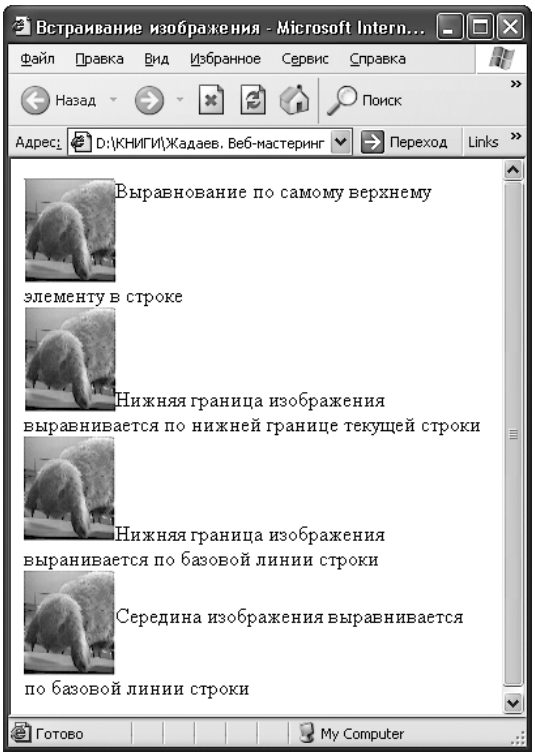


Рис. 1.40. Выравнивание картинки по вертикали

С выравниваниями по горизонтали дело обстоит проще. Код для выравнивания картинки по горизонтали приводится в листинге 1.48.

Листинг 1.48. Выравнивание картинки по горизонтали

```
<html>
<head>
<title>Встраивание изображения</title>
</head>
<body>
<img src="image.jpg" width="100" align="right" />
Кролики — потрясающие животные, особенно декоративные. Многие люди заводят их дома вместо кошек или собак, потому что они менее прихотливы. Кролики — чистоплотные животные, и их хозяева не испытывают проблем с воспитанием. Кормить этих зверюшек просто: овощи, сено и немного специального корма. Следите, чтобы клетка была чистой и в ней всегда лежали свежие опилки. Если вы рискуете выпускать зверя на прогулки по дому, то следите за проводами. Кролик может решить, что провода — это сено, и перегрызть их. </html>
```

На рис. 1.41 представлен результат обработки браузером кода из листинга 1.48. Как вы видите, текст обтекает картинку слева.

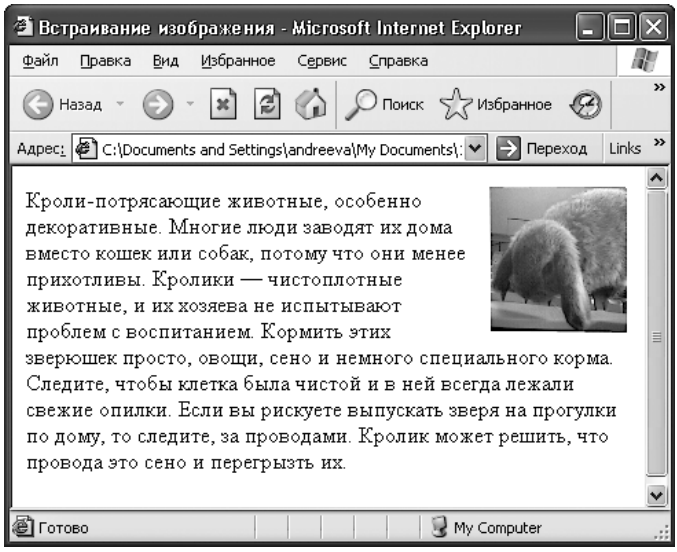


Рис. 1.41. Выравнивание картинки по горизонтали

При горизонтальном выравнивании текст плотно обтекает картинку. На внешний вид страницы весьма существенно влияет расстояние между текстом и изображением.

Чтобы отодвинуть текст от картинки, используются атрибуты `HSPACE` и `VSPACE` тега `<IMG>`. Они задают расстояние в пикселах между изображением и текстом по горизонтали и вертикали.

Рамка вокруг изображения

Изображение можно отделить от текста рамкой. Делается это с помощью атрибута `BORDER`, значение которого указывает ширину рамки в пикселах.

В листинге 1.49 задается ширина рамки (`border`) и отступы от текста (`vspace`, `hspace`) в 10 пикселов, а слева и справа выставлены разумные расстояния — по 50 пикселов.

Листинг 1.49. Задание рамки вокруг изображения

```
<html>
<head>
<title>Встраивание изображения</title>
```

```
</head>
<body>
<img src="image.jpg" width="100" align="right" vspace="10" hspace="10"
border="20"/>Кролики – потрясающие животные, особенно декоративные. Многие
люди заводят их дома вместо кошек или собак, потому что они менее прихот-
ливы. Кролики – чистоплотные животные, и их хозяева не испытывают проблем
с воспитанием. Кормить этих зверюшек просто: овощи, сено и немного специаль-
ного корма. Следите, чтобы клетка была чистой и в ней всегда лежали свежие
опилки. Если вы рискуете выпускать зверя на прогулки по дому, то следите за
проводами. Кролик может решить, что провода – это сено, и перегрызть их.
</body>
</html>
```

На рис. 1.42 видно, что изображение с такими параметрами выглядит намного лучше, чем в предыдущем примере, к тому же текст воспринимается гораздо проще.

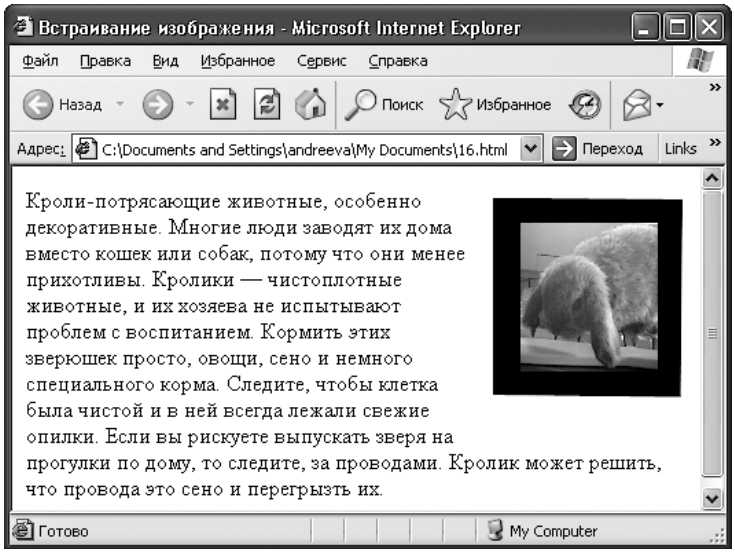


Рис. 1.42. Рамка вокруг изображения

Альтернативный текст

Иногда возникают ситуации, когда картинка не загружается, например, когда пользователь отключил загрузку картинок или «упал» сервер, на котором она расположена. Если размеры картинки указаны в явном виде, то посетитель имеет о них представление. Однако зачем ему эти размеры, если он даже не предполагает, что изображается на рисунке?

В таких случаях может помочь атрибут ALT. Его значением является текст, который будет показываться при наведении курсора на картинку или если картинка не загружается. Это дает возможность пользователю, который не видит картинок, решить, хочет ли он вообще их увидеть.

В листинге 1.50 приводится пример кода для задания альтернативного текста.

Листинг 1.50. Альтернативный текст

```
<html>
<head>
<title>Встраивание изображения</title>
</head>
<body>
<img src="image.jpg" width="100" align="right" vspace="50" hspace="50"
alt="Фотография кошки"/>
</body>
</html>
```

На рис. 1.43 вы видите результат обработки кода из листинга 1.50 при условии, что изображение загрузилось.

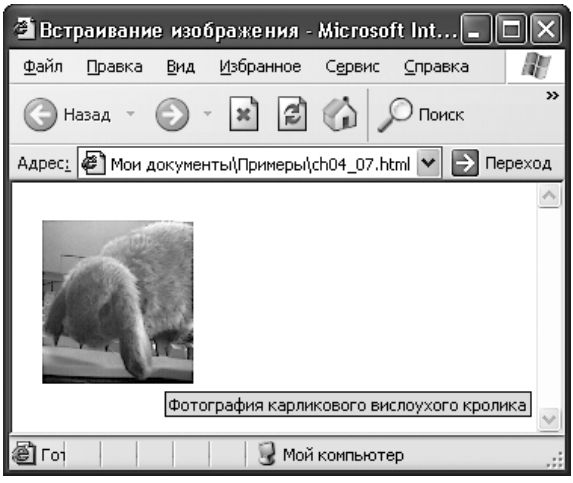


Рис. 1.43. Альтернативный текст при загруженном рисунке

Альтернативный текст появляется при наведении указателя мыши на картинку.

На рис. 1.44 вы видите результат обработки кода из того же листинга, но в случае, когда изображение не загрузилось.

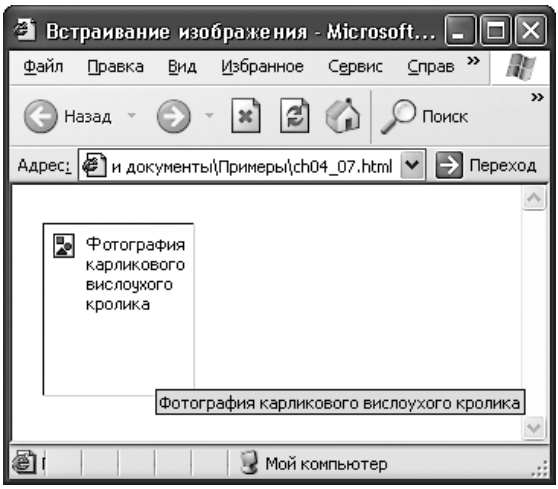


Рис. 1.44. Альтернативный текст при незагруженном рисунке

Вместо картинки отображается поясняющий текст, а при наведении курсора на изображение появляется всплывающая подсказка.

Формы

Сейчас мы рассмотрим элемент, который помогает организовать контакт пользователя с авторами сайта, — формы. Они позволяют получать информацию о посетителях. Любая информация, введенная в форму, может быть отправлена создателю сайта.

Формы напоминают анкеты: в них можно создавать поля, в которые пользователь сам вводит любую информацию, и поля-списки, где ему следует выбрать какой-либо ответ из предложенных автором вариантов. Возможности форм по группировке и представлению информации позволяют разместить ее наилучшим образом.

Главное при создании форм — следить за тем, чтобы поля были удобны для ответов посетителей на ваши вопросы.

Техническая функция форм состоит в том, что они передают данные от пользователя к удаленному серверу. Это позволяет создавать системы регистрации пользователей, форумы и гостевые книги. Для получения и обработки данных форм используются языки веб-программирования (PHP, Perl), на которых мы остановимся позже. А сейчас рассмотрим, как оформляются формы на языке HTML.

Тег ‹FORM›

Основой любой формы является тег <FORM>, который требует закрывающего тега. В одну форму можно включать поля, объединенные логически, так как одновременно на сервер можно отправлять данные только из одной формы.

Главным атрибутом этого тега является атрибут ACTION, который указывает обработчик данных для формы. *Обработчик* — это файл, в котором описывается, что нужно делать с данными формы. Сам обработчик может быть написан на любом языке, например PHP. В листинге 1.51 файл обработки находится по адресу test.php, именно ему и будут передаваться все данные, введенные в форму.

Листинг 1.51. Атрибут ACTION тега ‹FORM›

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php">
</form>
</body>
</html>
```

Если значение атрибута ACTION не задается, то страница перезагрузится, а все элементы формы примут значения, заданные по умолчанию. Следовательно, такая форма окажется абсолютно бесполезной. Значением этого атрибута является URL-путь к файлу-обработчику на сервере.

Если написать программу-обработчик формы невозможно, то можно сделать так, чтобы заполненная посетителем форма приходила на вашу почту. Для этого в качестве значения атрибута ACTION следует указать электронный адрес, на который должна приходить форма. При этом адрес должен начинаться с ключевого слова mailto:. Для корректной обработки данных в этом случае необходимо указать значение атрибута enctype="text/plain".

При отправке формы откроется почтовый клиент, установленный на компьютере пользователя, и появится предложение отправить данные формы в письме. В листинге 1.52 приводится код, который может использоваться при отправке данных по электронной почте.

Листинг 1.52. Отправка данных формы по электронной почте

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="mailto:test@test.ru" enctype="text/plain">
</form>
</body>
</html>
```

В письмо будет вложен текстовый файл. В этом файле названиям полей формы будут присвоены значения, которые ввел посетитель. Это значит, что вы получите текст следующего вида:

```
название_поля=значение_введенное_посетителем.
```

Еще одним обязательным атрибутом является атрибут METHOD. Этот атрибут задает способ передачи формы. При выборе метода отправки формы надо ориентироваться на данные, которые следует отправить вместе с ней. При этом в большей мере нужно ориентироваться на объем данных.

Существует два основных метода отправки формы: GET и POST. Рассмотрим область применения каждого из этих методов.

- **GET.** Это наиболее распространенный метод, используемый для получения данных с сервера. При использовании метода GET данные на сервер передаются через адресную строку браузера. Этот метод также вызывается, когда вы вводите адрес сайта в адресной строке или щелкаете кнопкой мыши на ссылке. При формировании запроса к серверу все переменные и их значения формируют последовательность вида www.mytestserver.ru/form.php?var1=1&var2=2, то есть имена и значения переменных присоединяются к адресу сервера после знака ? и разделяются знаком &.
- **POST.** Этот метод применяется для отправки данных на сервер. Отправляемые по этому методу данные не видны в заголовке URL, так как они содержатся в теле сообщения. Именно поэтому метод POST подходит для отправки информации больших объемов и файлов.

В листинге 1.53 для отправки формы используется метод POST, поскольку вместе с формой передаются и файлы.

Листинг 1.53. Атрибут METHOD тега ‹FORM›

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php" enctype="multipart/form-data" method="post">
</form>
</body>
</html>
```

Итак, данная форма готова для передачи файлов и информации больших объемов.

Создание полей формы

Форма может состоять из очень многих полей, поэтому следует правильно выбрать поле для представления определенной информации. Поля имеют схожие функции, разница состоит лишь в удобстве применения каждого отдельного поля для конкретной ситуации. Рассмотрим, какие виды полей используются в различных ситуациях.

Общие атрибуты тега ‹INPUT›

Тег <INPUT> позволяет создавать большинство полей для формы. Атрибуты тега различаются в зависимости от поля, для формирования которого используется этот тег.

Начнем с главного атрибута, который задает тип поля, — TYPE. В табл. 1.8 описывается, какие значения атрибута соответствуют определенным типам полей.

Таблица 1.8. Значения атрибута TYPE

Значение	**Какому типу поля соответствует**
text	Поле для ввода текста
password	Поле для ввода пароля
button	Кнопка
file	Поле для добавления файла
image	Поле-картинка
checkbox	Флажок

Продолжение ⇩

Таблица 1.8 *(продолжение)*

Значение	Какому типу поля соответствует
radio	Переключатель
reset	Кнопка очистки формы
submit	Кнопка отправки формы
hidden	Скрытое поле

Еще одним обязательным атрибутом является атрибут NAME, его значение задает уникальное имя поля. Введенная пользователем информация будет передаваться обработчику в качестве значения переменной с именем поля. Этот атрибут имеет значение для программистов, поэтому имя должно быть логичным.

Иногда требуется запретить пользователю изменение данных в поле, например, чтобы он не смог поменять ранее введенную информацию. Для этого нужно добавить атрибут DISABLED.

Атрибут VALUE задает значение, которое будет передаваться обработчику. По сути, он определяет данные, введенные в поле. Для текстовых полей этот атрибут устанавливает значение по умолчанию. Для полей, предоставляющих возможность выбора, этот атрибут задает значение, которое будет передаваться обработчику при выборе конкретного пункта. В этом случае важно, чтобы значения данного параметра логически соответствовали пунктам, из которых можно выбирать.

Текстовые поля

Поля для ввода текста в одну строку удобны при вводе имени, пароля и любой другой короткой информации.

Для создания текстового поля устанавливается значение атрибута TYPE="text". Вместе с этим атрибутом есть смысл использовать атрибут MAXLENGHT, значение которого указывает максимальное число символов, которые можно ввести в данную строку.

Можно также задать длину поля ввода с помощью атрибута SIZE. Следить за длиной поля важно, так как при вводе строки большой длины пользователю удобно видеть ее целиком. И наоборот, нет необходимости оставлять большое поле, если нужно ввести один или два символа — это окажется неудобным. Поэтому над размерами поля ввода текста следует подумать.

В листинге 1.54 приводится пример создания полей для ввода основной информации о посетителе. Для всех полей заданы значения по умолчанию, которые поясняют, какую информацию следует ввести.

Листинг 1.54. Создание поля для ввода текста

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php" enctype="multipart/form-data" method="post" name="test"
target="_blank">
Имя: <input type="text" name="Reg_name" value="Ваше имя" /><br />
Фамилия: <input type="text" name="Reg_surname" value="Ваша фамилия"  /><br />
Отчество: <input type="text" name="Reg_patronymic" value="Ваше отчество"
/><br />
Введите логин: <input type="text" name="Reg_login" maxlength="12" value="<
10 знаков" size="14" /><br />
</form>
</body>
</html>
```

На рис. 1.45 вы видите результат обработки кода из листинга 1.54.

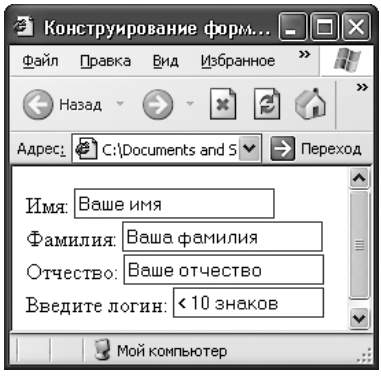


Рис. 1.45. Текстовые поля для ввода данных о пользователе

Чтобы создать поле для ввода пароля, нужно задать значение атрибута `TYPE="password"`. Особенность этого поля состоит в том, что все введенные буквы закрываются звездочками (или точками, в зависимости от системы) и подглядеть вводимый пароль невозможно.

После ввода информации о себе пользователь должен отправить ее на сервер. Для этого служит инструмент форм, который мы рассмотрим далее.

Кнопки

При создании формы без кнопок не обойтись: с их помощью можно очистить и отправить форму. Мы можем создавать разные кнопки — в зависимости от того, что требуется сделать с формой.

Чтобы создать обычную кнопку, присваиваем атрибуту TYPE тега `<INPUT>` значение `"button"`. Если надо создать кнопку, которая будет отправлять форму обработчику,

устанавливаем значение `TYPE="submit"`, а если кнопку, очищающую форму, — `TYPE="reset"`.

Значение атрибута `VALUE` для кнопок задает надпись на кнопке.

В примере из листинга 1.55 приводится код, с помощью которого создаются кнопки с разными надписями.

Листинг 1.55. Создание кнопок

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php" enctype="multipart/form-data" method="post" name="test"
target="_blank">
Надпись по умолчанию:<br />
<input name="submit" type="submit"/><input name="reset" type="reset"/><br>
Заданные надписи:<br />
<input name="submit" type="submit" value="Отправить"/><input name="reset"
type="reset" value="Очистить"/><br />
<input name="submit" type="submit" value="Отправить заполненную форму"/><input
name="reset" type="reset" value="Стереть все введенные данные"/><br />
</form>
</body>
</html>
```

На рис. 1.46 представлен результат обработки кода из листинга 1.55.

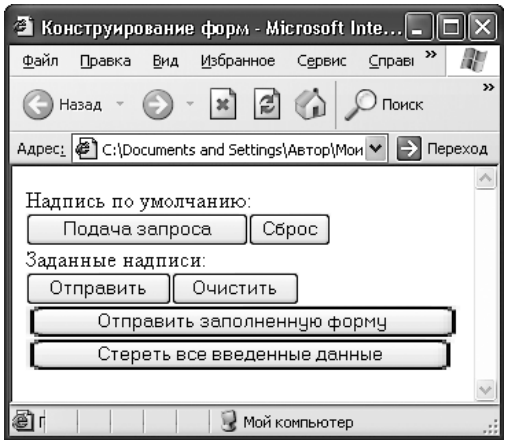


Рис. 1.46. Кнопки с разными надписями

Как вы видите, надписи по умолчанию могут быть не совсем понятны посетителю сайта, причем они зависят от системы компьютера пользователя.

Переключатели

Если вы задаете посетителю вопрос и уверены, что предоставили все возможные варианты ответа на него, или

же хотите, чтобы посетитель выбрал только один из предложенных вами вариантов, то используйте переключатели. Обязательно убедитесь, что действительно предоставили пользователю все возможные варианты ответов.

Для создания переключателей установите значение `TYPE="radio"`. Текст пункта введите рядом с тегом `<INPUT>`, за его пределами. Особенность переключателей состоит в том, что можно выбрать только один элемент из группы. Для всех переключателей, принадлежащих одной группе, нужно установить одинаковое значение атрибута `NAME`. Атрибут `VALUE` в этом случае содержит значение, которое будет передаваться обработчику (если пользователь выберет этот пункт).

При создании переключателей можно использовать атрибут `CHECKED`, который задает элемент, выбранный по умолчанию.

В листинге 1.56 представлен пример создания различных вариантов выключателей.

Листинг 1.56. Создание переключателей

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php" enctype="multipart/form-data" method="post" name="test"
target="_blank">
Ваш пол: <input name="Reg_Gender" type="radio" value="man" />Мужской
<input name="Reg_Gender" type="radio" value="woman" />Женский
<input name="Reg_Gender" type="radio" value="dontanswer" checked />Не хочу
отвечать<br />
Ваш возраст: <input name="Reg_Age" type="radio" value="before18" />до 18
<input name="Reg_Age" type="radio" value="1825" />18-25
<input name="Reg_Age" type="radio" value="2535" />25-35<br />
<input name="Reg_Age" type="radio" value="3550" />35-50
<input name="Reg_Age" type="radio" value="after50" />старше 50<br />
</form>
</body>
</html>
```

На рис. 1.47 вы видите результат обработки кода из листинга 1.56.

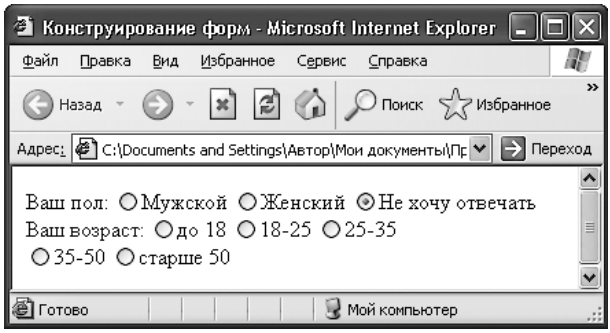


Рис. 1.47. Переключатели

Рассмотрим элемент, который позволяет посетителю выбрать несколько вариантов ответов из предложенных.

Флажки

Флажки напоминают переключатели тем, что дают пользователю возможность выбирать из предложенных вами вариантов. Главным отличием является то, что посетитель может выбрать сразу несколько пунктов, сами же пункты обозначаются квадратиками, а не кружочками.

В группы эти элементы объединяются таким же образом — путем присвоения одинакового имени всем полям. Для флажков также можно использовать атрибут `CHECKED`, здесь его можно установить для нескольких пунктов.

В листинге 1.57 приводится пример создания опроса с флажками, в котором задаются несколько выбранных по умолчанию вариантов ответа.

Листинг 1.57. Создание флажков

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php" enctype="multipart/form-data" method="post" name="test"
target="_blank">
Увлечения: <input name="Reg_hobby[]" type="checkbox" value="sing" />Петь
<input name="Reg_hobby[]" type="checkbox" value="float" />Плавать
```

```
<input name="Reg_hobby[]" type="checkbox" value="walk" />Гулять
<input name="Reg_hobby[]" type="checkbox" value="work" checked />Работать<br />
<input name="Reg_hobby[]" type="checkbox" value="drive" checked />Кататься на машине
<input name="Reg_hobby[]" type="checkbox" value="cook" />Готовить
<input name="Reg_hobby[]" type="checkbox" value="dance" />Танцевать
<input name="Reg_hobby[]" type="checkbox" value="read" checked />Читать<br />
</form>
</body>
</html>
```

На рис. 1.48 вы видите результат обработки браузером кода из листинга 1.57.

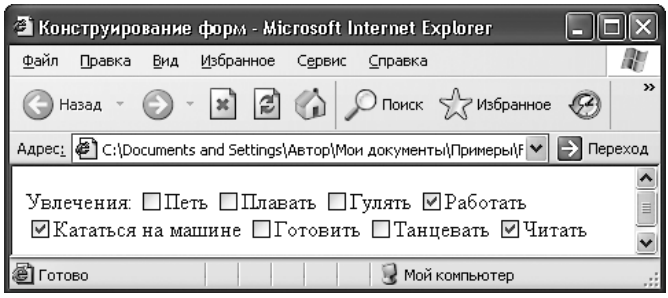


Рис. 1.48. Флажки

Поле для ввода файла

Для создания поля отправки файла установите значение атрибута `TYPE="file"`. Особенность этого поля состоит в том, что справа от него есть кнопка, при нажатии которой пользователь может выбрать загружаемый файл (через проводник).

В листинге 1.58 показывается пример создания поля для ввода файла. Атрибут `size` задает ширину поля равной 35 пикселов.

Листинг 1.58. Создание поля для загрузки файла

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
```

```
<form action="test.php" enctype="multipart/form-data" method="post" name="test"
target="_blank">
Добавьте свое фото: <input name="Reg_foto" type="file" size="55" /><br />
</form>
</body>
</html>
```

На рис. 1.49 вы видите результат обработки кода из листинга 1.58.

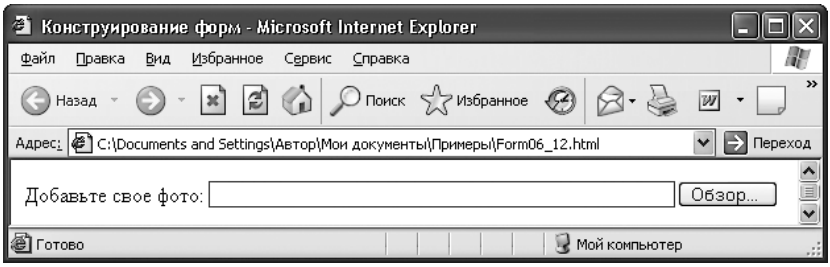


Рис. 1.49. Поле для ввода файла

Скрытые поля

Для передачи служебной информации используются скрытые поля. Такие поля не видны пользователю, однако информация из них передается обработчику. Эта информация не связана с данными, вводимыми пользователем, например с идентификатором посетителя в базе данных.

Для создания скрытого поля задайте значение атрибута `TYPE= "hidden"`. Для него нужны только атрибуты NAME и `VALUE` — имя и значение переменной, передаваемые обработчику.

В листинге 1.59 представлен пример создания скрытого поля, в котором передается идентификатор пользователя.

Листинг 1.59. Скрытое поле

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php" enctype="multipart/form-data" method="post" name="test"
target="_blank">
```

```
<input name="id_user" type="hidden" value="27" />
</form>
</body>
</html>
```

Списки

Списки дают возможность расположить большое количество пунктов компактно. При создании списков вы сами можете определить количество видимых элементов. Можно настроить возможность выбора одного или нескольких пунктов.

Для создания списков используется тег <SELECT>. Он имеет два атрибута, которые отвечают за общее отображение списка. Атрибут SIZE устанавливает количество видимых элементов списка. Если установлено значение SIZE="1" и количество элементов списка больше одного, то появится выпадающий список выбора. Если значение SIZE больше единицы, то при количестве пунктов списка, превышающем его размер, появится полоса прокрутки; если же пунктов меньше, то в поле останется пустое место. Атрибут MULTIPLE дает возможность выбирать несколько элементов списка, для этого при выборе нажмите клавишу Ctrl.

Для добавления в список пунктов используются теги <OPTION>, которые располагаются внутри блока тега <SELECT>. Атрибут VALUE этого тега задает значение, которое будет передаваться обработчику при выборе соответствующего пункта. Атрибут SELECTED задает пункт, выбранный по умолчанию. Название элемента списка находится внутри блока тега <OPTION>, закрывающий тег не обязателен.

В листинге 1.60 представлены примеры создания выпадающего списка и списка с полосой прокрутки (для этого списка установлена возможность множественного выбора).

Листинг 1.60. Создание списков

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php" enctype="multipart/form-data" method="post"
name="test">
Место проживания:
<select name="Reg_Place" size="1" >
<option value="SPb">Санкт-Петербург</option>
```

```
<option value="Mos" selected >Москва</option>
<option value="Ebr">Екатеринбург</option>
</select><br />
Области работы (можно выбрать несколько):<br />
<select name="Reg_Work[]" size="4" multiple="multiple" >
<option value="journ">Журналистика</option>
<option value="progr">Программирование</option>
<option value="site">Создание сайтов</option>
<option value="HR">Управление персоналом</option>
<option value="sale">Продажи</option>
<option value="admin">Администрирование</option>
<option value="trans">Перевод</option>
<option value="PR">Реклама</option>
</select><br />
</form>
</body>
</html>
```

На рис. 1.50 вы видите результат обработки кода из листинга 1.60.

Большие текстовые поля

В больших полях можно вводить комментарии, писать сообщения и многое другое. Для создания таких полей используется тег <TEXTAREA>, который обязательно должен иметь закрывающий тег. Текст, вводимый по умолчанию, находится внутри данного тега. Размеры поля устанавливаются с помощью атрибутов COLS и ROWS: первый задает размеры по горизонтали, второй — по вертикали.

Есть возможность запретить пользователю изменять текст в текстовом поле путем установки атрибута READONLY.

Атрибут WRAP указывает браузеру, как осуществляется перенос текста и в каком виде текст отправляется на сервер. Этот атрибут может принимать три значения.

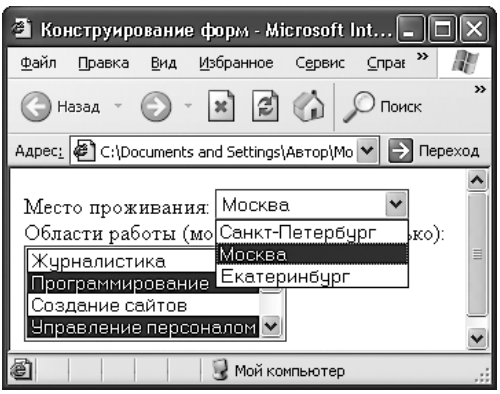


Рис. 1.50. Списки

Значения атрибута WRAP следующие:

- hard — автоматически переносит строку, когда она доходит до границы окна ввода, и передает на сервер с такой расстановкой переносов;
- soft — переносит строку автоматически, но на сервер знак переноса передает только в том месте, где была нажата клавиша Enter;
- off — переносит слова только при нажатии клавиши Enter; если строка превышает горизонтальный размер окна, появляется полоса прокрутки.

В листинге 1.61 представлен пример создания больших текстовых полей: первое — с горизонтальной полосой прокрутки, а второе — только для чтения.

Листинг 1.61. Создание больших текстовых полей

```
<html>
<head>
<title>Конструирование форм</title>
</head>
<body>
<form action="test.php" enctype="multipart/form-data" method="post" name="test"
target="_blank">
Расскажите немного о себе:<br />
<textarea cols="24" rows="10" name="about"
wrap="off" >Я самый лучший, лучше меня нет
никого.</textarea><br />
Соглашение:<br />
<textarea cols="24" rows="3" name="about"
readonly >Условия нашего соглашения</
textarea><br />
</form></body>
</html>
```

На рис. 1.51 вы видите результат обработки кода из листинга 1.61.

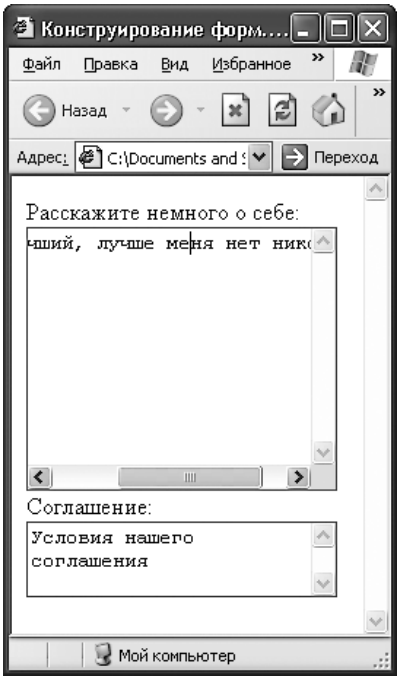


Рис. 1.51. Большие текстовые поля

Общие принципы CSS

В начале этой главы упоминалось, что существует три способа работы со стилями в HTML: можно просто указывать стили с помощью

атрибута `style` в тегах `p`, `div` и др., можно описывать стили в заголовках, а можно использовать внешний файл, описывающий все стили.

Последний вариант более предпочтителен, так как все страницы одного сайта желательно создавать в едином стиле. Если описание находится в отдельном файле, к которому обращаются остальные страницы, то это гарантирует единообразие дизайна. Любые изменения в файле, в котором хранится описание всех стилей, приводят к изменениям всего сайта.

В листинге 1.62 показывается, как можно подключить файл со стилями.

Листинг 1.62. Таблица стилей во внешнем файле

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
  <TITLE>Моя домашняя страница</TITLE>
  <LINK rel="stylesheet" href="my_style.css" type="text/css">
  </HEAD>
  <BODY>
    <H1>Моя домашняя страница</H1>
    <P>На этой странице вы найдете информацию обо мне и моих друзьях.
  </BODY>
</HTML>
```

Перед тем как непосредственно приступить к описанию CSS, рассмотрим комментарии. Эти элементы в CSS начинаются с символов `/*` и заканчиваются символами `*/`. Они могут находиться в любом месте таблицы. Использовать комментарии очень удобно: они помогут вам быстро разобраться, за какую часть документа отвечает тот или иной CSS-код.

Пример CSS-кода с комментарием следующий:

```
/* Цвет основного заголовка - синий */
H1 { color: blue }
```

Селекторы

Селектором в CSS будем считать названия тегов, для которых задаются свойства. Рассмотрим, как можно задать одни и те же свойства разным тегам документа.

Группировка

Для задания разным тегам документа одних и тех же свойств в CSS используется группировка. Селекторы, то есть теги, можно сгруппировать в список, разделив запятыми.

В следующем примере разным тегам задается одно свойство:

```
H1 { color: red }
H2 { color: red }
H3 { color: red }
```

Этот пример можно значительно упростить, используя правило группировки:

```
H1, H2, H3 { color: red }
```

Как отмечалось выше, при группировке селекторы разделяются запятыми.

Селекторы потомков

Иногда необходимо задать свойства тегу, который располагается внутри другого тега, то есть является так называемым потомком. Селектор потомков состоит из двух и более селекторов, разделенных пробелом.

Предположим, HTML-документ имеет следующий код:

```
<H1>Очень <EM>важный</EM> заголовок</H1>
```

Теперь допустим, что вы хотите изменить свойства тега <H1> и текста «важный». Если вы будете использовать CSS-код, приведеный ниже, то повлияете на все теги <EM> на странице (а этого вам не нужно):

```
H1 { color: red }
EM { color: blue }
```

Чтобы повлиять на свойства текста «важный», который находится внутри тега <EM>, а тот, в свою очередь, внутри тега <H1>, нужно использовать следующий CSS-код:

```
H1 EM { color: blue }
```

Обратите внимание, что, в отличие от группировки, при обращении к потомкам запятые не ставятся. Количество тегов, которые можно записать потомками, неограниченно, например:

```
BODY DIV P SPAN { color: green }
```

Если в вашем документе есть тег <SPAN>, который находится внутри тега <P>, который, в свою очередь, находится внутри тега <DIV>, то текст в данном теге будет зеленым. В столь длинных описаниях тег <BODY> можно опускать, так как все теги и так находятся внутри тега <BODY>.

Если при разработке таблицы стилей вы еще точно не знаете, какой тег будет размещаться внутри какого, то можете воспользоваться знаком звездочки. Учтите, что с обеих сторон звездочки обязательно должны находиться пробелы.

Пример, представленный выше, можно перезаписать так:

```
BODY * SPAN { color: green }
```

Однако помните, что текст будет зеленым, даже если тег <SPAN> просто находится внутри тега <DIV>. Другими словами, звездочкой считается любой набор селекторов и их потомков.

Сестринские селекторы

Сестринскими тегами называются теги, которые следуют друг за другом. Предположим, у вас есть абзац, который имеет атрибут `class` со значением `first`. Если за ним следует еще какой-то абзац, то размер вертикального пространства между ними можно уменьшить, используя такой код:

```
P.first + P {margin-top: -5mm }
```

Селекторы атрибутов

CSS позволяет создавать правила, сопоставляемые тегам с атрибутами, определенными в исходном коде HTML-документа. Селекторы атрибутов могут сопоставляться в четырех случаях:

- `[att]` — если для элемента установлен атрибут `att`, независимо от значения этого атрибута;
- `[att=val]` — если значение атрибута `att` данного элемента в точности равно `val`;
- `[att~=val]` — если значением атрибута `att` данного элемента является список слов, разделенных пробелами, одно из которых равно `val`;
- `[att|=val]` — если значением атрибута `att` элемента является список разделенных дефисом слов, начинающийся со слова `val`. Сопоставление всегда начинается с начала значения атрибута.

Чтобы разобраться с этим свойством, рассмотрим несколько примеров.

- Правило CSS сопоставляется всем элементам тега <P>, для которых описывается атрибут `align`, независимо от его значения:
  ```
  P[align] { color: blue;}
  ```
- Селектор, то есть правило CSS, сопоставляется всем элементам SPAN, у которых значение атрибута `class` равно `example`:
  ```
  SPAN[class=example] { color: blue; }
  ```
  Это значит, что данное правило не описывает сам класс и тег <P> с этим классом не будет отображать синий цвет.
- Для обращения к нескольким атрибутам элемента или для многократного обращения к одному и тому же атрибуту можно использовать несколько селекторов атрибутов. В этом примере селектор сопоставляется всем элементам TABLE, у которых значение атрибута `width` равно `90%`, а значение атрибута `height` — `50%`:
  ```
  TABLE[width="90%"][height="50%"] { color: blue; }
  ```
- Следующие селекторы иллюстрируют различия между «~=» и «=». Первый селектор сопоставляется значениям `right` или `left` атрибута `align`. Второй селектор будет сопоставляться только в том случае, если значение атрибута `href` равно `http://www.yoursite.com/`:
  ```
  P[align~="right left"]
  A[href="http://www.yoursite.com/"]
  ```
- Приведенное ниже правило скрывает все элементы, атрибут `lang` которых имеет значение `fr` , то есть элементы с текстом на французском языке:
  ```
  *[LANG=fr] { display : none }
  ```
- Следующее правило сопоставляется тем значениям атрибута `lang`, которые начинаются со значения `ru`, включая ru, `ru-RU` и `ru-UA`:
  ```
  *[LANG|="ru"] { color : red }
  ```

Селекторы классов

В таблицах стилей, используемых с HTML-документами, при сопоставлении значениям атрибута `class` вы можете использовать точку («.») как альтернативу условному обозначению «~=». Таким образом, выражения `DIV.value` и `DIV[class~=value]` имеют одинаковый смысл. Значение атрибута должно находиться непосредственно после точки.

Например, для всех элементов со значением `class=test` информацию о стиле можно определить следующим образом:

```
*.test { color: red }  /* все элементы с class=test */
```

или сокращенно:

```
.test { color: red }  /* все элементы с class=test */
```

Рассмотрим еще один пример. Следующее правило назначает стиль только элементу `H1` со значением `class=test`:

```
H1.test { color: green }  /* элемент H1 с class=test */
```

Благодаря этому правилу в следующем примере при первом появлении элемент `H1` не будет отображаться красным цветом, а при втором появлении будет:

```
<H1>Не зеленый цвет</H1>

<H1 class="test">Настоящий зеленый цвет</H1>
```

Для сопоставления перед каждым значением атрибута `class`, подмножество которых записано в произвольном порядке, следует поставить точку.

Например, следующее правило сопоставляется любому элементу `P`, атрибуту `class` которого в качестве значения назначен список разделенных пробелами значений, включающий слова test и marine:

```
P.test.marine { color: green }
```

Это правило сопоставляется, если, например, для тега `<P>` задается атрибут `class="Test blue aqua marine"`, и не сопоставляется, если задается атрибут `class="test blue"`

ID-селекторы

Атрибут `ID` позволяет назначить идентификатор одному экземпляру элемента в дереве HTML-документа. В CSS ID-селекторы сопоставляются экземпляру элемента в зависимости от его идентификатора. ID-селектор содержит символ «#», непосредственно за которым следует значение атрибута `ID`.

Например, следующий ID-селектор сопоставляется элементу `H1`, значение атрибута `ID` которого равно `chapter7`:

```
H1#chapter7 { text-align: center }
```

В примере из листинга 1.63 правило стиля сопоставляется элементу, у которого значение атрибута `ID` равно «first». Это значит, что данное правило будет сопоставляться элементу `P`.

Листинг 1.63. ID-селектор сопоставляет правило любому тегу

```
<HEAD>
  <TITLE>Глава 7</TITLE>
  <STYLE type="text/css">
    *#first { letter-spacing: 0.3em }
  </STYLE>
</HEAD>
<BODY>
   <P id=z98y>Текст абзаца</P>
</BODY>
```

Однако в примере из листинга 1.64 это правило стиля будет сопоставляться только элементу `H1`, у которого значение атрибута `ID` равно `first`. В этом примере данное правило не сопоставляется элементу `P`.

Листинг 1.64. ID-селектор сопоставляет правило только тегу H1

```
<HEAD>
  <TITLE>Глава 9</TITLE>
  <STYLE type="text/css">
    H1#first { letter-spacing: 0.5em }
  </STYLE>
</HEAD>
<BODY>
   <P id=first>Текст абзаца</P>
</BODY>
```

Отметим, что ID-селекторы имеют более высокий приоритет, чем селекторы атрибутов.

Псевдоклассы и псевдоэлементы

В CSS стиль обычно прикрепляется к элементу в зависимости от его местонахождения в дереве HTML-документа. В большинстве случаев эта простая модель оказывается приемлемой, но иногда из-за особенностей структуры дерева документа она не позволяет воплотить некоторые действия. Например, в HTML нет элемента, который указывает на первую строку абзаца, и поэтому нет простого селектора CSS, который смог бы на нее указывать.

В CSS вводится понятие о псевдоклассах и псевдоэлементах, позволяющих осуществлять форматирование на основании информации, не включенной в дерево HTML-документа. Сразу отметим, что при задании имен псевдоклассов и псевдоэлементов учитывается регистр.

Псевдокласс :first-child

Псевдокласс `:first-child` сопоставляется элементу, который является первым дочерним элементом некоторого другого элемента.

В следующем примере селектор сопоставляется каждому элементу P, который является первым дочерним элементом элемента `DIV`. Это правило запрещает отступ для первого абзаца элемента `DIV`:

```
DIV > P:first-child { text-indent: 0 }
```

Данный селектор сопоставляется элементу P внутри элемента `DIV` в следующем фрагменте:

```
<P>Последний P перед примечанием.
<DIV class="note">
   <P>Первый P внутри примечания.
</DIV>
```

Однако в следующем фрагменте он не сопоставляется элементу P, так как он второй, а первым дочерним элементом `DIV` является теперь тег `<H2>`:

```
<P>Последний P перед примечанием.
<DIV class="note">
   <H2>Примечание</H2>
   <P>Первый P внутри примечания.
</DIV>
```

В следующем примере устанавливается ширина шрифта `bold` для каждого элемента EM, являющегося одним из потомков первого дочернего элемента P:

```
P:first-child EM { font-weight : bold }
```

Обратите внимание, что безымянные блоки не являются частью дерева документа, поэтому они не учитываются при определении первого дочернего элемента. Например:

```
<P>Это <EM>жирный</EM> текст.</P>
<p>А <i>это <EM>нежирный</EM></i></p>
```

Псевдоклассы ссылок

Обычно браузеры по-разному отображают просмотренные и непросмотренные ссылки. В CSS предусматриваются псевдоклассы `:link` и `:visited` для различения этих ссылок и задания им своего стиля:

- псевдокласс `:link` применяется, если пользователь не просматривал ссылку;
- псевдокласс `:visited` применяется, если пользователь просматривал ссылку.

В HTML псевдоклассы ссылок применяются к тегам `<A>` с атрибутом `href`. Таким образом, следующие два объявления CSS эквивалентны:

```
A:link { color: red }
:link  { color: red }
```

Допустим, пользователь просмотрел такую ссылку:

```
<A class="external" href="http://yoursite.com/">перейти</A>
```

Тогда она будет представляться синим цветом согласно следующему правилу:

```
A.external:visited { color: blue }
```

Динамические псевдоклассы

Браузеры позволяют менять вид элементов HTML-документа после того, как пользователь выполнил некоторые действия. В CSS для наиболее часто встречающихся действий предусматривается три псевдокласса :

- `:hover` применяется, когда пользователь выделяет некоторый элемент (как правило, с помощью мыши), но не активизирует его;
- `:active` используется, когда пользователь активизирует элемент, например от момента, когда он нажал кнопку мыши, до момента, когда он ее отпустил;
- `:focus` применяется, когда элемент находится в «центре событий», например обрабатывает нажатия клавиш клавиатуры или другие типы ввода текста.

Несколько псевдоклассов могут одновременно сопоставляться элементу, например:

```
A:link    { color: red }    /* непросмотренные ссылки */
A:visited { color: blue }   /* просмотренные ссылки */
A:hover   { color: yellow } /* ссылка, над которой находится в данный момент
курсор */
A:active  { color: white }  /* активные ссылки */
```

Обратите внимание, что правило A:hover должно располагаться после правил A:link и A:visited. В противном случае правила каскадирования скроют свойство color правила A:hover. Аналогично, благодаря тому что правило A:active находится после правила A:hover, активная ссылка отображается белым цветом, когда пользователь устанавливает указатель поверх элемента A и одновременно активизирует его.

Псевдоэлемент :first-line

Псевдоэлемент :first-line задает стиль первой строки абзаца, например:

```
P:first-line { text-transform: uppercase }
```

Представленное выше правило делает буквы первой строки каждого абзаца заглавными. Селектор P:first-line не соответствует ни одному реальному элементу HTML. Он сопоставляется псевдоэлементу, который браузеры будут помещать в начало каждого абзаца.

Обратите внимание, что длина первой строки зависит от множества факторов: от ширины страницы, размера шрифта и т. д. Поэтому типичный абзац HTML-документа выглядит так:

```
<P>Этот длинный абзац HTML-документа будет разбит на несколько строк.</P>
```

Строки этого абзаца будут преобразованы следующим образом:

```
это длинный абзац HTML-документа,
будет разбит на несколько строк.
```

К псевдоэлементу :first-line применяются свойства шрифтов, цвета, фона, word-spacing, letter-spacing, text-decoration, vertical-align, text-transform, line-height, text-shadow и clear.

Псевдоэлемент :first-letter

Псевдоэлемент :first-letter может использоваться для создания такого простого типографического эффекта, как буквица. Он соответствует строчному элементу, если его свойство float равно none, иначе он соответствует плавающему элементу.

К псевдоэлементам :first-letter применяются свойства шрифтов, цвета, фона, границ, отступов, рамок: text-decoration, vertical-align (только если для свойства float устанавливается значение none), text-transform, line-height, float, text-shadow и clear.

В следующем примере из листинга 1.65 высота буквицы устанавливается равной высоте двух строк.

Листинг 1.65. Стиль заглавной буквы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
 <HEAD>
  <TITLE>Глава 7. Стиль заглавной буквы</TITLE>
  <STYLE type="text/css">
   P              { font-size: 12pt; line-height: 12pt }
   P:first-letter { font-size: 200%; font-style: italic;
                    font-weight: bold; float: left }
   SPAN           { text-transform: uppercase }
  </STYLE>
 </HEAD>
 <BODY>
  <P><SPAN>Первые</SPAN> несколько слов из книги
    "Самоучитель по HTML и CSS".</P>
 </BODY>
</HTML>
```

Текст из примера в листинге 1.65 может быть отформатирован так, как показано на рис. 1.52.

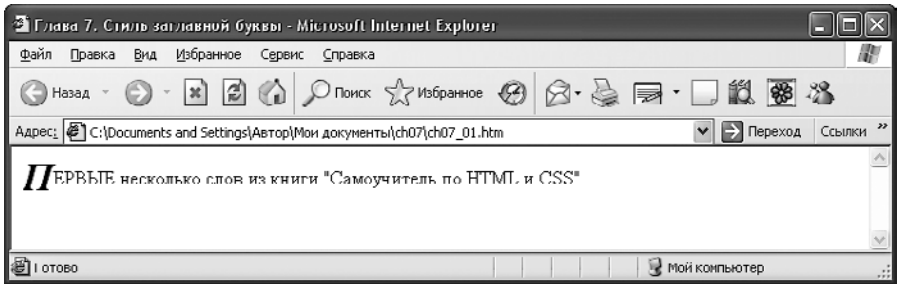


Рис. 1.52. Стиль заглавной буквы

Рассмотрим, в какой последовательности должны располагаться функциональные теги.

В следующем примере используются налагающиеся псевдоэлементы. Первая буква каждого элемента P выделяется зеленым цветом, а размер шрифта устанавливается

в 24 пт. Остальные буквы первой форматируемой строки представляются синим цветом, а все оставшиеся буквы абзаца — красным. Приведем CSS-код примера:

```
P { color: red; font-size: 12pt }
P:first-letter { color: green; font-size: 200% }
P:first-line { color: blue }
```

Псевдоэлементы :before и :after

Псевдоэлементы `:before` и `:after` можно использовать для вставки содержимого до или после содержимого элемента.

Когда псевдоэлементы `:first-letter` и `:first-line` сочетаются с псевдоэлементами `:before` и `:after`, то они применяются к первой букве или строке элемента, включая вставляемый текст:

```
P.special:before {content: "Важно! "}
P.special:first-letter {color: gold}
```

Слово «Важно!» добавится в начало абзаца стиля `special`, а буква `B` этого слова будет золотистого цвета.

Форматирование текста средствами CSS

Теперь рассмотрим основные приемы работы с текстом — форматирование и установку стиля, выбор шрифта и задание цвета.

Сразу оговоримся, что при работе со шрифтами следует учитывать, что выбранный вами шрифт должен быть установлен на компьютере клиента. В противном случае пользователь увидит страницу, шрифт текста которой задается по умолчанию настройками его браузера.

Выделение текста цветом

Все свойства CSS по работе с цветом наследуемы. Если вы, к примеру, зададите красный цвет текста для тега `<BODY>`, то весь текст на странице станет красным. Если после этого вы зададите синий цвет для таблиц с именем класса `blue_text`, то текст во всех таблицах с классом `blue_text` будет синим. Если же вы зададите красный цвет для тегов `<P>`, а синий — для `<BODY>`, то цвет текста в таблицах останется синим. Однако если внутри какой-то ячейки вы применяете тег `<P>`, то цвет текста в этой ячейке будет красным.

Цвет текста

Для изменения цвета текста используется свойство `color`, в качестве параметра которого задается название или код цвета:

```
H1 {
    color: red;
}
```

Значение цвета можно задать тремя способами:

- английским названием цвета (как в приведенном выше примере);
- шестнадцатеричным значением (например, `#ff0000`);
- rgb-значением (например, `rgb(255, 0, 0)`).

Свойство `color` используется для задания цвета ссылкам в документе, например:

```
A:hover {
    color: grey;
}
```

Цвет фона

Цвет фона задается с помощью свойства `background-color`.

Для примера выделим заголовки второго уровня (то есть теги `<H2>`) на HTML-странице синим цветом. Для этого добавим в файл таблицы стилей такой код:

```
H1 {
    background-color: blue;
}
```

Теги `<H1>`, `<P>` и др. выделяют цветом всю область. Например, тег `<H1>` выделяет строку целиком, а тег `<P>` — только прямоугольную область, даже если последняя строка обрывается на середине. Чтобы установить фон для области, занятой только текстом, необходимо внутрь тега `<H1>` вложить тег `<SPAN>` и задать цвет лишь для фона:

```
H1 SPAN {
    background-color: blue;
}
```

Теперь, когда вы научились работать с цветом, перейдем к важному подразделу, из которого вы научитесь устанавливать для своего текста вид и размер.

Шрифты

Шрифт — это «лицо» текста, поэтому для начала разберемся, как задавать тексту нужный шрифт. Далее подробно рассмотрим все возможные варианты работы со шрифтом.

Семейство шрифта

Свойство `font-family` указывает браузеру шрифт или список шрифтов, которым должен отображаться текст. Не забывайте, что заданный шрифт может быть не установлен на машине клиента. В этом случае браузер отобразит текст на экране шрифтом, который будет найден первым среди установленных шрифтов на компьютере пользователя при просмотре списка слева направо. Если ни один из перечисленных шрифтов не обнаружится, то браузер клиента отобразит текст шрифтом по умолчанию.

Вы можете также указать семейство шрифта. *Семейство* — это группа шрифтов, созданных по сходным принципам дизайна и внешнего вида. В CSS есть пять основных групп: serif, sans-serif, monospace, fantasy и cursive.



ПРИМЕЧАНИЕ

После задания шрифтов последним в списке лучше указывать название семейства, к которому относится шрифт. Тогда, если не найдется ни один из шрифтов, браузер отобразит текст шрифтом из заданного семейства.

Для примера зададим шрифт текста абзацев (Trebuchet MS) и семейство, к которому относится данный шрифт (`sans-serif`):

```
P {
    font-family: "Trebuchet MS", sans-serif;
}
```



ВНИМАНИЕ

Помните, что если название шрифта содержит пробелы, то его необходимо заключить в кавычки.

Вы можете задавать другие шрифты и группы. Ниже приведены несколько групп и их основные шрифты, доступные большинству пользователей с разными браузерами и операционными системами:

- serif — Times New Roman, Garamond, Georgia;
- sans-serif — Trebuchet, Arial, Verdana;
- monospace — Courier, Courier New, Andele Mono.

Старайтесь не использовать экзотические шрифты, и тогда вы сможете быть уверены, что страница у клиента отобразится так, как вы задумали.

Стиль шрифта

Стиль шрифта задается свойством `font-style`, которое может принимать одно из трех значений: `normal`, `oblique` или `italic`. Эти значения имеют следующий смысл:

- `normal` — при отображении на экране выглядит как обычный текст;
- `oblique` — такой текст в действительности может генерироваться автоматически путем наклона нормального шрифта;
- `italic` — определяет курсивный шрифт или, если наклонного шрифта нет, шрифт `oblique`.

Стиль `oblique` используется в тех случаях, когда курсивный шрифт не установлен. Он также наклонный, однако выглядит неэстетично.

Вид шрифта

Свойство `font-variant` может принимать одно из двух значений: `normal` или `small-caps`. Если задается значение `small-caps`, то текст будет выводиться шрифтом малых заглавных букв. Если задается значение `normal`, то текст отображается обычным образом.

ПРИМЕЧАНИЕ



Следует отметить, что если свойство font-variant имеет значение small-caps, а шрифт недоступен, то браузер будет симулировать такой шрифт, например, замещением букв нижнего регистра обычного шрифта пересчитанными символами верхнего регистра. В крайнем случае, весь текст будет воспроизводиться буквами верхнего регистра.

Ширина шрифта

Стилевое свойство `font-weight` определяет ширину шрифта, то есть насколько текст «толстый», или «жирный». Значение задается числом от 100 до 900 (с шагом 100), где каждый номер обозначает шрифт, который темнее своего предшественника, либо принимает значения, приведенные ниже.

- `normal` — текст отображается обычным шрифтом. В числовом выражении соответствует значению `400`.
- `bold` — текст отображается более широким шрифтом, чем обычно. Численно оно равно значению `700`. Такая ширина называется «полужирным начертанием».

- `bolder` — определяет вес шрифта, который темнее, чем наследуемый. Если наследуемое значение равно значению `900`, то результат будет также равен значению `900`.
- `lighter` — определяет вес шрифта, который светлее, чем наследуемый. Если наследуемое значение равно значению `100`, то результат будет также равен значению `100`.

Размер шрифта

Для указания размера используется свойство `font-size`. Размер шрифта задается с помощью значения и единицы измерения, выбранной в одной из двух групп: абсолютной и относительной.

Абсолютные единицы:

- `in` — дюйм ~ 2,5 см;
- `mm` — миллиметр;
- `cm` — сантиметр;
- `pt` — пункт ~ 1/7 дюйма;
- `pc` — пика (равна 12 пт).

Относительные единицы:

- `em` — высота шрифта элемента;
- `ex` — высота буквы *x*;
- `px` — пиксел;
- `%` — процентное соотношение.

Группу относительных единиц удобно использовать, дабы сохранить первозданный вид документа на любом устройстве (экране и при печати).

Обычно размер шрифта текста задается равным 14 пт:

```
P {
    font-size: 14pt;
}
```

Мы рассмотрели основные возможности CSS по работе со шрифтами. Теперь приступим к изучению форматирования текста.

Форматирование текста

В этом подразделе вы познакомитесь с впечатляющими средствами CSS для отображения текста.

Отступы

HTML не предоставляет специальных средств для задания отступа абзаца. Их приходится делать с помощью пробелов, и это неудобно. Для решения этой проблемы CSS предлагает вам замечательное свойство — `text-indent`. Размер отступа может задаваться как абсолютными, так и относительными единицами, в том числе и процентами. В качестве 100 % считается ширина страницы.

```
P {
    text-indent: 25px;
}
```

Чтобы лучше разобраться в этом свойстве, вы можете поэкспериментировать с разными значениями и единицами измерений.

Выравнивание текста

Выравнивание текста задается с помощью свойства `text-align`, значения которого такие же, как и значения свойства `align` у тега `<P>`:

- `left` — выравнивание текста влево;
- `right` — выравнивание текста вправо;
- `center` — выравнивание текста по центру;
- `justify` — выравнивание текста по формату.

Оформление

С помощью CSS можно добавлять или изменять декоративное оформление текста. Для этого используется свойство `text-decoration`.

Ниже приводятся все возможные значения данного свойства и их описания:

- `none` — не выполняет декоративное оформление;
- `underline` — каждая строка текста подчеркивается;
- `overline` — над каждой строкой текста отображается линия;
- `line-through` — каждая строка текста отображается зачеркнутой;
- `blink` — текст мигает.

ПРИМЕЧАНИЕ



Вы можете использовать одно значение или сразу несколько. Если вы применяете несколько значений, то между ними нужно ставить только пробел. Использование запятой недопустимо, поскольку браузер использует только те эффекты, которые записываются после последней запятой.

Свойство `none` на первый взгляд может показаться лишним: ведь если не задавать это свойство, то текст и так будет без декоративного оформления. Но его полезно использовать для ссылок. По умолчанию для ссылок браузер считает заданным значение этого свойства `underline`, то есть все ссылки на странице отображаются подчеркнутыми. Если вы захотите сделать ссылки такими же, как и основной текст (без подчеркивания), то в этом случае вам пригодится значение `none`. Нередко это значение используется для обычных ссылок, а `underline` — для тех, на которые наведен указатель мыши:

```
A {
    text-decoration: none;
}
A:hover {
    text-decoration: underline;
}
```

Расстояние между буквами

Расстояние между буквами можно задать свойством `letter-spacing`. В качестве значения указываются необходимая величина и абсолютная единица измерения.

Для примера зададим расстояние между буквами в заголовке равным 6 пикселов:

```
H1 {
    letter-spacing: 6px;
}
```

Расстояние между словами

Вы можете задать также расстояние между словами, используя свойство `word-spacing`. В качестве значения можно указать желаемое значение либо значение `normal`, чтобы использовать значение браузера по умолчанию.

```
H1 { word-spacing: 1em }
```

В данном примере расстояние между всеми словами в тегах `<H1>` увеличивается на `1em`.

Трансформация текста

Свойство `text-transform` управляет регистром символов. Для данного свойства возможно три значения:

- `capitalize` — пишет все слова с большой буквы;
- `uppercase` — меняет все символы в верхний регистр;
- `lowercase` — меняет все символы в нижний регистр.

Пустое место

CSS также позволяет задавать способ обработки браузером пустого пространства. Для этого используется свойство `white-space`. Значения, которые может принимать это свойство, и их описания приводятся ниже.

- `normal`. Браузер будет сжимать последовательное пустое пространство и разбивать строки для вмещения в них линейных блоков.
- `pre`. Браузер не может изменять последовательность символов пустого пространства. Строки разбиваются только в местах начала новых строк, обозначенных в исходном тексте.
- `nowrap`. При сокращении пустого пространства это значение действует так же, как значение `normal`. Однако оно запрещает разрывать строки в тексте, за исключением разрывов, определенных с помощью тега `<BR>`.

Рассмотрим следующий пример:

```
BODY { white-space: pre }
```

Браузер покажет HTML-документ с такой же таблицей стилей, с какой он отображается в исходном виде, например в программе Блокнот.

Оформление документа CSS

Теперь рассмотрим возможности CSS по оформлению документа. Изучив этот материал, вы с легкостью сможете придать своим страницам эксклюзивный вид.

Фон

Выше мы уже говорили, как задать фон для текста (с помощью свойства `background-color`). Точно так же задается цвет для любых тегов — таблиц, слоев и т. д.

CSS также позволяет задать для тегов фоновое изображение.

Фон может принимать следующие свойства: `background-color`, `background-image`, `background-repeat`, `background-attachment`, `background-position` и `background`.

Свойство `background-image` задает графический объект, то есть картинку из файла, в качестве фона элемента. При указании фонового изображения рекомендуется задавать фон, который будет использоваться в случае, если изображение окажется недоступным. Если изображение доступно, то оно отображается поверх фонового цвета, который все же будет виден сквозь прозрачные фрагменты изображения (если такие есть). В качестве значения этого свойства задается адрес картинки. Рассмотрим такой фрагмент таблицы стилей.

```
BODY { background-image: url("some_bgimage.gif") }
P { background-image: none }
```

В данном примере в качестве фона страницы используется изображение из файла `some_bgimage.gif`. Содержимое страницы внутри тега `<P>` будет находиться на белом фоне либо на фоне, который задается в браузере по умолчанию.

Если вы задали фоновое изображение, то можете также использовать свойство `background-repeat`, которое определяет, будет ли изображение дублироваться, и если да, то каким образом. Данное свойство может принимать следующие значения:

- `repeat` — изображение дублируется как по вертикали, так и по горизонтали;
- `repeat-x` — изображение дублируется только по горизонтали;
- `repeat-y` — изображение дублируется только по вертикали;
- `no-repeat` — изображение не дублируется: отображается только одна копия изображения.

Рассмотрим такой пример:

```
BODY {
  background: white url("pendant.gif");
  background-repeat: repeat-y;
  background-position: center;
}
```

В данном примере страница имеет белый цвет. Фоновое изображение будет вертикально продублировано и расположено посередине страницы.

Если вы задали фоновое изображение, то можете задать свойство `background-attachment`, которое определяет, будет ли изображение фиксироваться относительно окна просмотра (значение `fixed`) или перемещаться вместе с документом (значение `scroll`) при его прокрутке.

Вы также можете задать для фонового изображения начальное положение на странице, используя свойство `background-position`. Местоположение можно задавать, используя проценты от размера окна браузера.

Это значение можно задать, указав число и единицу длины. Например, если вы зададите пару «2cm 2cm», то верхний левый угол изображения сместится на 2 см правее и 2 см ниже верхнего левого угла области, предназначенной для отступов.

В качестве значений вы можете также использовать ключевые слова. Ниже приводится их список и указывается, какой паре значений они аналогичны.

- `top left` и `left top` — «0 % 0 %».
- `top`, `top center` и `center top` — «50 % 0 %».
- `right top` и `top right` — «100 % 0 %».
- `left`, `left center` и `center left` — «0 % 50 %».
- `center` и `center center` — «50 % 50 %».
- `right`, `right center` и `center right` — «100 % 50 %».
- `bottom left` и `left bottom` — «0 % 100 %».
- `bottom`, `bottom center` и `center bottom` — «50 % 100 %».
- `bottom right` и `right bottom` — «100 % 100 %».

Для этих свойств можно использовать сокращенную запись, используя свойство `background`. Например:

```
BODY { background: url("chess.png") gray 50% 20% repeat fixed }
```

В данной таблице стилей задаются значения для всех свойств фона документа.

Поля и отступы

Разберем важную составляющую создания веб-страниц — задание полей и отступов.

Свойства полей

Браузер, отображая каждый элемент, как бы создает прямоугольный блок, внутри которого размещает содержимое тега. Свойства полей определяют, какую ширину

будет иметь область поля блока. Свойство `margin` позволяет управлять шириной всех четырех полей. Вы можете использовать одно из следующих свойств: `margin-top`, `margin-right`, `margin-bottom` и `margin-left` либо несколько из них в любой комбинации, чтобы задать значения полей только для отдельных сегментов.

Все свойства `margin` могут принимать значения, которые приводятся ниже.

- Длина. Вы указываете фиксированную ширину поля.
- Проценты. Вы указываете значения поля в процентах. Процентное соотношение вычисляется относительно ширины сгенерированного блока.
- `Auto`. Браузер задает значение поля автоматически. Каждый браузер использует собственные значения.

В качестве примера рассмотрим небольшое правило CSS, которое устанавливает значение ширины верхнего поля, равное `2em`:

```
H1 { margin-top: 2em }
```

Сокращенная запись указанных выше свойств выглядит следующим образом: `margin`. Если данное свойство принимает только одно значение, то это значение применяется ко всем полям. Если оно принимает два значения, то первое применяется к верхнему и нижнему полям, а второе — к правому и левому полям. Если же оно принимает три значения, то первое применяется к верхнему, второе — к левому и правому, а третье — к нижнему полям. И, наконец, если оно принимает четыре значения, то они применяются, соответственно, к верхнему, правому, нижнему и левому сегментам. Например:

```
BODY { margin: 1em 2em 3em }
```

Если применить эту таблицу стилей к HTML-документу, ширина верхнего, правого, нижнего и левого полей страницы будет равна, соответственно, `1em`, `2em`, `3em` и `2em`.

Свойства отступов

Свойства отступов позволяют задать ширину области отступов блока.

Существует четыре свойства отступов: `padding-top`, `padding-right`, `padding-bottom` и `padding-left`, которые задают, соответственно, верхний, правый, нижний и левый отступы. Свойство `padding` является сокращенной записью всех этих свойств отступов. Вы можете также комбинировать эти свойства.

Как и для свойств поля, процентные соотношения, задаваемые для свойств отступов, вычисляются относительно ширины сгенерированного блока. Однако, в отли-

чие от свойств поля, значения свойств отступов не могут принимать отрицательные значения, и для них не может задаваться значение `auto`.

Фон отступов будет соответствовать цвету или изображению, заданным этому тегу с помощью свойства `background`:

```
H1 {
  background: white;
  padding: 1em 2em;
}
```

Границы

Для всех элементов страницы вы можете задать границы, их ширину, цвет и стиль.

Ширина границы

За ширину границы отвечают свойства `border-top-width`, `border-right-width`, `border-bottom-width` и `border-left-width`, которые определяют, соответственно, ширину верхней, правой, нижней и левой границ элемента страницы.

Свойство `border-width` позволяет установить значения одновременно для всех четырех свойств. Причем если для `border-width` задать только одно значение, то оно будет применяться ко всем границам. Если задать два значения, то первое будет применяться к верхней и нижней, а второе — к правой и левой границам и т. д.

Рассматриваемые здесь свойства могут принимать одно из следующих значений:

- `thin` — тонкая граница;
- `medium` — граница средней толщины;
- `thick` — широкая граница;
- длина — толщина границы задается числом, которое, естественно, не может быть отрицательным.

Каждый браузер может немного по-своему устанавливать заданные числовые значения `thin`, `mediun` и `thick`.

Теперь давайте рассмотрим, как можно задать границе цвет.

Цвет границы

Аналогично заданию ширины границы цвет можно указать с помощью четырех свойств: `border-top-color`, `border-right-color`, `border-bottom-color`

и border-left-color (сокращенная запись — свойство border-color). Они задают цвет верхней, правой, левой и нижней границ соответственно. Для всех этих свойств вы можете либо задать значения цвета, либо использовать ключевое слово transparent. Если задать значение transparent, то граница будет невидимой, то есть прозрачной, даже если она имеет некоторую ширину.

Приведем такой пример:

```
P {
  border-width: 2px
  border-color: blue;
}
```

Для элемента P будет создана граница синего цвета размером 2 пиксела.

Стиль границы

Стиль границы задается свойствами border-top-style, border-right-style, border-bottom-style и border-left-style (сокращенная запись — свойство border-style). Свойства стиля границы определяют стиль линии (сплошная, двойная, пунктирная и т. д.), выступающей в качестве границы. Все свойства могут принимать одно из следующих значений:

- none — граница отсутствует (значение свойства border-width равно нулю);
- hidden — аналогично значению none, за исключением сценария разрешения конфликтов между границами, который используется для элементов таблицы;
- dotted — граница отображается пунктирной линией;
- dashed — граница отображается штрихпунктирной линией;
- solid — граница отображается сплошной линией;
- double — граница отображается двумя непрерывными линиями; сумма значений ширины двух линий и расстояния между ними равна значению свойства border-width;
- groove — граница отображается вдавленной линией;
- ridge — граница отображается выпуклой линией (противоположно значению groove);
- inset — весь блок выглядит вдавленным;
- outset — весь блок выглядит выпуклым (противоположно значению inset).

Границы выводятся поверх фоновой заставки блока. Цвет границ для значений groove, ridge, inset и outset зависит от того, какое значение имеет свойство элемента color.

Свойство `border-style` определяет стиль всех четырех сегментов границ. Оно может иметь от одного до четырех значений, каждое из которых присваивается различным сегментам подобно тому, как это происходит у свойства `border-width`.

Рассмотрим такой пример:

```
#xyz { border-style: solid dotted }
```

В приведенном примере горизонтальные сегменты границы будут иметь значение `solid`, а вертикальные — `dotted`.

Начальное значение стиля границы равно `none`, поэтому она не будет видна до тех пор, пока для нее не будет указан другой стиль.

Работа с блоками

В этом подразделе мы рассмотрим работу с блоками, или слоями (тег `<DIV>`). Вы научитесь указывать точное место на странице, где будут размещаться элементы страницы.

Важно подчеркнуть значение слоев для дизайна. Существует два подхода к воплощению графического дизайна в коде HTML-страницы. В первом случае создается множество различных вложенных таблиц, в ячейках которых находятся элементы дизайна. Во втором случае все элементы расставляются по странице с помощью слоев. На практике эти два подхода удобно сочетать.

Выбор схемы позиционирования

Свойства `position` и `float` задают алгоритм позиционирования, который позволяет определить положение блока.

Приведем значения этого свойства и интерпретацию их браузером.

- `static`. Этот блок является обычным блоком, позиционируемым в соответствии с нормальным потоком. Свойства `left` и `top` не применяются.
- `relative`. Положение блока рассчитывается в соответствии с нормальным потоком (оно называется нормальным положением). Затем блок смещается относительно своего нормального положения. Если для нахождения расположения блока используется модель относительного позиционирования, то положение следующего блока рассчитывается так, как будто блок не смещался.
- `absolute`. Положение блока (можно также и размер) указывается с помощью свойств `left`, `right`, `top` и `bottom`. Они указывают величину смещения

относительно контейнера блока. Абсолютно позиционируемые блоки изымаются из нормального потока. Это значит, что они не влияют на размещение последующих сестринских элементов. Стоит отметить, что поля абсолютно позиционируемых блоков не перекрываются никакими другими полями.

- `fixed`. Положение блока рассчитывается в соответствии с моделью абсолютного позиционирования, а затем он фиксируется относительно некоторого объекта. При использовании устройств без разбивки блок фиксируется относительно области просмотра (то есть не перемещается при прокрутке). Если же применяются устройства с постраничной разбивкой, то блок фиксируется относительно страницы, даже если она отображается в области просмотра (например, при предварительном просмотре перед печатью). Вы можете сделать так, что значение `fixed` станет аппаратно-зависимым.

Смещение блоков

Говорят, что позиция была присвоена элементу, если для свойства `position` устанавливается значение, отличное от `static`. Позиционированные элементы порождают позиционированные блоки, положение которых регулируется с помощью следующих четырех свойств:

- `top` — задает смещение верхнего сегмента внутренней краевой линии относительно верхнего сегмента краевой линии контейнера блока;
- `right` — задает смещение правого сегмента внутренней краевой линии влево относительно правого сегмента краевой линии контейнера блока;
- `bottom` — определяет смещение нижнего сегмента внутренней краевой линии вверх относительно нижнего сегмента краевой линии контейнера блока;
- `left` — определяет смещение левого сегмента внутренней краевой линии вправо относительно левого сегмента краевой линии контейнера блока.

Рассмотрим значения этих свойств.

- Длина — смещение задается фиксированным значением расстояния от исходной краевой линии.
- Проценты — смещение задается процентным соотношением относительно ширины (для свойств `left` и `right`) или высоты (для свойств `top` и `bottom`) контейнера. Если высота контейнера точно не указывается, то есть зависит от высоты содержимого, то процентное соотношение, определяющее значение свойств `top` и `bottom`, интерпретируется как значение `auto`.
- `auto` — результат этого значения зависит от того, какие свойства, близкие по своему содержанию к рассматриваемым, также имеют значение `auto`.

Абсолютно позиционируемые блоки смещаются относительно контейнера, в то время как относительно позиционируемые блоки — относительно внешней краевой линии самого блока. Сначала блоку присваивается положение в нормальном потоке, а затем, с использованием названных свойств, указывается смещение относительно этого положения.

Позиционирование перемещаемого объекта

Перемещаемый объект — это блок, который смещается по строке в левую или правую сторону. Свойство `float` определяет, будет ли перемещаться блок, и если да, то в какую сторону — влево или вправо. Оно может устанавливаться для элементов, порождающих блоки, которые не являются абсолютно позиционируемыми. Ниже приводятся значения этого свойства.

- `left` — элемент порождает структурный блок, перемещаемый влево. Содержимое выводится вдоль правой стороны блока, начиная с самого верха (за счет свойства `clear`). Свойство `display` игнорируется, если ему не присваивается значение `none`.
- `right` — аналогично значению `left` с той лишь разницей, что содержимое выводится вдоль левой стороны блока, начиная с самого верха.
- `none` — блок не перемещается.

Следующее правило перемещает все блоки, порожденные элементом `IMG` со значением `class="icon"`, влево и устанавливает значение ширины поля равным нулю:

```
IMG.icon {
  float: left;
  margin-left: 0;
}
```

Рассмотрим исходный HTML-документ и таблицу стилей из листинга 1.66.

Листинг 1.66. Пример перемещаемого объекта

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
    <TITLE> Пример перемещаемого объекта</TITLE>
    <STYLE type="text/css">
      IMG { float: left }
      BODY, P, IMG { margin: 2em }
    </STYLE>
```

```
  </HEAD>
  <BODY>
     <P><IMG src="img.gif" alt="Данный рисунок иллюстрирует перемещаемые
объекты">
       Пример текста, у которого нет другого...
  </BODY>
</HTML>
```

Блок, порожденный элементом `IMG`, перемещается влево. Следующее за ним содержимое располагается справа от него, начиная с той же строки, на которой находится он сам. Присутствие перемещаемого объекта влияет на то, что пока линейные блоки находятся справа от него, то они укорачиваются. Однако как только они достигают его конца, то сразу восстанавливают свою исходную ширину (равную ширине поля (контейнера), назначенной элементом `P`). Форматирование было бы точно таким же, как если бы HTML-документ выглядел следующим образом:

```
<BODY>
  <P>Произвольный текст,
  <IMG src="img.gif" alt="Данный рисунок иллюстрирует перемещаемые объекты">
            у которого нет другого...
</BODY>
```

Объясняется это тем, что содержимое, которое находится слева от перемещаемого объекта, заменяется им, а само отображается справа от него.

Фиксированное позиционирование

Фиксированное позиционирование является разновидностью абсолютного позиционирования. Единственной его особенностью является то, что контейнер при этом позиционировании определяется областью просмотра, а не общей областью страницы. В этом отношении такие контейнеры схожи с фиксированными фоновыми изображениями. Это может оказаться удобным, например, при размещении подписей в нижней части каждой страницы, однако на практике используется нечасто.

Вы можете использовать фиксированное позиционирование для создания презентации в виде совокупности кадров. Рассмотрим один из примеров такой презентации, представленный на рис. 1.53.

Подобный эффект можно достичь с помощью кода HTML и правил стиля, которые содержатся в листинге 1.67.

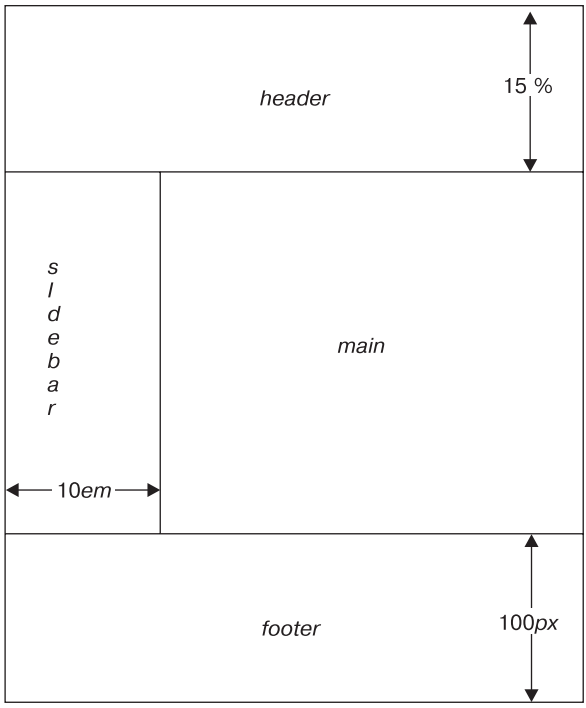


Рис. 1.53. Фиксированное позиционирование

Листинг 1.67. Фиксированное позиционирование

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
    <TITLE> Создание кадров в CSS</TITLE>
    <STYLE type="text/css">
      BODY { height: 8.5in } /* будет использовано ниже для определения */
                             /* процентных соотношений, задающих значение */
                             /* высоты элемента */
      #header {
        position: fixed;
        width: 100%;
        height: 15%;
        top: 0;
        right: 0;
        bottom: auto;
```

```
      left: 0;
    }
    #sidebar {
      position: fixed;
      width: 10em;
      height: auto;
      top: 15%;
      right: auto;
      bottom: 100px;
      left: 0;
    }
    #main {
      position: fixed;
      width: auto;
      height: auto;
      top: 15%;
      right: 0;
      bottom: 100px;
      left: 10em;
    }
    #footer {
      position: fixed;
      width: 100%;
      height: 100px;
      top: auto;
      right: 0;
      bottom: 0;
      left: 0;
    }
  </STYLE>
</HEAD>
<BODY>
  <DIV id="header"> ...  </DIV>
  <DIV id="sidebar"> ...  </DIV>
  <DIV id="main"> ...  </DIV>
```

```
    <DIV id="footer"> ...  </DIV>
  </BODY>
</HTML>
```

Нарисуйте на бумаге разметку своей страницы, а затем попробуйте создать ее, используя приобретенные навыки.

Определение позиционного уровня

Для размещенного блока вы можете задать позиционный уровень, используя свойство `z-index` в текущем контексте. Это свойство может принимать такие значения:

- `auto` — позиционный уровень блока в текущем контексте совпадает с позиционным уровнем родительского блока;
- целое число — данное значение является значением позиционного уровня блока, сгенерированного в текущем контексте. Блок также назначает локальный позиционный контекст с позиционным уровнем, равным нулю.

Рассмотрим пример, приведенный в листинге 1.68. Позиционные уровни блоков здесь именуются с использованием атрибута `id`, причем позиционный уровень `text2` наследуется от корневого блока. Другие уровни указываются свойством `z-index`.

Листинг 1.68. Позиционирование вдоль оси OZ

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
    <TITLE>Позиционирование вдоль оси z</TITLE>
    <STYLE type="text/css">
      .pile {
        position: absolute;
        left: 2in;
        top: 2in;
        width: 3in;
        height: 3in;
      }
    </STYLE>
  </HEAD>
  <BODY>
```

```
    <P>
      <IMG id="image" class="pile"
           src="someimg.gif" alt="Картинка"
           style="z-index: 1">
    <DIV id="text1" class="pile"
         style="z-index: 3">
      Этот текст будет находиться поверх изображения.
    </DIV>
    <DIV id="text2">
      Этот текст будет находиться под всем остальным.
    </DIV>
    <DIV id="text3" class="pile"
         style="z-index: 2">
      Этот текст будет находиться ниже текста text1, но поверх изображения.
    </DIV>
  </BODY>
</HTML>
```

Этот пример поясняет понятие прозрачности. По умолчанию находящиеся за блоком другие блоки видны сквозь прозрачную область его содержимого. В данном примере каждый блок, накладывающийся на другие блоки, является прозрачным. Такой тип поведения может быть переназначен с помощью одного из существующих свойств фона.

Резюме

Эта глава посвящена такому важному вопросу, как верстка веб-страниц. Мы разобрали основные возможности HTML и CSS по созданию и оформлению веб-документов, ключевые моменты построения HTML-документа и основные приемы, используемые при верстке. Знание CSS позволяет в полной мере воспользоваться всеми возможностями, предоставляемыми HTML-верстальщику для воплощения в жизнь дизайнерских задумок.

Немного практики — и вы сможете создать и оформить страницу как истинный профессионал. Хотя, конечно, осветить подробно такие обширные темы в одной главе невозможно. Но, надеюсь, на первых порах этого будет достаточно.

Глава 2

Работа с JavaScript

Область применения JavaScript

Создание сценариев

Синтаксис и основные средства языка JavaScript

Объектная модель

Работа с формами

Динамический HTML

Резюме

Эта глава посвящается языку программирования сценариев JavaScript. Вы узнаете, что же такое язык JavaScript и для чего он необходим.

Рассмотрим особенности создания динамики на странице, обработки форм, а также работы со слоями и стилями.

JavaScript — это объектно-ориентированный язык программирования сценариев с синтаксисом, немного аналогичным синтаксисам языков C, Perl и Python. Несмотря на то что язык имеет схожее название с языком программирования Java, ничего общего, кроме части названия, у них нет.

В настоящее время JavaScript используется в основном для создания встраиваемых в веб-страницы сценариев, позволяющих полностью управлять как самими веб-страницами, там и веб-браузерами, в которых эти веб-страницы открыты. Таким образом, язык JavaScript в большинстве случаев используется для создания интерактивных веб-страниц и веб-приложений.

Сеть Интернет построена по технологии «клиент-сервер».

Серверы — это компьютеры или программы, которые предоставляют какие-либо сервисы клиентам, например веб-сервер или сервер электронной почты.

Клиенты — это компьютеры или программы, которые используют то, что предлагают серверы, например веб-браузеры или почтовые клиенты.

Сценарии JavaScript — это небольшие программы, которые выполняются на компьютере пользователя при загрузке с сервера вместе с веб-страницами.

Область применения JavaScript

С самого начала своего развития язык JavaScript применялся для написания различных клиентских сценариев. Они широко применялись для решения таких задач, как, например, проверка информации, введенной пользователем в форму, перед ее отправкой на сервер или программирование ответных реакций на действия пользователя, делающие веб-страницы интерактивными. Сегодня с помощью языка JavaScript создаются уже целые веб-приложения, некоторые из них мало чем уступают своим настольным «собратьям». Хорошие примеры таких веб-приложений можно найти среди сервисов компании Google, например Google Calendar — многофункциональный органайзер в веб-браузере и Google Doc & Spreadsheet — текстовый и табличный редактор, позволяющий работать с офисными документами прямо в окне веб-браузера. Для создания динамического содержимого веб-страниц

используется язык создания динамических страниц HTML Dynamic HTML (Динамический HTML), ключевой частью которого является опять же JavaScript.

Язык JavaScript применятся не только в Интернете, но и в таких программах, как, например, Adobe Dreamweaver, Adobe Acrobat Reader и Adobe Photoshop для расширения их возможностей, аналогично использованию языка Visual Basic в Microsoft Office.

Теперь рассмотрим создание сценариев и способы их внедрения в HTML-документ.

Создание сценариев

Разберем все этапы создания JavaScript-сценария — от организации рабочего окружения до подключения кода к HTML-документу.

Для работы над сценариями подойдет практически любой текстовый редактор, позволяющий создавать обычные текстовые файлы и сохранять их с расширением htm или html. Например, в операционной системе Windows в числе стандартных есть два подходящих текстовых редактора — Блокнот и WordPad. Существуют также редакторы, специально разработанные для создания веб-страниц, например Microsoft FrontPage и Adobe Dreamweaver (ранее известный как Macromedia Dreamweaver). Такие редакторы позволяют создавать веб-страницы с помощью визуальных инструментов, а затем переключаться в режим отображения кода и вносить нестандартные изменения, например добавлять сценарии. В отличие от обычных текстовых редакторов, в таких редакторах подсвечивается синтаксис языка HTML и JavaScript, что очень удобно.

Кроме того, в Интернете вы можете найти большое количество других текстовых редакторов, которые позволяют создавать веб-страницы и подсвечивать синтаксис языков HTML и JavaScript. Эти редакторы имеют также много других полезных возможностей, которые существенно упрощают процессы создания веб-страниц и разработки сценариев. Многие такие редакторы распространяются бесплатно или условно бесплатно, и, возможно, какой-нибудь из них придется вам по душе.

Вы будете создавать сценарии, управляющие веб-страницами, поэтому для проверки работоспособности этих документов вам понадобится какой-либо веб-браузер. Учтите, что, проверив работоспособность своего творения в одном веб-браузере, нельзя быть уверенным в его работоспособности и в других. Следовательно, веб-страницы желательно проверять в нескольких веб-браузерах, и чем их будет больше, тем лучше.

Веб-разработчикам приходится считаться, в основном, с особенностями веб-браузера Microsoft Internet Explorer. Следовательно, свои веб-страницы нужно обязательно протестировать в этом веб-браузере, который имеется в каждой операционной системе Windows. Большой популярностью пользуются также веб-браузеры Mozilla Firefox и Opera, которые стремятся максимально соответствовать стандартам. Mozilla Firefox можно загрузить с сайта http://www.mozilla.ru/, а Opera — с сайта http://www.opera.com/.

Отдельно рассмотрим отладку программы (сценария). Когда сценарий большой, разобраться в нем не так просто и отладка, то есть процесс поиска и устранения ошибок, может занимать большую часть времени разработки всей программы (или сценария). Для упрощения и ускорения отладки созданы специальные программы, называемые *отладчиками*. Типичный отладчик имеет такие возможности, как пошаговое выполнение программы с остановкой в каждой строке кода или в заранее отмеченных для этого точках остановки, а также отслеживание и изменение значений каждой переменной прямо при выполнении программы.

Существует немало программ для отладки сценариев на JavaScript. Например, отладчик Microsoft Script Debugger имеет все необходимое для успешной отладки сценариев на JavaScript. Он применяется совместно с веб-браузером Microsoft Internet Explorer. Этот наладчик распространяется бесплатно компанией Microsoft, его можно загрузить с сайта http://msdn.microsoft.com/scripting/. Кроме того, данный отладчик распространяется вместе с редактором Microsoft FrontPage в рамках программы Microsoft Script Editor, которая практически не отличается от Microsoft Script Debugger.

Существуют также и другие отладчики, например для веб-браузера Mozilla Firefox они созданы в виде расширений. В частности, отладчик FireBug можно загрузить с сайта https://addons.mozilla.org/firefox/1843/, причем если на эту страницу перейти в веб-браузере Mozilla Firefox, то отладчик сразу же установится.

Мы не будем рассматривать процесс отладки в этой книге, так как это отдельная тема, требующая внимательного изучения. Чтобы писать простые сценарии, отладка не требуется.

Встраивание сценариев

Рассмотрим встраивание сценариев в веб-страницы.

Сценарии внедряются в HTML-документ тремя стандартными способами:

- в виде гиперссылки;
- в виде обработчика события;
- в контейнерный тег `<script>`.

Встраивание сценариев в гиперссылку

Как известно, для создания на веб-страницах гиперссылок применяется контейнерный тег `<a>`, имеющий атрибут `href` для указания адреса страницы или файла, к которому приведет гиперссылка. Однако вместо адреса данный атрибут может содержать и JavaScript-сценарий, если перед сценарием поставить слово `javascript:` (со знаком двоеточие). Пример такой гиперссылки следующий:

```
<a href="javascript:alert('Привет, мир!');">Нажмите сюда</a>
```

Когда пользователь щелкает кнопкой мыши на гиперссылке, интерпретатор сценариев JavaScript получает код

```
alert('Привет, мир!');
```

и исполняет его, в результате чего на экране появляется диалоговое окно (рис. 2.1).



Рис. 2.1. Результат перехода по гиперссылке

Здесь сценарий состоит всего из одной функции `alert()`, которая выводит на экран диалоговое окно с текстом, полученным в качестве параметра. Объем кода может быть любым.

Встраивание сценариев для обработки событий

Поведение каждого элемента зависит от событий, которые постоянно происходят в веб-браузере. Допустим, пользователь выполнил следующие действия: щелкнул кнопкой мыши на тексте, переместил указатель мыши и загрузил HTML-документ. Веб-браузер сам решает, как реагировать на каждое событие, однако вы можете «перехватить» отдельные события и задать для них новое поведение. В языке HTML многие теги имеют атрибуты, соответствующие различным событиям. Эти атрибуты могут содержать код языка JavaScript, который выполняется в ответ на событие, например:

```
<p onclick="alert('Не нажимать!');">Это простой текст</p>
```

Контейнерный тег <p> просто объединяет текст в параграфы, однако на него возлагается еще одна функция. Если вы определите атрибут `onclick`, то ваш текст будет реагировать на щелчок кнопкой мыши. Результат обработки кода приводится на рис. 2.2.

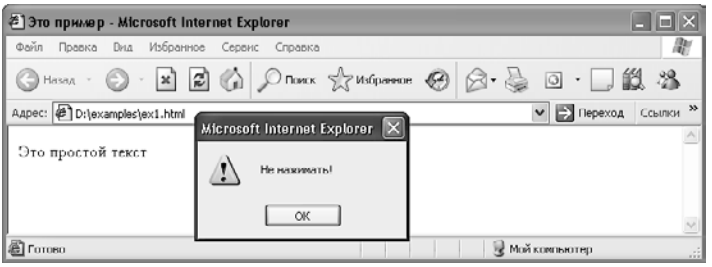


Рис. 2.2. Результат щелчка кнопкой мыши на тексте

В языке HTML практически каждый тег имеет атрибуты для обработки событий. Благодаря возможности обработки событий вы можете создавать интерактивные страницы, которые будут реагировать на действия пользователей.

Встраивание сценариев в тег ‹script›

В HTML есть специальный контейнерный тег `<script>`. Если при разборе кода HTML-документа веб-браузер встретит этот тег, то он интерпретирует последующий до закрывающего тега `</script>` текст как сценарий на каком-либо языке. Язык сценария указывается с помощью атрибута `type`. Например, для указания языка JavaScript используется следующий код:

```
<script type="text/javascript">
  Здесь располагается код сценария
</script>
```

Этот атрибут можно и не указывать, так как значение «`text/javascript`» является значением по умолчанию.

В коде сценария можно использовать *комментарии* — текст, который игнорируется интерпретатором JavaScript. Комментарии `//` и `<!--` позволяют скрыть строку в коде сценария. Ниже приводится код, в котором используются комментарии.

```
<script>
  // Это первый комментарий
  // Второй комментарий, далее идет код
  Код сценария
```

```
  <!-- Еще один комментарий
  Продолжение кода
  // Комментарий
</script>
```

Комментарии могут быть и многострочными, что облегчает отладку сценария и позволяет закомментировать проблемный участок кода. Для этой цели используются наборы символов `/*` и `*/`, как, например, в следующем коде:

```
<script>
  /* Это
  тоже
  комментарий */
  Код сценария
</script>
```

Сценарии могут содержаться в самой веб-странице, как описано выше, и в отдельном файле. Тег `<script>` имеет атрибут `src`, в качестве значения которого применяется адрес файла со сценарием. Обычно файлы со сценариями JavaScript имеют расширение .js, как, например, в файле lib1.js. Таким образом, чтобы подключить к HTML-документу файл lib1.js, необходимо добавить в него следующую строку:

```
<script type="text/javascript" src="lib1.js"></src>
```

Использование отдельных файлов для хранения кодов сценариев под код сценариев позволит вам создавать, например, собственные библиотеки функций и объектов. При необходимости можно просто подключить к веб-странице файл библиотеки, что избавит вас от создания функций с нуля. Расположение часто используемых функций в одном файле упростит также отладку и дальнейшее совершенствование сценариев.

Стоит отметить, что обычно объявления всех глобальных переменных, функций и внешних файлов со сценариями помещают внутри контейнерного тега `<head>`, то есть в заголовке страниц. В листинге 2.1 показываются фрагменты веб-страницы, используемые для встраивания сценариев в HTML-документ.

Листинг 2.1. Пример веб-страницы со сценариями

```
<html>
<head>
  <title>Страница со сценариями</title>
```

```
  <!-- Подключаем внешние файлы со сценариями -->
  <script type="text/javascript" src="lib1.js"></src>
  <!-- Глобальные переменные, функции и объекты -->
  <script type="text/javascript">
    ... // Код на JavaScript
  </script>
</head>

<body>
  ... <!-- Элементы страницы -->

  <script type="text/javascript>
  <!--
     ... // Тоже код на веб-странице
  //-->
  </script>

  ... <!-- Элементы страницы -->

  <p onclick="...">Элемент страницы, события которого обрабатываются</p>

  ... <!-- Элементы страницы -->
</body>
</html>
```

Мы рассмотрели, как можно внедрить в HTML-документ код сценариев на JavaScript. Как видите, все довольно просто. В этом вы убедитесь, когда создадите свой первый сценарий, к составлению которого мы сейчас и перейдем.

Ваш первый сценарий

Классическим примером в начале изучения любого языка программирования является программа, выводящая сообщение «Привет, мир!!!». Вот и мы создадим на JavaScript сценарий, который будет представлять это сообщение.

Прежде стоит отметить, что язык JavaScript учитывает регистр символов. Это означает, что вам следует внимательно следить за тем, какими буквами вы набираете

код сценария. Ведь, например, слова `if` и `If` в языке JavaScript — это абсолютно разные слова. Итак, приступим.

1. Запустите текстовый редактор, в котором вы собираетесь создавать свои веб-страницы.
2. Создайте новый текстовый файл.
3. Введите в этот файл текст своего первого сценария из листинга 2.2.

Листинг 2.2. Ваш первый сценарий

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<title>Наш первый сценарий</title>
</head>
<body>
<script type="text/javascript">
for (var i = 0; i < 10; i++)
{
  document.write("Привет, мир!!!");
}
</script>
</body>
</html>
```

4. Сохраните файл, например с именем `script1.html`.

   Создав веб-страницу со сценарием, можно посмотреть, как она будет выглядеть в веб-браузере. Чтобы убедиться, что сценарий работает как надо, проверьте его во всех установленных на вашем компьютере веб-браузерах.

5. Запустите веб-браузер и откройте в нем созданную веб-страницу.

Итак, если вы все сделали правильно, на экране вашего веб-браузера появится десять сообщений «Привет, мир!!!» (рис. 2.3).

При разработке сценариев обычно одновременно открыты и текстовый редактор, в котором создается веб-страница, и несколько веб-браузеров, в которых проверяется работоспособность создаваемой веб-страницы. После внесения изменений в сценарий достаточно переключиться в веб-браузер и нажать кнопку Обновить.

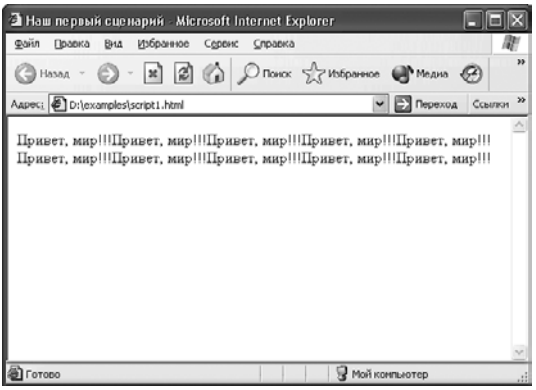


Рис. 2.3. Ваш первый сценарий в веб-браузере

Синтаксис и основные средства языка JavaScript

Теперь разъясним основные элементы языка JavaScript: переменные, типы данных, выражения, различные операторы, функции и т. д. Эти знания являются базовыми, так как без них невозможно понять остальной материал.

Работа с информацией

Любая программа или сценарий работают с информацией, то есть они получают некие данные, обрабатывают их согласно своему алгоритму, а затем, как правило, возвращают некий результат в виде изображения, звука, файла, сигнала другой программе и т. д.

Клиентские сценарии часто выступают буфером между пользователем и серверной программой, осуществляя предварительную обработку и верификацию данных.

Клиентские сценарии могут получать информацию следующими способами:

- данные, заложенные в сценарий разработчиком (начальные значения);
- данные, полученные от пользователя с помощью форм;
- данные, переданные через URL;
- информация, полученная обработкой событий, например при перемещении указателя мыши, щелчке или нажатии клавиш;
- данные, полученные с других сайтов или переданные серверной программой.

Рассмотрим ввод/вывод информации в JavaScript с помощью следующих трех методов: `alert()`, `prompt()` и `confirm()`. Эти методы генерируют различные окна сообщений.

Метод alert()

Метод `alert()` отображает диалоговое окно предупреждения с соответствующим сообщением. После прочтения сообщения пользователь должен щелкнуть на кнопке OK, чтобы закрыть окно. Аргументом данного метода является строка.

В простейшем случае текст предупреждения, заключенный в кавычки, вводится внутри круглых скобок (листинг 2.3).

Листинг 2.3. Работа с предупреждением

```
<html>
  <head>
    <title>Работа с предупреждением</title>
  </head>
  <script>
    alert("Это мое предупреждение");
  </script>
  <body>
  </body>
</html>
```

Окно предупреждения генерируется самим браузером, поэтому внешний вид окна в различных браузерах может различаться. На рис. 2.4 приводится вид окна, сгенерированного с помощью представленного кода в браузере Internet Explorer.

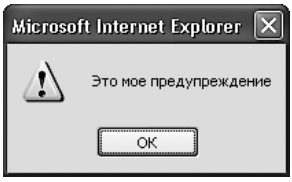


Рис. 2.4. Работа с предупреждением

Метод prompt()

Метод `prompt()` служит для получения данных от пользователя (листинг 2.4).

Листинг 2.4. Работа с запросом

```
<html>
  <head>
    <title>Работа с запросом</title>
  </head>
```

```
  <script>
    //Объявляем переменную
    var nameUser;
    //Введенное пользователем значение присваивается переменной nameUser
    nameUser=prompt("Здравствуйте, как вас зовут?", "аноним");
    //Используем метод alert() для вывода введенного имени
    alert("Рад вас видеть, "+ nameUser);
  </script>
  <body>
  </body>
</html>
```

При его вызове отображается диалоговое окно приглашения с текстовым полем. Этот метод может содержать два аргумента. Первый аргумент — сообщение, отображаемое в диалоговом окне. Второй аргумент — текст по умолчанию, который должен появиться в соответствующем поле. Кроме того, этот метод возвращает значение — текст, который ввел пользователь.

После запуска страницы с вышеприведенным кодом появится запрос (рис. 2.5), вслед за которым отобразится предупреждение.

В этом примере для демонстрации работы метода `prompt()` используется переменная, а в методе `alert()` в качестве аргумента выступает выражение. Эти понятия разберем ниже.

В методе `prompt()` можно опустить второй аргумент, то есть указать только одну строку.

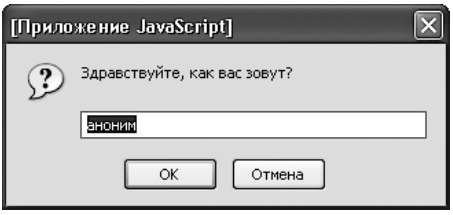


Рис. 2.5. Работа с запросом

Метод confirm()

Метод `confirm()` отображает диалоговое окно подтверждения, которое содержит две кнопки: OK и Cancel (Отмена). В листинге 2.5 приводится код для создания окна подтверждения, однако в этом сценарии не подчеркивается разница между кнопками OK и Cancel.

Листинг 2.5. Работа с подтверждением

```
<html>
  <head>
```

```
    <title>Работа с подтверждением</title>
  </head>
  <script>
    confirm("Это мое подтверждение");
  </script>
  <body>
  </body>
</html>
```

Щелчок на любой из кнопок закроет диалоговое окно (рис. 2.6).

Сценарий может по-разному реагировать на щелчки кнопкой мыши по кнопкам OK и Cancel.

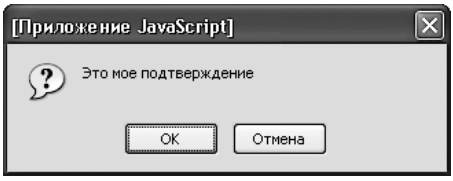


Рис. 2.6. Работа с подтверждением

Переменные и типы данных

Во время интерпретации сценария браузер разбивает код на отдельные символы, слова или фразы, которые сможет распознать. Эти элементы называются *лексемами*. В языке JavaScript лексемы делятся на четыре типа: идентификаторы, ключевые слова, литералы и операции.

Идентификаторы

Идентификаторы — это имена, которые обозначают переменные, функции и объекты. Некоторые имена не могут использоваться в качестве идентификатора, так как имеют особый смысл. Идентификаторы образуются с помощью комбинаций различных символов со следующими ограничениями:

- все идентификаторы должны начинаться с буквы;
- после первой буквы остальными символами могут быть буквы и цифры;
- буквами считаются все заглавные и строчные буквы латинского алфавита: от A до Z и от a до z;
- символ подчеркивания (_) выступает в качестве буквы и часто используется вместо пробела, который нельзя использовать в идентификаторах;
- символ доллара ($) выступает в качестве буквы и обычно используется при автоматической генерации кода.

СОВЕТ



Не следует использовать в одном сценарии идентификаторы, отличающиеся только символами верхнего и нижнего регистра, например flagId и FlagID. Это различные идентификаторы, которые могут вызывать трудноуловимые ошибки.

В табл. 2.1 приводятся примеры допустимых и недопустимых идентификаторов.

Таблица 2.1. Примеры идентификаторов JavaScript

Допустимые	Недопустимые
count	2my
x	new
X	my№
is_well	is well

Обратите внимание, что идентификатор `new`, хотя и содержит разрешенные символы, относится к числу ключевых слов, поэтому не может выступать в качестве пользовательского идентификатора.

Ключевые и зарезервированные слова

Ключевые слова — это предопределенные идентификаторы, которые образуют ядро языка JavaScript. Эти слова имеют особый смысл и выполняют определенные функции. Ключевые слова не могут использоваться для пользовательских идентификаторов. В табл. 2.2 представлен список ключевых слов JavaScript.

Таблица 2.2. Ключевые слова JavaScript

break	else	new	typeof
case	finally	null	var
catch	for	return	void
continue	function	switch	while
default	if	this	with
delete	in	throw	
do	instanceof	try	

Зарезервированные слова не рекомендуется применять в качестве пользовательских идентификаторов, так как в дальнейшем они могут использоваться как ключевые слова. Список зарезервированных слов приводится в табл. 2.3.

Таблица 2.3. Зарезервированные слова JavaScript

abstract	enum	int	short
boolean	export	interface	static
byte	extends	long	super
char	final	native	synchronized
class	float	package	throws
const	goto	private	transient
debugger	implements	protected	volatile
double	import	public	

Литералы

Литералы — это числа или строки, которые используются для представления значений в JavaScript. Информация может быть очень разнообразной, поэтому значения бывают различных видов. Простейшие типы данных в JavaScript называются *основными типами данных*: числа, строки и логические значения.

Числа в языке JavaScript бывают двух типов: целые числа и числа с плавающей точкой. Целочисленные величины могут быть положительными, например 1, 2, 3, отрицательными, например –1, –2, –3, и равными нулю. Кроме того, целочисленные величины могут выражаться в десятичной, восьмеричной или шестнадцатеричной системах счисления.

Числа в десятичном формате могут включать любую последовательность цифр от 0 до 9, которая не начинается с нуля.

Числа в восьмеричном формате могут включать любую последовательность цифр от 0 до 7, которая обязательно начинается с нуля.

Числа в шестнадцатеричном формате могут включать любую последовательность цифр от 0 до 9 и букв от a до f, которая обязательно начинается с 0x.

Рассмотрим примеры представления целых чисел в различных форматах (табл. 2.4).

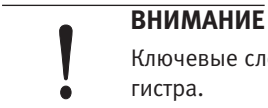
Таблица 2.4. Примеры целых чисел

Десятичный формат	**Восьмеричный формат**	**Шестнадцатеричный формат**
45	055	0x2d
– 255	– 0377	– 0xff
10000000	046113200	0x989680
10	012	0XA

Числа с плавающей точкой определяют десятичные числа с дробной частью. Эти числа могут выражаться в обычном виде или в экспоненциальном представлении. В последнем случае для представления порядка используется символ `e` или `E`. Десятичная мантисса и порядок могут быть положительными либо отрицательными. Ниже приводятся примеры чисел с плавающей точкой.

```
1.41421356237309504880168872420 97
-35.0
4567.0002
3.4e100
-5.456e-3
0.007
```

В языке JavaScript поддерживаются логические типы данных, которые могут принимать лишь два значения: `true` (истина) и `false` (ложь). Логические (булевые) величины необходимы для работы с условиями, с которыми вы познакомитесь позднее в этой главе.



ВНИМАНИЕ

Ключевые слова true и false обязательно должны вводиться буквами нижнего регистра.

Наиболее часто используемым типом данных в JavaScript наверняка является строка. *Строка* — это набор символов (возможно пустой), заключенный в одинарные или двойные кавычки. Два типа кавычек используются потому, что строка может содержать кавычки какого-либо вида. Далее представлены примеры строковых литералов:

```
"Строка текста"
"Россия!"
"4567"
""
'Операционные системы'
'Льюис Кэрролл "Алиса в стране чудес"'
```

Иногда может возникнуть необходимость дать компьютеру команду на использование специальных символов, например табуляции или перевода строки. Чтобы включить в строку любой Unicode-символ, нужно ввести его код после служебного символа `\u` в виде `\uXXXX`. В табл. 2.5 приводятся наиболее часто используемые управляющие символы.

Таблица 2.5. Управляющие символы

Управляющая последовательность	Unicode	Описание	Символ
\b	\u0008	Возврат на одну позицию (backspace)	<BS>
\t	\u0009	Горизонтальная табуляция	<HT>
\n	\u000A	Перевод строки (новая строка)	<LF>
\v	\u000B	Вертикальная табуляция	<VT>
\f	\u000C	Подача страницы	<FF>
\r	\u000D	Возврат каретки	<CR>
\"	\u0022	Двойная кавычка	"
\'	\u0027	Одинарная кавычка	'
\\	\u005C	Обратная косая черта	\

Часто управляющие символы используются для выравнивания данных (листинг 2.6).

Листинг 2.6. Выравнивание данных с помощью управляющих символов

```
<html>
  <head>
    <title> Выравнивание с помощью управляющих символов </title>
  </head>
  <script>
    alert("Имя:\t\tВася\nФамилия:\t\tПетров\nКомпания:\tВася&К");
  </script>
  <body>
  </body>
</html>
```

Диалоговое окно, создаваемое методом `alert()`, зависит от браузера, поэтому и выравнивание в различных браузерах происходит по-разному. Вышеприведенный код адаптирован к программе Microsoft Internet Explorer (рис. 2.7).



Рис. 2.7. Выравнивание с помощью управляющих символов

Помимо чисел, строк и логических величин, существует еще два специальных типа данных: функции и объекты. Функции могут быть встроенными, как, например, `alert()`, а могут создаваться разработчиком. Кроме того, функции могут принадлежать объекту (в этом случае

они называются *методами*). Объекты также могут быть встроенными (например, `document`), а могут создаваться программистом. Считается, что значение `null` является объектом.

Переменные

Переменная — это имя, присваиваемое ячейке памяти компьютера, которая хранит определенные данные во время работы сценария. Переменные упрощают работу программиста по манипулированию данными.

Имя переменной является идентификатором, поэтому подчиняется тем же правилам.

Перед использованием переменной ее необходимо объявить, то есть зарезервировать место для хранения данных. Чтобы объявить переменную, после ключевого слова `var` следует указать ее идентификатор. Можно сразу объявить несколько переменных, указав их имена через запятую:

```
var myData1;
var x,y,z;
var k, msg1, msg2, IM;
```

В процессе объявления переменной ее можно проинициализировать, то есть задать для нее начальное значение:

```
var myStr="Здравствуйте, ";
var k=1000, x=12, y=-5;
var s=1.34e-5, msg11="Error", Flag=false;
```

Пока переменная не получит значение, она является неопределенной — `undefined`. Чтобы определить тип переменной, можно воспользоваться операцией `typeof`, которая возвращает строку с типом переменной (листинг 2.7).

Листинг 2.7. Типы данных

```
<html>
  <head>
    <title>Типы данных</title>
  </head>
  <body>
    <h2>Типы данных</h2>
    <pre>
      <script>
```

```
        //Объявляем переменные
        var i=45.78, msg="Строка";
        var f=false, y;
        //Выводим типы переменных
        document.write("\n"+"Переменная i="+i+" - "+ typeof(i)+"\n");
        document.write("Переменная msg="+msg+" - "+ typeof(msg)+"\n");
        document.write("Переменная f="+f+" - "+ typeof(f)+"\n");
        document.write("Переменная y ="+y+" - "+ typeof(y)+"\n");
        //Инициализируем переменную y
        y=null;
        document.write("А теперь переменная y="+y+" - "+ typeof(y)+"\n");
      </script>
    </pre>
  </body>
</html>
```

В вышеприведенном коде для вывода информации мы воспользовались методом `document.write()`, который позволяет дописывать строку текста прямо в страницу (рис. 2.8).

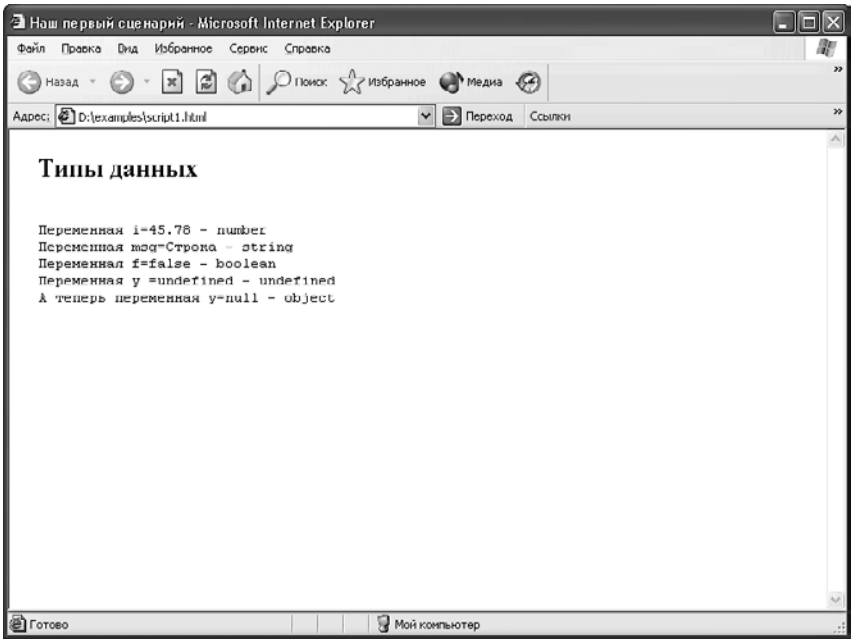


Рис. 2.8. Типы данных

Для вывода текста, значения переменных и их типа аргументом данного метода выступает довольно сложное выражение. Более подробно выражения будут разъясняться ниже.

Переменные по области действия делятся на глобальные и локальные. Переменная, объявленная внутри функции, является локальной, и только эта функция имеет доступ к ее значению. Локальные переменные создаются и уничтожаются вместе с соответствующей функцией. Глобальные переменные объявляются вне функций. Такие переменные позволяют функциям обмениваться данными.

Массивы

Массив — это особая переменная, позволяющая хранить сразу несколько значений. Обычно эти значения связаны между собой, например названия месяцев и дни недели. Массивы могут существенно упростить код и при использовании циклов снизить трудоемкость разработки сценария.

Рассмотрим пример объявления и задания значений элементов массива (листинг 2.8).

Листинг 2.8. Работа с массивом

```
<html>
  <head>
    <title>Работа с массивом</title>
    <script>
      //Объявляем массив с 5 элементами
      var stars= new Array(5);
      //Задание значений элементам массива
      stars[0]='Сириус';
      stars[1]='Канопус';
      stars[2]='Арктур';
      stars[3]='Капелла';
      stars[4]='Вега';
      //Вывод первого элемента массива
      alert(stars[0]);
      //Вывод всех элементов массива
      alert(stars);
    </script>
  </head>
```

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

Обратите внимание, что при объявлении массива количество элементов указывается в круглых скобках, а при обращении к элементу его индекс задается в квадратных скобках. Кроме того, нумерация элементов массива в языке JavaScript начинается с нуля, поэтому первым элементом массива будет элемент `stars[0]`.

JavaScript дает возможность вывести все элементы массива, а не перечислять их по одному. Для этого необходимо обратиться к самому массиву. В результате получится строка, в которой все элементы массива перечисляются через запятую (рис. 2.9).

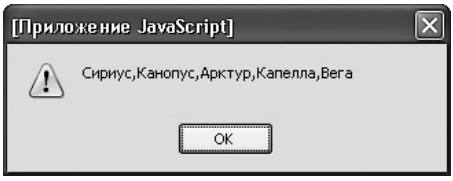


Рис. 2.9. Вывод всех элементов массива

Значения элементов массива можно задать при объявлении массива. В этом случае не нужно указывать количество элементов в массиве (листинг 2.9).

Листинг 2.9. Инициализация массива

```
<html>
  <head>
    <title>Инициализация массива</title>
    <script>
      //Объявляем массив и инициализируем значения элементов
      var stars= new Array('Сириус','Канопус','Арктур','Капелла','Вега');
      //Вывод первого элемента массива
      alert(stars[0]);
      //Вывод всех элементов массива
      alert(stars);
    </script>
  </head>
  <body>
  </body>
</html>
```

Размерность массива можно не указывать, так как язык JavaScript этого не требует. Массив можно расширять по мере добавления данных. Более того, можно объявить элемент с индексом `n`, что увеличит длину массива до `n+1`. Узнать длину массива можно с помощью свойства `length` (листинг 2.10).

Листинг 2.10. Размер массива

```
<html>
  <head>
    <title>Размер массива</title>
    <script>
      //Объявляем массив и инициализируем значения элементов
      var stars= new Array('Сириус','Канопус','Арктур','Капелла','Вега');
      stars[99]='Ригель';//Создаем сотый элемент
      //Вывод размера массива
      alert(stars.length);
    </script>
  </head>
  <body>
  </body>
</html>
```

Несмотря на то что фактически элементов в массиве 6, длина массива равна 100 (рис. 2.10).



Рис. 2.10. Размер массива

Выражения

Выражения — это операнды, скомбинированные с помощью операций. Обычно в качестве операндов выступают значения и переменные, но могут выступать и другие выражения. Например, выражение `4+6` складывает два значения, в результате чего получается значение `10`. А выражение `q=5` присваивает переменной `q` значение `5`. Если манипуляции проводятся с одной величиной, то операция называется *унарной*, а если над двумя — *бинарной*. Далее рассмотрим операции в JavaScript.

Операции присвоения

Одной из часто используемых операций является операция присвоения. В простейшем случае эта операция присваивает значение переменной:

```
a=7
```

Эта операция может использоваться одновременно для нескольких переменных:

```
msg1=msg2="Привет"
```

Другие операции присвоения являются комбинациями операции присваивания и арифметической или поразрядной операций (табл. 2.6).

Таблица 2.6. Комбинированные операции присваивания

Тип комбинации	Операция	Пояснение
Комбинация с арифметическими операциями	x += y	x = x + y
	x – = y	x = x – y
	x *= y	x = x * y
	x /= y	x = x / y
	x %= y	x = x % y
Комбинация с поразрядными операциями	x <<= y	x = x << y
	x >>= y	x = x >> y
	x >>>= y	x = x >>> y
	x &= y	x = x & y
	x ^= y	x = x ^ y
	x |= y	x = x | y

Арифметические операции

Для работы с числами используются арифметические операции.

- Сложение — знак плюс (+). Например, 5 + 7 = 12.
- Вычитание — знак минус (–). Например, 67 – 43 = 24.
- Умножение — звездочка (*). Например, 2 * 2 = 4.
- Деление — косая черта (/). Например, 45 / 5 = 9.
- Остаток от деления — процент (%). Например, 7 % 5 = 2.

В программировании очень часто встречаются операции увеличения или уменьшения переменной на единицу (операции инкремента и декремента). Инкремент обозначается последовательностью **++**, а декремент — последовательностью **--**. Например, `i++` является альтернативой выражению `i=i+1`.

Различают префиксную и постфиксную форму этих операций. В случае префиксной операции вначале производится инкремент или декремент, а затем вычисляется выражение:

```
i = 4
++i * 2 = 10
```

Если же используется постфиксная операция, то вначале вычисляется выражение, а затем выполняется увеличение или уменьшение переменной:

```
i = 4
i++ * 2 = 8
```

В листинге 2.11 демонстрируется работа с арифметическими операциями.

Листинг 2.11. Арифметические операции

```
<html>
  <head>
    <title>Арифметические операции</title>
  </head>
  <body>
    <h2>Арифметические операции</h2>
    <pre>
      <script>
        var i=5, z;
        document.write("\nПеременная i="+i+"\n");
        z=i+56*2;
        document.write("z=i+56*2="+z+"\n");
        z=6.7*2-i;
        document.write("z=6.7*2-i="+z+"\n");
        z=i++ +10;
        document.write("z=i++ +10="+z+"\n");
        document.write("Переменная i="+i+"\n");
        z=--i *4;
        document.write("z=--i *4="+z+"\n");
        document.write("Переменная i="+i+"\n");
        z=(5+11)%8;
        document.write("z=(5+11) %8="+z+"\n");
      </script>
    </pre>
  </body>
</html>
```

В этом примере можно заметить, что арифметические операции подчиняются математическим правилам. Например, умножение совершается до сложения (рис. 2.11).

Это выполняется за счет приоритета операций, который можно изменить с помощью скобок.

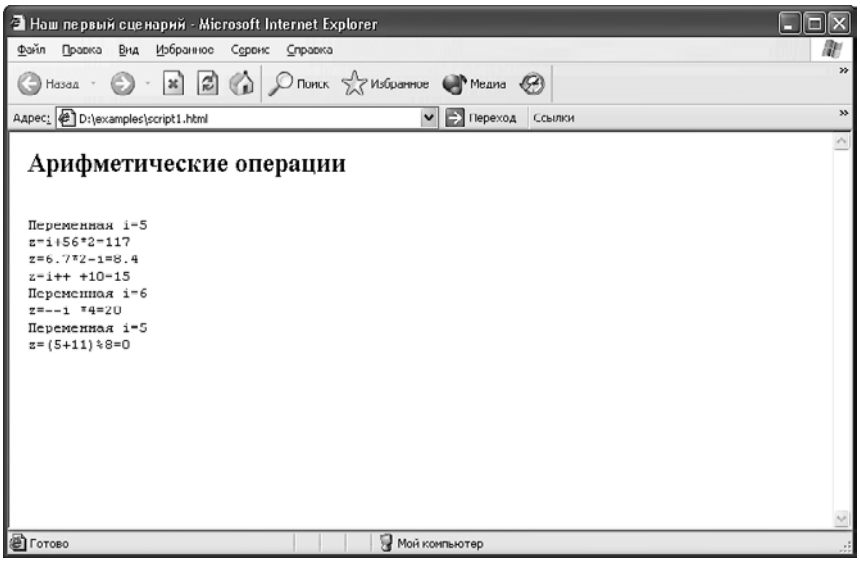


Рис. 2.11. Арифметические операции

Операции сравнения

Операции сравнения используются для сопоставления операндов. В этих операциях операндами могут быть не только числа, но и строки, логические величины и объекты. В табл. 2.7 приводятся все операции сравнения.

Таблица 2.7. Операции сравнения

Операция	Описание
==	Равенство — возвращает true, если операнды равны
!=	Неравенство — возвращает true, если операнды не равны
>	Больше — возвращает true, если первый операнд больше второго
>=	Больше или равно — возвращает true, если первый операнд больше или равен второму
<	Меньше — возвращает true, если первый операнд меньше второго
<=	Меньше или равно — возвращает true, если первый операнд меньше или равен второму
===	Идентичность — возвращает true, если операнды равны без преобразования типов
!==	Неидентичность — возвращает true, если операнды не равны без преобразования типов

В листинге 2.12 показывается код, в котором используются операции сравнения.

Листинг 2.12. Операции сравнения

```
<html>
  <head>
    <title>Операции сравнения</title>
  </head>
  <body>
    <h2>Операции сравнения</h2>
    <pre>
      <script>
        var i=5, m1="строка1";
        var m2;
        document.write("\nПеременная i="+i+"\n");
        document.write("i>7 - ");
        //Результат сравнения
        document.write(i>7);
        document.write("\n(3+i)>=8 - ");
        //Результат сравнения
        document.write((3+i)>=8);
        m2="строка1"
        document.write("\nПеременная m1="+m1);
        document.write(", переменная m2="+m2+"\n");
        document.write("m1!=m2 - ");
        document.write( m1!=m2);
        m2="строка2"
        document.write("\nПеременная m1="+m1+", переменная m2="+m2+"\n");
        document.write("m1<=m2 - ");
        document.write(m1<=m2);
        m2="5"
        document.write("\nПеременная i="+i+", переменная m2="+m2+"\n");
        document.write("i==m2 - ");
        document.write(i==m2);
        document.write("\ni===m2 - ");
        document.write( i===m2);
        document.write("\ni===5 - ");
```

```
        document.write(i===5);
      </script>
    </pre>
  </body>
</html>
```

В приведенном коде сравниваются переменные `i=5` и `m2="5"`. Они считаются равными, но не идентичными (рис. 2.12).

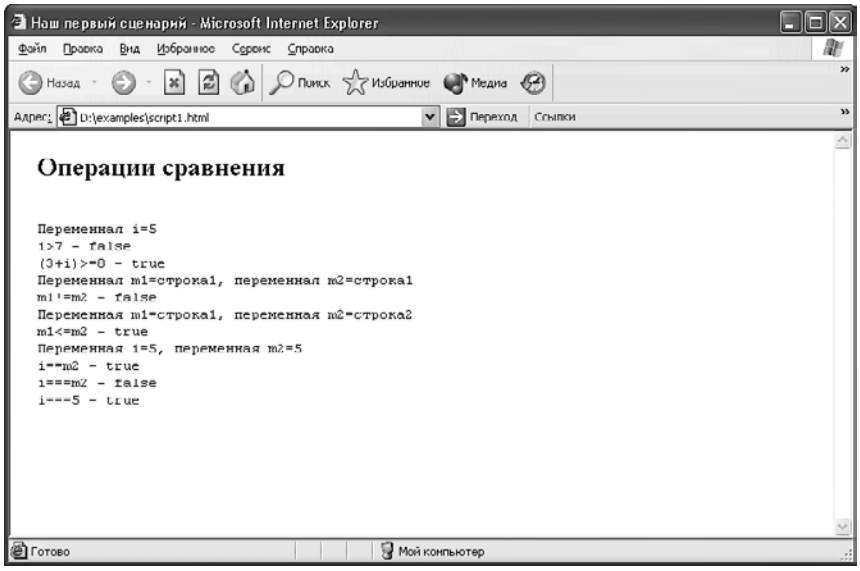


Рис. 2.12. Операции сравнения

Операции над строками

Как уже говорилось, строки можно сравнивать. Кроме того, их можно объединять с помощью операции конкатенации (+), что мы уже не раз делали. Например:

```
S1="Здравствуй";
S2=",";
S3="Мир!";
S=S1+S2+" "+S3;
```

В итоге переменная S будет содержать строку `"Здравствуй, Мир!"`.

Благодаря автоматическому приведению типов можно объединять числа и строки:

```
"год"+2008="год 2008"
```

Логические операции

Логические операции позволяют комбинировать выражения, возвращающие логические величины. Язык JavaScript поддерживает три логические операции.

Операция *логического И* (&&) возвращает `true`, только если оба операнда истинны. Например, `(1<7)&&(3>2)`.

Операция *логического ИЛИ* (||) возвращает `true`, если хотя бы один операнд истинен. Например, `(2<3)||(1>2)`.

Операция *логического НЕ* (!) является унарной и изменяет значение логической величины на обратное.

Условные операции

В JavaScript есть операция `?:`, которая присваивает значение переменной в зависимости от выполнения условия. Рассмотрим следующий пример:

```
var sign = (a>=0) ? "Положительное": "Отрицательное";
```

Результат выражения до вопросительного знака влияет на значение переменной: если значение истинно, то вычисляется выражение до двоеточия, если ложно, то определяется выражение, стоящее после двоеточия. В нашем случае, если переменная `a` больше или равна 0, переменная `sign` принимает значение `"Положительное"`, в противном случае — значение `"Отрицательное"`.

В JavaScript существуют также операции (например, поразрядные), которые встречаются довольно редко и нужны при решении специфических задач, поэтому их мы рассматривать не будем.

Условия и циклы

Для изменения порядка выполнения кода в JavaScript имеются механизмы ветвления (операторы `if` и `switch`), позволяющие выбрать одно из альтернативных действий в зависимости от условия. Для многократного выполнения кода используются операторы цикла (`while`, `for` и `do`). Циклы также удобны для манипулирования массивами.

Оператор if

Оператор `if` управляет последовательностью выполнения команд. Синтаксис этого оператора таков:

```
if (логическое выражение)
{
 операторы
}
```

Вначале вычисляется `логическое выражение`, затем, если оно равно `true`, выполняются `операторы`; если же оно равно `false`, то `операторы` пропускаются и продолжается выполнение сценария.

Рассмотрим пример. Допустим, функция $f(x)$ определяется следующим образом:

$$f(x) = \begin{cases} x + 10, & \text{при } x < 0 \\ x2 + 4, & \text{при } 0 \le x \le 5 \\ 5 / x, & \text{при } x > 5 \end{cases}$$

Для определения значения этой функции в определенной точке можно воспользоваться кодом, приведенным в листинге 2.13.

Листинг 2.13. Пример работы оператора if

```
<html>
  <head>
    <title>Пример работы оператора if</title>
      <script>
        var x, y;
        //Запрашиваем значение x
        x=prompt ("Введите значение x","0");
        //Преобразуем введенную строку в число
        x=+x;
        //Определяем значение функции
        if (x<0){
          y=x+10;
        }
        if ((x>=0)&&(x<=5)){
          y=x*x+4;
        }
        if (x>5){
          y=5/x;
```

```
        }
        alert("Функция f("+x+")="+y);
      </script>
  </head>
  <body>
  </body>
</html>
```

Оператор if..else

Часто удобно использовать полную форму условного оператора. С помощью этого оператора можно задать действия, которые необходимо выполнить, если `логическое выражение` равно `false`:

```
if (логическое выражение)
{
 операторы1
}
else
{
 операторы2
}
```

Более того, можно совместить `else` с другим оператором `if`. В этом случае вы можете рассмотреть несколько альтернативных вариантов и выполнить соответствующие операторы:

```
if (логическое выражение1)
{
 операторы1
}
else if (логическое выражение2)
{
 операторы2
}
else
{
 операторы3
}
```

Предыдущий сценарий перепишем для определения функции с помощью оператора `if..else` (листинг 2.14).

Листинг 2.14. Пример работы оператора if..else

```
<html>
  <head>
    <title>Пример работы оператора if..else</title>
      <script>
        var x, y;
        //Запрашиваем значение x
        x=prompt ("Введите значение x","0");
        //Преобразуем введенную строку в число
        x=+x;
        //Определяем значение функции
        if (x<0) y=x+10; //при x<0
        else if (x>5) y=5/x; //при x>5
        else  y=x*x+4; //при 0<=x<=5
        alert("Функция f("+x+")="+y);
      </script>
  </head>
  <body>
  </body>
</html>
```

Как видно из данного кода, фигурные скобки не обязательны, если выполняется только один оператор. Однако рекомендуется всегда их употреблять для упрощения чтения кода.

Оператор switch

Оператор `switch` сравнивает значение со множеством других значений. Этого же эффекта можно достичь с помощью нескольких операторов `if`, однако оператор `switch` позволяет создать краткий и наглядный код:

```
switch (выражение)
{
 case значение1:
  операторы
 case значение2:
```

```
  операторы
  ..
 default:
  операторы
}
```

Допустим, необходимо создать сценарий, который переводил бы названия животных с английского языка (листинг 2.15).

Листинг 2.15. Пример работы оператора switch

```
<html>
  <head>
    <title>Пример работы оператора switch</title>
      <script>
        var trans;
        //Запрашиваем у пользователя имя животного
        var beast=prompt ("Введите название животного","dog");
        //Находим соответствующее название на русском языке
        switch (beast)
        {
          case "dog":
           trans="собака";
           break;
          case "cat":
           trans="кошка";
           break;
          case "cow":
           trans="корова";
           break;
          default:
           trans="неизвестное животное"
        }
        alert(beast+" – "+trans);
      </script>
  </head>
  <body>
```

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

Этот код позволяет перевести одно из трех названий животных (рис. 2.13).

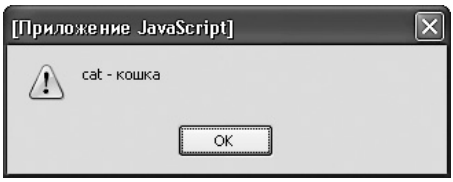


Рис. 2.13. Результат работы оператора switch

Обратите внимание на оператор `break`, который прекращает выполнение оператора `switch`. В противном случае оставшийся код будет исполняться в операторе `switch` независимо от значения переменной.

Цикл for

Оператор `for` используется для создания цикла. Этот оператор имеет следующий синтаксис:

```
for (выражение инициализации; выражение условия; выражение цикла)
{
 операторы
}
```

`Выражение инициализации` обычно служит для задания начального значения счетчика цикла. `Выражение условия` прекращает выполнение цикла, когда оно принимает значение `false`. `Выражение цикла` обычно осуществляет инкремент или декремент счетчика цикла. Любое из этих выражений может быть пропущено, но соответствующая точка с запятой ставиться должна.

В листинге 2.16 представлен код, позволяющий найти факториал числа $n! = 1 \cdot 2 \cdot 3 \cdot ... \cdot n$ с помощью цикла `for`.

Листинг 2.16. Нахождение факториала с помощью оператора for

```
<html>
  <head>
    <title>Факториал с помощью оператора for</title>
      <script>
        var f=1;
        var x=prompt ("Введите число","5");
        x=+x;
        for(var i=1;i<=x;i++)
        {
```

```
          f=f*i;
        }
        alert(f);
      </script>
  </head>
  <body>
  </body>
</html>
```

В этом коде организуется цикл от `1` до `x`, при этом для простоты введенное значение не проверяется на допустимость. Исходя из этого можно, например, получить значение `Infinity` (бесконечность) при большом значении `x`.

Цикл while

Оператор `while` сходен с оператором `for`, но он не производит инициализацию и инкремент счетчика в своем объявлении. Синтаксис этого оператора следующий:

```
while (выражение условия)
{
 операторы
}
```

Если `выражение условия` в цикле `while` ложно, то `операторы` ни разу не выполнятся.

В листинге 2.17 представлен код, позволяющий найти факториал с помощью цикла `while`.

Листинг 2.17. Нахождение факториала с помощью оператора while

```
<html>
  <head>
    <title>Факториал с помощью оператора while</title>
      <script>
        var f=1;
        var x=prompt ("Введите число","5");
        x=+x;
        var i=1;
        while(i<=x)
        {
```

```
        f=f*i;
        i++;
      }
      alert(f);
    </script>
  </head>
  <body>
  </body>
</html>
```

Цикл do..while

Оператор `do..while` практически идентичен оператору `while`. Однако в этом операторе условие проверяется в конце, поэтому он гарантирует выполнение `операторов`, по крайней мере, один раз вне зависимости от истинности условия:

```
do
{
 операторы
} while (выражение условия)
```

В листинге 2.18 представлен код, позволяющий найти факториал с помощью цикла `do..while`.

Листинг 2.18. Нахождение факториала с помощью оператора do..while

```
<html>
  <head>
    <title>Факториал с помощью оператора do..while</title>
      <script>
        var f=1;
        var x=prompt ("Введите число","5");
        x=+x;
        var i=1;
        do
        {
          f=f*i;
          i++;
        } while(i<=x)
        alert(f);
```

```
      </script>
  </head>
  <body>
  </body>
</html>
```

Цикл for..in

Цикл `for..in` позволяет выполнить операторы почти для каждого свойства объекта. Синтаксис этого оператора следующий:

```
for (свойство in объект)
{
 операторы
}
```

Например, с использованием этого оператора можно произвести инкремент всех элементов массива (листинг 2.19).

Листинг 2.19. Использование оператора for..in

```
<html>
  <head>
    <title>Пример использования оператора for..in </title>
      <script>
        var a = new Array (5,76,43,12,77,-5, 0);
        for (var i in a)
        {
          a[i]++;
        }
        alert(a);
      </script>
  </head>
  <body>
  </body>
</html>
```

В результате выполнения этого кода все элементы массива `a` увеличатся на 1 (рис. 2.14).

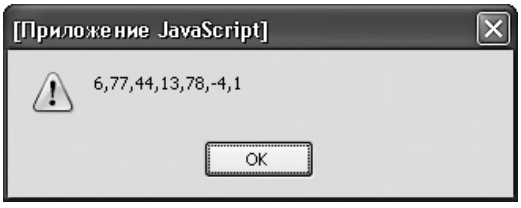


Рис. 2.14. Инкремент элементов массива

Операторы break и continue

Иногда в ходе выполнения цикла возникает необходимость прервать весь цикл или одну из его итераций. Для этой цели служат операторы `break` и `continue`. Оператор `break` полностью прекращает выполнение цикла и передает управление операторам, следующим за циклом. Оператор `continue` прерывает текущее выполнение цикла и переходит к выполнению следующего шага цикла. Различие в работе этих операторов демонстрируется в листинге 2.20.

Листинг 2.20. Использование операторов выхода из цикла

```
<html>
  <head>
    <title>Пример использования операторов выхода из цикла</title>
      <script>
        var a= new Array (5,0,2,0-3,0);
        var b= new Array();
        //Создаем бесконечный цикл
        while(true)
        {
          var x=prompt ("Введите число","0");
          //Преобразуем введенную строку в число
          x=+x;
          //Если введен нуль, то цикл прерывается
          //Это единственный выход из бесконечного цикла
          if (x==0) {break}
          //Вложенный цикл, просматривающий все элементы массива
          for (var i=0; i<a.length; i++)
          {
            //Проверяем значение элемента массива a
            if (a[i]==0)
            {
               b[i]="-";
               //Прерываем текущую итерацию цикла for, чтобы избежать
               //деления на нуль
               continue;
            }
```

```
            //Находим частное
            b[i]=x/a[i];
          }//Конец цикла for
          alert(a+"\n"x+"\n"+b);
        }//Конец цикла while
      </script>
  </head>
  <body>
  </body>
</html>
```

Данный код позволяет определить частное от введенных пользователем числа и элементов массива `a`. При этом среди элементов этого массива могут быть нули, тогда в элемент массива с результатом `b` необходимо ввести прочерк «-». Результат обработки кода в этом случае вы видите на рис. 2.15.



Рис. 2.15. Использование операторов выхода из цикла

Вышеприведенный код также демонстрирует возможность вложения одного цикла в другой. Из цикла `while` можно выйти только тогда, когда введен нуль (значение по умолчанию). Если вводить другие числа, цикл не закончит работу. Цикл `for` во время исполнения иногда прерывается, если элемент массива `a` равен нулю (чтобы избежать деления на нуль).

Функции

До сих пор наши сценарии представляли собой простой код, который выполнялся сразу или ожидал какого-либо события. Однако зачастую требуется многократное исполнение одного и того же кода. Для этого и предназначены функции.

Функция — это набор команд для выполнения некоторых задач, объединенных под общим именем. Синтаксис функции имеет следующий вид:

```
function ИмяФункции([аргумент1][,..аргуметN])
{
 операторы
}
```

Ключевое слово `function` объявляет функцию с именем *ИмяФункции*. В функцию могут передаваться (но могут и не передаваться) аргументы, которые перечисляются через запятую в скобках рядом с именем функции. В качестве аргументов могут выступать переменные, значения и выражения.

Использование функций

Функции можно объявлять в любом месте внутри тега `<script>`. Тем не менее функции рекомендуется объявлять в блоке `<head>` до начала выполнения основного кода, иначе функция может оказаться недоступной при ее вызове, что повлечет за собой ошибку. В листинге 2.21 представлен пример работы с функциями.

Листинг 2.21. Использование функций

```
<html>
  <head>
    <title>Пример использования функций</title>
      <script>
        //Глобальный массив
        var a = new Array (1,2,4,8);
        function sum()
         {
           var s = 0; //локальная переменная для хранения суммы
           //Цикл по всем элементам массива a
           for (var i in a)
           {
            s=s+a[i];
           }
           alert("Сумма элементов массива="+s);
         }
        function product()
         {
           var p=1; //локальная переменная для хранения произведения
           for (var i in a)
           {
            p=p*a[i];
           }
           alert("Произведение элементов массива="+p);
         }
```

```
      </script>
  </head>
  <body>
    <script>
      var x=prompt("Найти сумму или произведение (0 или 1)?","0");
      if (x=="0")
      {
         sum();
      }
      if (x=="1")
      {
        product();
      }
    </script>
  </body>
</html>
```

При запуске этого кода отображается запрос, в ответ на который необходимо ввести 0 или 1. В зависимости от ответа запускается одна из функций — `sum()` или `product()`, — которые находят, соответственно, сумму или произведение элементов глобального массива `a`.

Функции могут возвращать значение. Для этого используется оператор возврата `return`. Вообще, этот оператор может встречаться в теле функции несколько раз. В листинге 2.22 приводится пример использования возврата значения функции.

Листинг 2.22. Использование возврата значения функции

```
<html>
  <head>
    <title>Использование значения функции </title>
      <script>
        function sum(a,b)
         {
           return (a+b);
         }
      </script>
```

```
  </head>
  <body>
    <script>
      var x1=5, x2=6, x3=3, x4=3;
      //Отображаем (x1+x2)*(x3+x4)
      alert (sum(x1,x2)*sum(x3,x4));
    </script>
  </body>
</html>
```

Функция может объявляться в выражении. Как правило, функция присваивается переменной, которая затем может использоваться в выражении:

```
var cube=function(x){return x*x*x}
alert(cube(2)+cube(3));
```

В данном случае объявляется переменная-функция, вычисляющая куб числа.

Переменное количество аргументов

Иногда при создании функции неизвестно, сколько ей будет передаваться аргументов. В этом случае необходимо воспользоваться объектом `arguments`. Все аргументы, переданные функции, сохраняются в этом объекте и могут извлекаться из него. Например, для доступа к первому аргументу можно использовать следующий код:

```
var arg1=arguments[0];
```

Таким образом, аргументы индексируются, начиная с нуля. Для определения общего количества аргументов, переданных функции, можно использовать свойство `length` объекта `arguments`.

Листинг 2.23 демонстрирует возможность обработки переменного количества аргументов.

Листинг 2.23. Использование переменного количества аргументов в функции

```
<html>
  <head>
    <title>Пример переменного числа аргументов в функции </title>
      <script>
        function sum(x1,x2) //Функция содержит не меньше двух аргументов
```

```
        {
          var s=0; //Начальное значение суммы
          //Цикл по всем аргументам
          for (var i=0;i<arguments.length;i++)
          {
            s+=arguments[i];
          }
          return (s);
        }
      </script>
  </head>
  <body>
    <script>
      //Вызываем функцию
      alert (sum(1,2,3,4));
      alert (sum(1,-3,40,2,5,7));
    </script>
  </body>
</html>
```

Дополнительная информация о функциях

Важно помнить, что при передаче функции строки, числа или логической величины в функции создается копия переданного значения и оригинал не изменяется. Другими словами, что бы вы ни делали с аргументом внутри функции, после ее завершения эта переменная уничтожается, а оригинальная переменная остается прежней. При передаче объекта все происходит по-другому: копия объекта не создается, а передается ссылка на этот объект. В противном случае на создание дубликатов терялось бы много времени и памяти. Представьте, сколько требуется времени на передачу массива из 10 000 элементов, когда фактически можно обойтись передачей одного значения — ссылки на этот массив. Однако программист должен помнить, что в функции содержится оригинальный объект.

События

Очень часто язык JavaScript используется для реагирования на события, которые вызывают пользователи и браузеры. К таким событиям относятся открытие новой

страницы, перемещение указателя мыши, щелчок кнопкой мыши и т. д. Каждое событие имеет соответствующий обработчик события, который автоматически реагирует на его возникновение.

Так, событие `onLoad()` будет возникать после загрузки какого-либо элемента, например веб-страницы. Использование обработчика данного события позволяет выполнить JavaScript-код сразу после того, как документ полностью загрузился.

Одним из часто используемых событий является щелчок на объекте. При его возникновении обработчик события `onClick()` объекта выполняет код JavaScript, приведенный в листинге 2.24.

Листинг 2.24. Пример обработчиков событий

```
<html>
  <head>
    <title>Пример обработчиков событий</title>
      <script>
        function mes()
        {
           alert ("Замечательная ссылка");
        }
      </script>
  </head>
  <body>
    <h2>Пример обработчиков событий</h2>
    <a href=http://www.myserver.ru/page1.html onClick="mes()">Ссылка 1</a>
    <br>
    <a href=http://www.myserver.ru/page2.html
      onClick="alert('Еще одна ссылка')">Ссылка 1
    </a>
  </body>
</html>
```

Из этого кода видно, что обработчик события может вызывать функции, а может указывать на JavaScript-код. В данном случае оба обработчика вызывают метод `alert()` (рис. 2.16).

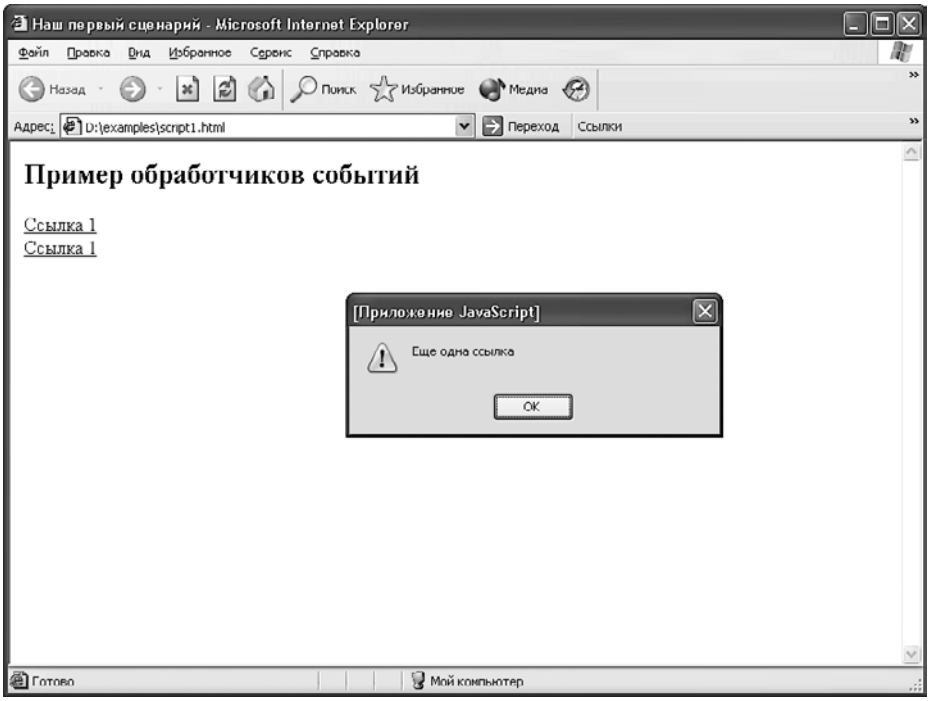


Рис. 2.16. Пример обработчиков событий

Встроенные объекты JavaScript

Объекты JavaScript объединяют в себе переменные (свойства) и функции, управляющие этими свойствами (методы). Кроме того, они могут реагировать на события. Пользователь имеет возможность создавать свои объекты, однако JavaScript предоставляет несколько базовых объектов:

- `Global` — набор высокоуровневых свойств и методов, у которых нет родительского объекта;
- `String` — позволяет форматировать и изменять текстовые строки;
- `Number` — позволяет обрабатывать числа;
- `Boolean` — позволяет создавать логические величины;
- `Array` — позволяет создавать массивы и управлять ими;
- `Function` — позволяет создать функцию;
- `Date` — позволяет работать со временем и датой;
- `Math` — предоставляет некоторые математические функции и константы;

- `RegExp` — позволяет выполнять функции регулярных выражений для заданных строк (в данной главе не рассматривается);
- `Object` — является прототипом всех объектов.

Объект Global

Этот объект содержит несколько методов и свойств, не принадлежащих ни одному объекту (табл. 2.8). Для удобства они объединяются в объект `Global`.

Таблица 2.8. Свойства и методы объекта Global

Тип	Элемент	Описание
Свойство	Infinity	Свойство, представляющее собой положительную бесконечность
	NaN	Свойство, не равное ни одному числу
	undefined	Неопределенное значение
Метод	escape()	Возвращает строку, в которой все символы, не являющиеся алфавитно-цифровыми, преобразуются в цифровые эквиваленты
	eval()	Принимает строку с операторами JavaScript и рассматривает ее как исходный код
	isFinite()	Определяет, имеет ли заданная переменная конечные границы (не является ли NaN или Infinity)
	isNaN()	Определяет, является ли заданная переменная допустимым числом (не является ли NaN)
	parseFloat()	Преобразует строку в число с плавающей точкой
	parseInt()	Преобразует строку в целое число
	string()	Преобразует объект в строку
	unescape()	Принимает шестнадцатеричное значение и возвращает ее ASCII-эквивалент в таблице символов ISO-Latin-1

Для доступа к методам и свойствам этого объекта родительский объект не указывается. В этом примере мы проверяем, не происходит ли деление на нуль:

```
flag=isFinite(x/y);
```

Объект String

Объект `String` предоставляет средства для форматирования и выделения частей строк. Этот объект содержит обширный перечень методов (табл. 2.9).

Таблица 2.9. Свойства и методы объекта String

Тип	Элемент	Описание
Свойство	constructor	Имя функции, создающей прототип объекта
	length	Количество символов в строке
	prototype	Позволяет задавать дополнительные свойства

Методы объекта `String` могут использоваться следующим образом:

```
MyText="Я люблю HTML";
MyText=(MyText.substring(0,8)+"JavaScript").bold();
```

В результате строка `MyText` примет значение `"<b>Я люблю JavaScript</b>"`.

Объект Number

Объект `Number` обычно используется для доступа к некоторым постоянным значениям, например к отрицательной бесконечности. Свойства и методы этого объекта указываются в табл. 2.10.

Таблица 2.10. Свойства и методы объекта Number

Тип	Элемент	Описание
Свойство	constructor	Ссылка на функцию, создающую прототип объекта
	MAX_VALUE	Наибольшее число в машинном представлении (1,7976931348623157e+308)
	MIN_VALUE	Наименьшее число в машинном представлении (5e-324)
	NaN	Специальное значение «не число»
	NEGATIVE_INFINITY	Специальное значение отрицательной бесконечности
	POSITIVE_INFINITY	Специальное значение положительной бесконечности
	prototype	Позволяет задать дополнительные свойства
Метод	toSource()	Возвращает строку, создавшую объект Number
	toString()	Преобразует число в строку
	valueOf()	Возвращает числовое значение объекта Number

Объект Boolean

Объект `Boolean` обычно используется для преобразования объекта в логическое значение. Этот объект имеет ограниченное количество методов и свойств, что демонстрируется в табл. 2.11.

Таблица 2.11. Свойства и методы объекта Boolean

Тип	Элемент	Описание
Свойство	prototype	Позволяет задать дополнительные свойства
Метод	toString()	Преобразует логическое значение в строку «true» или «false»

Объект Array

С массивами вы уже познакомились в этой главе. *Массивы* — это объекты, которые имеют свои свойства и методы, облегчающие работу программиста. Свойства и методы этого объекта приводятся в табл. 2.12.

Таблица 2.12. Свойства и методы объекта Array

Тип	Элемент	Описание
Свойство	constructor	Ссылка на функцию, создающую прототип объекта
	index	Для массива, созданного путем сравнения с регулярным выражением, это свойство возвращает индексную позицию, в которой произошло совпадение
	input	Для массива, созданного путем сравнения с регулярным выражением, это свойство возвращает первоначальную строку
	length	Количество элементов в массиве
	prototype	Позволяет задать дополнительные свойства
Метод	concat()	Объединяет два массива и возвращает новый массив
	join()	Объединяет все элементы массива в строку
	pop()	Удаляет последний элемент из массива и возвращает этот элемент
	push()	Добавляет один или несколько элементов в конец массива и возвращает новую длину массива
	reverse()	Инвертирует расположение элементов в массиве
	shift()	Удаляет первый элемент из массива и возвращает этот элемент
	slice()	Удаляет несколько элементов из массива и возвращает новый массив
	splice()	Добавляет и/или удаляет элементы из массива
	sort()	Сортирует элементы массива
	toSource()	Преобразует элементы в строки, заключенные в квадратные скобки
	toString()	Возвращает строку, содержащую в себе значения всех элементов
	unshift()	Добавляет один или несколько элементов в начало массива и возвращает новую длину массива
	valueOf()	Возвращает массив элементов, разделенных запятыми

Объект Function

Объект `Function` позволяет манипулировать функцией как объектом. Этот объект содержит совокупность специфических методов и свойств (табл. 2.13).

Таблица 2.13. Свойства и методы объекта Function

Тип	Элемент	Описание
Свойство	arguments	Массив фактических аргументов функции
	arguments.callee	Ссылка на тело выполняемой функции
	arguments.caller	Ссылка на функцию, вызвавшую данную функцию
	arguments.length	Количество элементов в массиве arguments
	arity	Количество формальных аргументов функции
	constructor	Ссылка на функцию, создающую прототип объекта
	length	Аналогично arity
	prototype	Позволяет задать дополнительные свойства
Метод	apply()	Позволяет создавать всплывающие сообщения
	call()	Позволяет выполнить операцию наследования метода
	toSource()	Возвращает строку, содержащую тело функции
	toString()	Аналогично toSource()
	valueOf()	Возвращает указатель на функцию

Синтаксис объекта `Function` выглядит следующим образом:

```
ИмяОбъектаФункции = new Function ([аргумент1, аргумент2,.. аргументN],
  телоФункции);
```

Пример использования данного объекта таков:

```
sumFunc = new Function ("a","b","return(a+b)");
alert(sumFunc(5,4));
```

Объект Date

Объект `Date` предоставляет значительное количество методов и свойств, связанных с заданием даты и времени (табл. 2.14). Базовой датой принято считать 1 января 1970 г. JavaScript имеет доступ к времени и дате только на клиентской машине, которые могут оказаться неправильными.

Таблица 2.14. Свойства и методы объекта Date

Тип	Элемент	Описание
Свойство	constructor	Ссылка на функцию, создающую прототип объекта
	prototype	Позволяет задать дополнительные свойства

Конструктор этого объекта принимает значение даты в числовом или строковом формате:

```
ИмяОбъектаДаты = new Date();
ИмяОбъектаДаты = new Date(миллисекунды);
ИмяОбъектаДаты = new Date("месяц дд, гггг чч:мм:сс");
ИмяОбъектаДаты = new Date(год, месяц, день [час, минуты, секунды, миллисекунды]);
```

В данном коде элементы имеют следующие значения:

- `Миллисекунды` — целые числа, определяющие количество миллисекунд, прошедших с 00:00:00 1 января 1970 г. (по Гринвичу);
- `Месяц`, `дд`, `гггг`, `чч`, `мм`, `сс` — соответственно месяц, день, год, час, минуты и секунды;
- `Год`, `месяц`, `день`, `час`, `минуты`, `секунды`, `миллисекунды` — целые числа, определяющие соответствующие величины.

Если параметры не указываются, то объект инициализируется текущими датой и временем.

Например, чтобы узнать текущий день недели, можно воспользоваться следующим кодом:

```
MyDate=new Date();
alert(MyDate.getDay());
```

Этот код отобразит число от 0 до 6, соответствующее дню недели (например, 0 соответствует воскресенью).

Объект Math

Объект `Math` предназначен для математических вычислений в JavaScript. Он содержит совокупность математических констант и функций (табл. 2.15).

Таблица 2.15. Свойства и методы объекта Math

Тип	Элемент	Описание
Свойство	E	Константа Эйлера (число e)
	LN10	Константа — натуральный логарифм десяти
	LN2	Константа — натуральный логарифм двух
	LOG10E	Константа — логарифм числа e по основанию 10
	LOG2E	Константа — логарифм числа e по основанию 2
	PI	Константа — число π
	SQRT1_2	Корень квадратный из 1/2
	SQRT2	Корень квадратный из 2
Метод	abs()	Абсолютная величина заданного значения
	acos()	Арккосинус заданного значения (в радианах)
	asin()	Арксинус заданного значения (в радианах)
	atan()	Арктангенс заданного значения (в радианах)
	atan2()	Арктангенс показателя заданного значения или арктангенс частного от деления аргументов (в радианах)
	ceil()	Возвращает наименьшее целое число, большее или равное данному числу
	cos()	Возвращает косинус числа
	exp()	Возвращает число — константу Эйлера, возведенную в заданную степень
	floor()	Возвращает наибольшее целое число, меньшее или равное данному числу
	log()	Возвращает натуральный логарифм числа
	max()	Возвращает максимальное из 2 чисел
	min()	Возвращает минимальное из 2 чисел
	pow()	Возведение числа, заданного в первом аргументе, в степень, указанную вторым аргументом
	random()	Возвращает число, равномерно распределенное в интервале от 0 до 1
	round()	Округляет аргумент до ближайшего целого
	sin()	Возвращает синус числа
	tan()	Возвращает тангенс числа
	sqrt()	Возвращает квадратный корень из числа
	toSourse()	Копия объекта
	toString()	Строковое представление объекта

Объект Object

Все объекты в JavaScript, как встроенные, так и пользовательские, наследуются от объекта `Object`, то есть его методы и свойства присутствуют в каждом объекте, однако их можно и переопределить (табл. 2.16).

Таблица 2.16. Свойства и методы объекта Object

Тип	Элемент	Описание
Свойство	constructor	Ссылка на функцию, создающую прототип объекта
	prototype	Позволяет задать дополнительные свойства
Метод	toLocaleString()	Возвращает строковое представление объекта с учетом интернациональных установок
	toSource()	Возвращает строку, которая создала объект
	toString()	Возвращает строку, содержащую значение объекта
	valueOf()	Возвращает примитивное значение заданного объекта

Объектная модель

JavaScript является объектно-ориентированным языком программирования, то есть все элементы на веб-странице и в окне вашего браузера предстают в виде объектов.

Каждый объект имеет свои свойства, и над ним можно совершать определенные действия. Это позволяет разработчику легко получать доступ к любому элементу веб-страницы.

Мы уже говорили о таких объектах языка JavaScript, как `String`, `Math` и др. Сейчас же речь пойдет об объектах, которые связывают сценарий со страницей и браузером.

Веб-браузер и HTML-документ

Объектами являются элементы рабочей области браузера и HTML-документа. Окно браузера, веб-страница, ее фон, изображения, текст и все, что находится на странице, — это объекты. В языке JavaScript принято различать объекты браузера и объекты HTML-документа. Рассмотрим их подробнее.

Объекты браузера создаются автоматически при загрузке веб-страницы. К ним относятся такие объекты, как `Window`, `Navigator`, `Screen`, `History` и `Location`. Эти объекты можно кратко охарактеризовать следующим образом:

- Window — объект, предоставляющий доступ к окну браузера;
- Navigator — объект, предоставляющий доступ к характеристикам браузера;
- Screen — объект, предоставляющий доступ к характеристикам экрана монитора;
- History — объект, предоставляющий доступ к истории посещенных ссылок;
- Location — объект, содержащий текущий URL-адрес страницы.

В JavaScript окно вашего браузера — это объект `Window`. Этот объект, в свою очередь, содержит другие объекты — элементы оформления, например строку состояния и полосу прокрутки.

Внутри окна браузера размещается веб-страница — HTML-документ. Это объект `Document`. В свою очередь, он содержит другие объекты — объекты HTML — это ссылки, изображения, формы, фон, то есть все, что находится на веб-странице.

Каждый объект обладает своими *свойствами*, или характеристиками. Свойства описывают объекты, как прилагательные описывают существительные. В синтаксисе языка JavaScript свойство любого объекта описывается так: объект.свойство. Например, установить желтый фон веб-страницы в JavaScript можно с помощью следующей конструкции: `document.bgColor = "yellow"`. Здесь `document` — это объект (веб-страница, то есть HTML-документ), `bgColor` — свойство объекта `Document` (фон веб-страницы) и `yellow` — значение свойства `bgColor`.

С объектами можно совершать определенные действия, называемые *методами*. Согласно синтаксису языка JavaScript после метода всегда ставятся скобки по схеме *объект.метод()*. Например, чтобы с помощью языка JavaScript отобразить на веб-странице текст `Hello, world!`, используется следующая конструкция: `document.write("Hello, world!")`. Здесь `document` — это объект, `write` — метод, присущий объекту `Document`, а выражение в скобках — текст, который должен отобразиться на странице в результате использования метода `write`. На языке программирования это означает, что метод возвращает результат.

В JavaScript объекты подчиняются строгой иерархии.

Родительские и дочерние объекты

Согласно правилам языка JavaScript все элементы на веб-странице выстраиваются в иерархическую структуру. Каждый объект является потомком объекта более высокого уровня.

Иерархия объектов JavaScript показывается на следующей схеме (рис. 2.17).

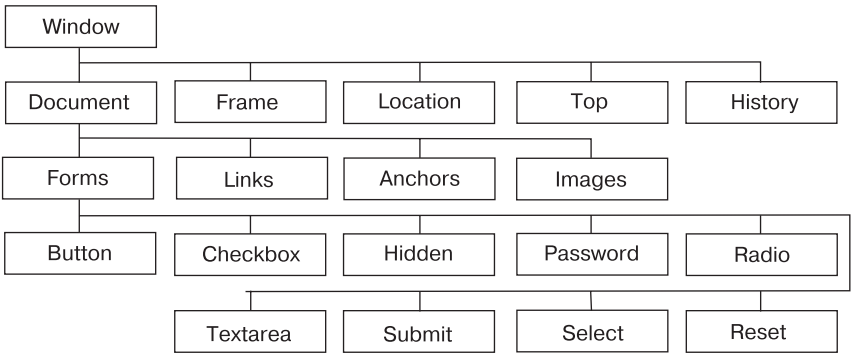


Рис. 2.17. Иерархия объектов в языке JavaScript

Родительским по отношению ко всем остальным объектам является объект `Window`, расположенный на самом верхнем уровне иерархии. Он представляет окно браузера и создается при запуске браузера. Свойства объекта `Window` относятся ко всему окну, в котором отображается документ.

Свойства объекта `Location` связаны с URL-адресом отображаемого в данный момент в окне браузера HTML-документа, а объекта `Frame` — со специальным способом представления данных в HTML-документе через фреймы. Свойства объекта `History` представляют адреса ранее загруженных веб-страниц.

Для каждой веб-страницы создается один объект `Document`. Он содержит в себе другие объекты — объекты HTML. Это различные элементы веб-страницы: формы, ссылки на другие HTML-документы или локальные ссылки внутри одного документа, объекты, определяющие URL-адрес документа, и т. д. Все эти объекты являются дочерними по отношению к объекту `Document`.

Если в HTML-документе имеются формы, то они также предстают в виде иерархического набора объектов. Сама форма соответствует объекту `Form`, который является дочерним по отношению к объекту `Document`. Объект `Form` может содержать в себе такие объекты, как кнопки, переключатели и поля для ввода текстовой информации. Все элементы формы являются дочерними объектами этой формы.

Иерархия объектов внутри веб-страницы задается вложенностью тегов HTML друг в друга и текста внутрь элементов. Объекты, имеющие открывающий и закрывающий теги (элементы-контейнеры), могут иметь дочерние объекты. Текст, атрибуты и теги, не имеющие закрывающего тега (например, `img`), не имеют дочерних объектов.

Чтобы лучше понять иерархию объектов в HTML-документе, рассмотрим простейший пример (листинг 2.25).

Листинг 2.25. Иерархия объектов в HTML-документе

```
<HTML>
  <HEAD>
    <TITLE>JavaScript Objects</TITLE>
  </HEAD>
  <BODY>
    <H1>Объекты JavaScript</H1>
    <P>Все элементы этой страницы являются объектами</P>
  </BODY>
</HTML>
```

Разберем этот документ с точки зрения иерархии объектов в языке JavaScript. Самому HTML-документу соответствует объект `Document`. Он является родителем для всех элементов, расположенных на веб-странице. Эти элементы принято называть *узлами*, а их иерархию — *деревом HTML-документа*.

Узел `<HTML>` является родительским по отношению ко всем остальным элементам веб-страницы. Узел `<HEAD>` имеет один дочерний узел `<TITLE>`. Узел `<TITLE>`, в свою очередь, имеет свой дочерний узел — текст «JavaScript Objects». Узел `<BODY>` имеет два дочерних узла: `<H1>` и `<P>`. Текст «Объекты JavaScript» является дочерним объектом по отношению к узлу `<H1>`, а текст «Все элементы этой страницы являются объектами» — по отношению к узлу `<P>`.

Таким образом, с точки зрения сценария JavaScript браузер и HTML-документ представляются иерархически организованным набором объектов. Свойства и методы этих объектов позволяют выполнять различные операции над окном браузера, HTML-документом, загруженным в это окно, а также над отдельными элементами на веб-странице.

Объекты веб-браузера

Обращаясь к объектам браузера, сценарий JavaScript взаимодействует с пользователем и документом HTML, загруженным в окно браузера, а также с самим браузером. Рассмотрим эти объекты.

Объект Window

Объект `Window` представляет собой окно браузера и является родительским по отношению ко всем остальным объектам в языке JavaScript.

Объект Window обладает следующими свойствами, которые описывают размеры окна, расположенные в окне фреймы, имя окна и содержимое строки состояния окна браузера:

- defaultStatus — сообщение, отображаемое в строке состояния окна браузера по умолчанию;
- status — текущее сообщение, отображаемое в строке состояния окна браузера;
- frames — массив всех фреймов данного окна;
- length — количество фреймов в родительском окне;
- name — имя окна, указанное при его открытии методом open, а также в параметре TARGET оператора <A> или в параметре NAME оператора <FORM>;
- parent — синоним имени окна, относится к окну, содержащему набор фреймов;
- self и window — синонимы имени окна, относятся к текущему окну;
- top — синоним имени окна, относится к окну верхнего уровня.

Свойства frames, length, parent и top применяются в том случае, когда в окно загружен документ HTML с фреймами. Свойство length позволит вам определить количество фреймов в окне, а свойство frames (оно является массивом) — получить доступ к окнам этих фреймов. О том, как использовать фреймы, в этой книге не рассказывается, поскольку в настоящее время они применяются крайне редко.

Рассмотрим использование свойств объекта Window на примере сценария из листинга 2.26.

Листинг 2.26. Свойство status объекта Window

```
<HTML>
<HEAD>
<TITLE>Свойство status объекта Window</TITLE>
</HEAD>
<BODY>
<SCRIPT language="JavaScript">
window.status="ЗДЕСЬ БУДЕТ ВАШ ТЕКСТ!"
</SCRIPT>
<P>Обратите внимание на текст в строке состояния вашего браузера!</P>
</BODY>
</HTML>
```

Сценарий позволяет изменить текст в строке состояния браузера (рис. 2.18).

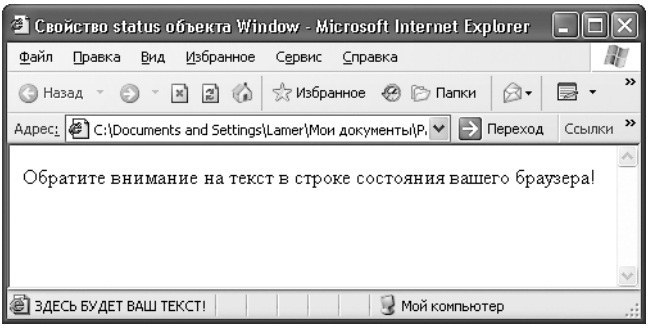


Рис. 2.18. Изменение текста в строке состояния браузера

Среди методов, определенных для объекта `Window`, отметим методы, позволяющие открывать и закрывать окна, отображать на экране диалоговые панели с сообщениями, а также методы для установки таймера:

- `alert` — отображает диалоговую панель Alert (Предупреждение) с сообщением и кнопкой OK;
- `open` — открывает окно;
- `close` — закрывает окно;
- `confirm` — отображает диалоговую панель Confirm (Подтверждение) с кнопками OK и Отмена;
- `prompt` — отображает диалоговую панель Prompt (Запрос) с полем ввода;
- `setTimeout` — устанавливает таймер;
- `clearTimeout` — сбрасывает таймер.

Методы `alert`, `prompt` и `confirm` мы подробно рассмотрели ранее. При их вызове не указывается объект, для которого вызывается метод, — объект `Window`. В этом случае интерпретатор сценария, встроенный в браузер, предполагает, что метод относится к объекту `Window`. Тем не менее вы можете явно указать объект `Window`: `window.alert()`. Результат будет аналогичным.

Метод `close` закрывает текущее окно. Его часто сочетают с методом `confirm` (листинг 2.27).

Листинг 2.27. Использование метода close

```
<HTML>
<HEAD>
```

```
<TITLE>Метод close</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function okno()
{
 if (confirm("Вы действительно желаете закрыть окно браузера?") )
 {
  window.close();
 }
</SCRIPT>
</HEAD>
<BODY onLoad="okno()">
</BODY>
</HTML>
```

В листинге 2.27 в области заголовка документа определяется функция `okno()`. Эта функция содержит условие и обращается к методу `confirm`, который выводит на экран диалоговую панель с запросом на закрытие окна.

Если вы нажмете кнопку **OK**, то метод возвратит значение `true`, а если кнопку **Отмена** — значение `false`.

Если ответ будет положительным, то функция `okno()` для текущего объекта `Window` вызовет метод `close`. В данном случае таким объектом является окно браузера. Метод `close` позволяет закрыть текущее окно браузера.

Метод `open` в сценарии JavaScript позволяет открыть новое окно браузера и загрузить в это окно документ HTML или изображение для просмотра.

Синтаксис команды с использованием метода `open` следующий: `open('адрес URL', 'имя_окна', config='параметр 1, параметр 2, параметр 3,…, параметр n')`.

Первый параметр метода `open` задает URL-адрес документа HTML, предназначенный для загрузки в новое окно, например `data.html`. Если загружаемая HTML-страница располагается на другом сервере, то следует добавить выражение **http://** и полный адрес страницы в Интернете, например `http://www.mysite.ru/download/javascript.html`.

Второй параметр в синтаксисе метода `open` определяет имя окна для использования в атрибуте `TARGET` оператора `<A>` или в теге `<FORM>`. Вы также можете указать

его как пустую строку вида "". Если необходимо, чтобы страница загружалась в маленьком окне, то данный атрибут должен иметь значение joe. Это значение следует также указать после команды TARGET.

Параметр config показывает, что следующие атрибуты относятся к конфигурации нового окна и определяют его внешний вид:

- width — задает ширину окна (в пикселах);
- height — задает высоту окна (в пикселах);
- toolbar — если атрибут имеет значение yes или 1, то окно снабжается стандартной панелью инструментов; если же значение равно no, то панель инструментов в новом окне не отображается;
- location — определяет, будет ли отображаться поле ввода адреса HTML-документа (адресная строка);
- status — отображает строки состояния;
- menubar — отображает строки меню;
- scrollbars — отображает полосы прокрутки;
- resizable — если этот атрибут имеет значение yes или 1, то вы можете изменять размер созданного окна.

Все атрибуты параметра config перечисляются через запятую без пробелов и заключаются в одинарные кавычки, поскольку они являются подкомандами.

Теперь рассмотрим пример сценария, в котором используется метод open (листинг 2.28).

Листинг 2.28. Использование метода open

```
<HTML>
<HEAD>
<TITLE>Метод open</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
window.open('okno2.html', 'joe', config='height=250,width=300,toolbar=no,
menubar=no,scrollbars=no, resizable=yes,location=no,status=yes')
</SCRIPT>
</BODY>
</HTML>
```

Сценарий в файле `okno.html` открывает документ `okno2.html` в новом маленьком окне браузера. Это окно имеет размеры 300 на 250 пикселов. В нем не отображаются меню, панель инструментов, адресная строка и полоса прокрутки — за это отвечают значения атрибутов параметра `config`, равные `no`. В новом окне отображается только строка состояния, что определяется атрибутом `status=yes`. Кроме того, пользователь может изменять размеры нового окна с помощью атрибута `resizable=yes`.



СОВЕТ

Открывайте новые окна только при необходимости. В большинстве случаев браузеры блокируют окна, которые появляются сами по себе, без запроса пользователя.

Рассмотрим следующий метод объекта `Window` — `setTimeout`. С его помощью вы можете запрограммировать компьютер на выполнение определенных команд по истечении некоторого времени. Этот метод применяется со следующим синтаксисом: `setTimeout("cmd", timeout)`. Когда пройдет время, заданное параметром `timeout` (в миллисекундах), запустится команда JavaScript, заданная параметром `cmd`.

Пример сценария, содержащего метод `setTimeout`, приводится в листинге 2.29.

Листинг 2.29. Использование метода setTimeout

```
<HTML>
<HEAD>
<TITLE>Метод setTimeout</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function timer()
{
 idTimer=window.setTimeout("alert('Время истекло!')", 3000);
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
 <INPUT TYPE="button" value="Timer" onClick="timer()">
</FORM>
</BODY>
</HTML>
```

Листинг 2.29 содержит сценарий и форму с кнопкой Timer, при нажатии на которую открывается окно Alert с задержкой в 3 с (рис. 2.19).

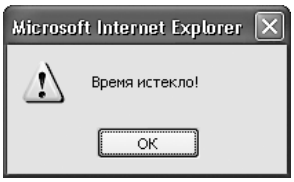


Рис. 2.19. Окно Alert с сообщением

Здесь `setTimeout()` — это метод объекта `Window`. Он определяет интервал времени в 3000 мс (3 с) и команду, которая должна выполниться по истечении этого времени. В данном случае это вызов окна с сообщением при помощи метода `alert`: `"alert('Время истекло!')"`.

Метод `clearTimeout` позволяет остановить таймер, запущенный методом `setTimeout`. Параметром метода `clearTimeout` является идентификатор таймера, полученный от метода `setTimeout`, — `clearTimeout(idTimer)`.

Методы `setTimeout` и `clearTimeout` используются для периодического вызова функции сдвига сообщения в строке состояния браузера («бегущая» строка) и для ограничения времени ввода пользователем пароля в формах. Этот метод используется также при создании анимационных эффектов с помощью сценариев JavaScript.

С объектом `Window`, помимо свойств и методов, связаны два обработчика событий — `onLoad` и `onUnload`. Первый из них срабатывает, когда браузер заканчивает загрузку окна, а второй — когда пользователь завершает работу с документом HTML.

Вы можете предусмотреть в своем сценарии обработку этих событий, назначив определенную функцию для каждого из них. Функция, которая вызывается по завершении загрузки документа, может выполнять некоторые действия, создавать дополнительные окна или выводить сообщения. Обработчик события `onUnload` может освобождать полученные ресурсы или выводить какие-либо дополнительные сообщения при выходе пользователя с веб-страницы.

ВНИМАНИЕ

Как правило, вызов обработчиков событий onLoad и onUnload находится в теге ‹BODY›.

В качестве примера рассмотрим документ HTML, содержание которого приводится в листинге 2.30.

Листинг 2.30. Использование обработчиков событий onLoad и onUnload

```
<HTML>
<HEAD>
<TITLE>Обработчики событий onLoad и onUnload</TITLE>
```

```
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
function hello()
{
  alert("Добро пожаловать на мою домашнюю страничку!")
}
function bye()
{
  alert("До свидания! Спасибо, что зашли!")
}
</SCRIPT>
<BODY BGCOLOR=white onLoad="hello()" onUnload="bye()">
</BODY>
</HTML>
```

Здесь в операторе <BODY> мы определили обработчики событий onLoad и onUnload. При возникновении первого события будет вызываться функция hello, а при возникновении второго — функция bye.

Объект Navigator

Теперь рассмотрим объект JavaScript, с помощью которого можно получить некоторые данные о браузере и компьютере пользователя, — объект Navigator. Эта информация может пригодиться при формировании стилей веб-страниц для различных браузеров. Данный объект создается автоматически при открытии веб-страницы браузером и содержит общую информацию о браузере пользователя.

Как и другие объекты JavaScript, объект Navigator имеет свои свойства — конкретные строковые значения о браузере пользователя. Рассмотрим их кратко.

- appCodeName — определяет кодовое имя (платформу, «движок») браузера.
- appName — определяет имя (название) браузера, например Netscape или Microsoft Internet Explorer.
- appVersion — определяет версию браузера, включая платформу, выпуск браузера и операционную систему пользователя.
- userAgent — возвращает строку, идентифицирующую браузер пользователя, то есть заголовок протокола, используемого браузером во время работы с сервером. Это комбинация значений свойств appCodeName и appVersion.

- `cookieEnabled` — определяет, разрешается или нет использование текста cookies, позволяющих сохранить небольшие фрагменты информации на компьютере пользователя. Возвращает `true`, если пользователь разрешил браузеру прием cookie.
- `browserLanguage` — определяет текущий язык браузера.
- `systemLanguage` — определяет язык операционной системы по умолчанию, например ru, если используется русская версия Windows.
- `userLanguage` — определяет пользовательские настройки языка операционной системы.
- `platform` — определяет платформу операционной системы пользователя, например Win32.
- `cpuClass` — задает класс (тип) центрального процессора компьютера пользователя, например x86 или Alpha.

Рассмотрим пример сценария, в котором определяются различные характеристики браузера (листинг 2.31).

Листинг 2.31. Определение характеристик браузера

```
<HTML>
<HEAD>
<TITLE>Свойства объекта Navigator</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=JavaScript>
var an = navigator.appName;
var av = navigator.appVersion;
var acn = navigator.appCodeName;
var ua = navigator.userAgent;
var cook = navigator.cookieEnabled;
var blang = navigator.browserLanguage;
var slang = navigator.systemLanguage;
var ulang = navigator.userLanguage;
var platf = navigator.platform;
var cpu = navigator.cpuClass;
document.write("<b>Ваш браузер</b> "+an+" <b>версии</b> "+av+"<br><b>Его
кодовое имя</b> "+acn+" <b>и заголовок протокола</b> "+ua+"<br><b>Текущее
```

```
значение cookies </b>"+cook+"<br><b>Текущий язык браузера </b>"+blang+
"<br><b>Язык ОС by default </b>"+slang+"<br><b>Пользовательские настройки
языка ОС </b>"+ulang+ "<br><b>Платформа ОС </b>"+platf+"<br><b>Тип процес-
сора вашего компьютера </b>"+cpu)
</SCRIPT>
</BODY>
</HTML>
```

В данном примере каждому свойству объекта `Navigator` назначается отдельная переменная. С помощью команды `document.write` информация о характеристиках браузера отображается на веб-странице (рис. 2.20).

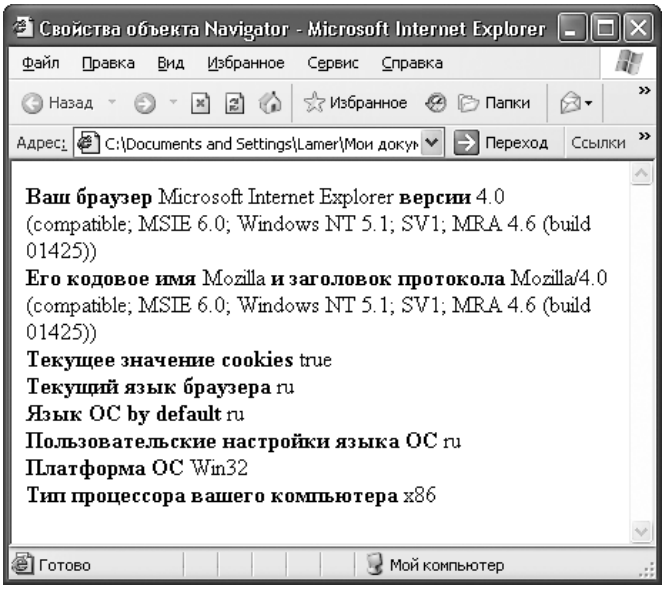


Рис. 2.20. Характеристики браузера

Объект Screen

Объект `Screen` содержит информацию о различных параметрах экрана монитора пользователя: разрешающая способность, глубина цвета, частота обновления экрана и т. п. Эти данные могут помочь веб-разработчику подстроить интерфейс сайта под конкретные параметры экрана, если это необходимо.

Объект `Screen` имеет следующие свойства:

- `width` — определяет полную ширину экрана монитора (в пикселах);
- `height` — определяет полную высоту экрана монитора (в пикселах);

- `availHeight` — определяет высоту полезной области экрана монитора без Панели задач и подобных ей элементов графического интерфейса операционной системы;
- `availWidth` — определяет ширину полезной области экрана монитора без Панели задач и подобных ей элементов графического интерфейса операционной системы;
- `colorDepth` — возвращает глубину цвета; для 16 цветов возвращается 2, для 256 — 8, для 16,7 млн (режим High Color) — 32;
- `updateInterval` — возвращает интервал времени (в миллисекундах) между обновлениями экрана. Значение 0 позволяет браузеру выбрать среднее число, так как обычно работать с ним лучше.

В следующем примере приводится сценарий, который позволит вам определить параметры экрана монитора пользователя (листинг 2.32).

Листинг 2.32. Свойства объекта Screen

```
<HTML>
<HEAD>
<TITLE>Свойства объекта Screen</TITLE>
</HEAD>
<BODY>
 <SCRIPT LANGUAGE=JavaScript>
 document.write('width: '+window.screen.width+'<br>');
 document.write('availWidth: '+window.screen.availWidth+'<br>');
 document.write('height: '+window.screen.height+'<br>');
 document.write('availHeight: '+window.screen.availHeight+'<br>');
 document.write('colorDepth: '+window.screen.colorDepth+'<br>');
 document.write('updateInterval: '+window.screen.updateInterval)
 </SCRIPT>
</BODY>
</HTML>
```

Результат работы сценария показывается на рис. 2.21.

Особенно важны для веб-разработчика размеры экрана монитора пользователя. Используя свойства объекта `Screen`, в сценарии JavaScript можно адаптировать дизайн веб-страниц к разрешению экрана монитора пользователя.

Объект History

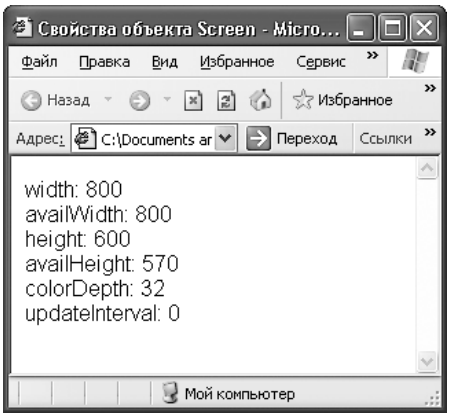


Рис. 2.21. Параметры экрана монитора пользователя

Объект `History` является частью объекта `Window`. Он содержит информацию о посещенных пользователем веб-страницах за текущий сеанс путешествия по Интернету. Эти URL-адреса можно загружать снова, таким образом передвигаясь по истории посещений. Основной целью объекта `History` является доступ к списку URL-адресов, введенных в браузере пользователем.

Объект `History` имеет всего одно свойство — `length`. Оно определяет количество посещенных пользователем веб-страниц за текущий сеанс работы в браузере. Сценарий из листинга 2.33 показывает, сколько страниц пользователь посетил за сеанс.

Листинг 2.33. Определение количества посещенных пользователем веб-страниц

```
<HTML>
<HEAD>
<TITLE>Количество посещенных страниц</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=JavaScript>
function hislen()
{
 alert(window.history.length)
}
</SCRIPT>
<INPUT type="button" onclick="hislen()" value="Показать количество посещенных страниц">
</BODY>
</HTML>
```

Данный сценарий содержит функцию `hislen()`, вызывающую окно Alert при нажатии кнопки формы. В окне отображается количество посещенных веб-страниц за текущий сеанс работы с браузером. Если вы запускаете сценарий с диска своего

компьютера, не удивляйтесь, если количество посещенных страниц будет равняться нулю (рис. 2.22).



Рис. 2.22. Количество посещенных пользователем веб-страниц

Передвигаться по истории и загружать ранее посещенные пользователем веб-страницы позволяют следующие методы объекта `History`:

- `go` — загружает веб-страницу с определенным номером относительно страницы, открытой в браузере в данный момент (текущая страница имеет индекс 0, предыдущая — «– 1», страница, посещенная до этого, — «– 2» и т. д.);
- `back` — загружает предыдущую веб-страницу, посещенную пользователем (эквивалентно `go(-1)`);
- `forward` — загружает следующую веб-страницу из списка истории, если таковая имеется (эквивалентно `go(1)`).

Например, чтобы перейти на две страницы назад, можно использовать сценарий из листинга 2.34.

Листинг 2.34. Перемещение по истории посещений

```
<HTML>
<HEAD>
<TITLE>Перемещение по истории посещений</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=JavaScript>
function goback()
{
  window.history.go(-2)
}
</SCRIPT>
<INPUT type="button" onclick="goback()"  value="Перейти на 2 страницы назад">
</BODY>
</HTML>
```

Если нажать кнопку формы (рис. 2.23), запустится функция `goback()`, позволяющая перейти на две страницы назад.

Используя методы объекта `History`, вы сможете организовать удобную систему навигации по сайту. Учтите, что для этого пользователь должен просматривать все страницы вашего сайта в одном окне браузера. Как только какая-либо страница будет открыта в новом окне, история посещений применительно к этому окну браузера начнется с нуля.

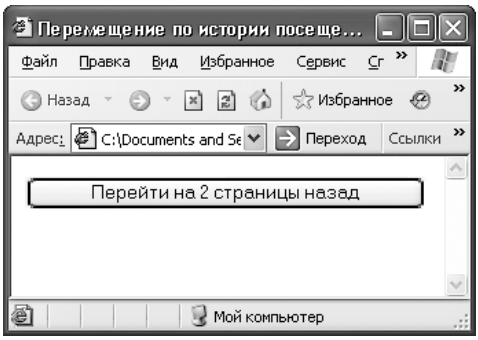


Рис. 2.23. Кнопка для запуска функции

Объект Location

Этот объект содержит информацию об URL-адресе текущей страницы и его составляющих.

Объект `Location` в сценарии JavaScript позволяет обновить текущую страницу или полностью поменять адрес URL, то есть перейти на другую веб-страницу.

Свойства объекта `Location` содержат различную информацию о подключении пользователя к HTTP-серверу:

- `href` — полный URL-адрес текущей веб-страницы;
- `hash` — имя «якоря» в URL-адресе веб-страницы, то есть значение атрибута `name` (если он есть);
- `host` — часть URL-адреса страницы, содержащая имя сервера в Интернете и номер порта;
- `hostname` — имя сервера в Интернете, с которого загружена текущая веб-страница;
- `pathname` — путь к файлу на веб-сервере без имени сервера и порта;
- `port` — сообщает номер порта HTTP-сервера, через который происходит обращение к веб-странице (если он не указывается, возвращает номер 80 — стандартный порт, через который работает протокол HTTP);
- `protocol` — протокол передачи данных, например HTTP, FTP и др. (если протокол не указывается, возвращается значение `http:`);
- `search` — строка параметров для серверных сценариев (начинается со знака `"?"`).

Рассмотрим использование свойства `href` объекта `Location` на примере сценария из листинга 2.35.

Листинг 2.35. Свойство href объекта Location

```
<HTML>
<HEAD>
<TITLE>URL текущего HTML-документа</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE=JavaScript>
 document.write(location.href);
</SCRIPT>
</BODY>
</HTML>
```

Данный сценарий определяет полный URL-адрес текущего HTML-документа. При запуске сценария с жесткого диска компьютера команда `location.href` покажет полный путь к файлу со сценарием (рис. 2.24).

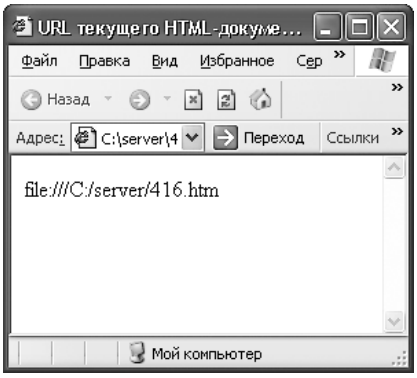


Рис. 2.24. Путь к файлу со сценарием

Методы объекта `Location` позволяют перезагружать текущую веб-страницу или менять адрес URL, а также загружать другую веб-страницу вместо предыдущей. При этом переход к предыдущей странице невозможен с помощью кнопки Назад в браузере пользователя. Возникает эффект, что страницы как бы подменяются друг другом.

Методы объекта `Location` следующие:

- `assign` — загружает другую страницу, меняя URL-адрес текущей веб-страницы на адрес, указанный в параметре метода;
- `reload` — обновляет текущую веб-страницу (срабатывает не всегда правильно, поскольку некоторые браузеры все равно берут эту страницу из кэша, не обращаясь к серверу);
- `replace` — замена текущей веб-страницы на страницу, URL-адрес которой указывается в параметре метода; при этом в списке истории браузера адрес предыдущего документа HTML заменяется адресом нового.

В листинге 2.36 приводится пример сценария, содержащего один из методов объекта `Location` — метод `replace`.

Листинг 2.36. Использование метода replace

```
<HTML>
<HEAD>
<TITLE>Метод replace</TITLE>
<SCRIPT LANGUAGE=JavaScript>
function replaceDoc()
  {
   window.location.replace("http://www.yandex.ru")
  }
</SCRIPT>
</HEAD>
<BODY>
<input type="button" value="Replace" onclick="replaceDoc()" />
</BODY>
</HTML>
```

Сценарий позволяет заменить текущую страницу в окне браузера на главную страницу сайта www.yandex.ru. За это отвечает функция `replaceDoc`, которая вызывается нажатием кнопки Replace.

Таким образом, объект `Location` позволяет не только определить адрес загруженного в браузер HTML-документа, но и, что более важно, загрузить в текущее окно браузера новую веб-страницу.

Итак, мы рассмотрели объекты браузера, их свойства и методы. Для HTML-документа и его содержимого в языке JavaScript также имеются свои объекты.

Объектная модель документа

Стандартный набор объектов в документе HTML, их методы, свойства и способы доступа к ним определяются так называемой *Объектной моделью документа* (Document Object Model, сокращенно DOM). DOM позволяет управлять всеми элементами веб-страницы, изменять их свойства и содержание, а также создавать новые элементы.

Объект Document

Для работы с документами HTML в языке JavaScript предназначен отдельный объект — `Document`. Пользуясь его свойствами и методами, сценарий JavaScript

может получить информацию о текущем документе, загруженном в окно браузера, а также управлять отображением содержимого этого документа.

Любая веб-страница, которую вы просматриваете в окне браузера, может описываться как набор объектов. Она включает собственно HTML-документ «в целом» и более мелкие объекты — элементы веб-страницы, например заголовок, абзацы, рисунок, ссылку, форму и т. п.

Элементы веб-страницы называются *HTML-объектами*, поскольку они определяются тегами языка HTML. Все HTML-объекты являются дочерними по отношению к объекту `Document`, благодаря чему вы можете обращаться к ним из сценариев JavaScript.

Объект `Document` является главным в модели DOM. Он включает в себя множество подчиненных объектов и коллекций.

Коллекция — это массив объектов, проиндексированный не только по числовым номерам элементов, но и по их именам, обладающий свойствами и методами (именно этим коллекция и отличается от обычного массива). Поэтому каждая коллекция является одновременно и объектом, и свойством объекта `Document`.

Для объекта `Document` существуют следующие коллекции:

- `anchors` — коллекция всех локальных меток («якорей»), размещенных в HTML-документе; эти метки содержатся в тегах `<a name="..."> </a>` и применяются для организации ссылок внутри одной веб-страницы;
- `links` — коллекция всех ссылок в HTML-документе, содержащихся в тегах `<a href="..."> </a>`;
- `images` — массив всех изображений на веб-странице;
- `forms` — коллекция всех форм в документе HTML.

Благодаря этим коллекциям сценарию JavaScript доступны все локальные метки, формы, изображения и ссылки в HTML-документе как элементы соответствующих массивов.

Пример сценария, в котором используется одна из коллекций объекта `Document`, приводится в листинге 2.37.

Листинг 2.37. Использование коллекции anchors

```
<HTML>
<HEAD>
<TITLE>Anchors</TITLE>
```

```
</HEAD>
<BODY>
<a name="first">1 anchor</a><br />
<a name="second">2 anchor</a><br />
<a name="third">3 anchor</a><br />
<br />Количество «якорей» в этом HTML-документе:
<SCRIPT LANGUAGE=JavaScript>
 document.write(document.anchors.length)
</SCRIPT>
</BODY>
</HTML>
```

Массив `anchors` в данном сценарии позволяет определить количество «якорей» в HTML-документе. Команда `document.write` отображает количество «якорей» на странице в числовом выражении (рис. 2.25).

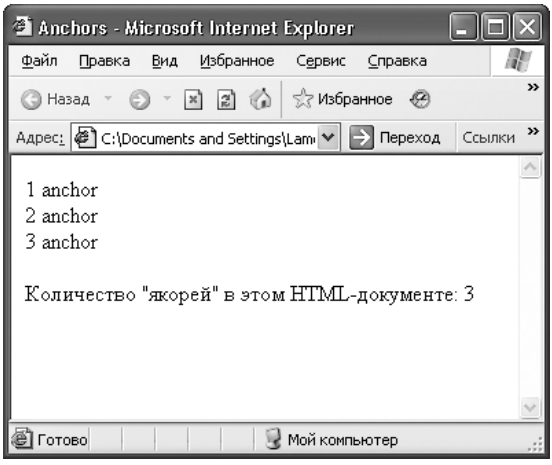


Рис. 2.25. Использование коллекции anchors

Количество «якорей» подсчитывается при помощи команды `document.anchors.length`, где `length` — это функция для измерения длины массива, которую мы рассмотрели ранее.

Аналогичным образом можно определить количество ссылок, изображений и форм на веб-странице. Для этого вместо коллекции `anchors` в строку `document.anchors.length` достаточно подставить названия массивов: `links`, `images` или `forms`.

Помимо коллекции, для объекта Document существуют свои свойства, анализируя которые сценарий JavaScript может определить значения различных параметров веб-страницы. Перечислю эти свойства объекта Document.

- alinkColor — содержимое атрибута ALINK. Это свойство определяет цвет ссылок, выбранных пользователем.
- linkColor — содержимое атрибута LINK. Это свойство определяет цвет ссылок, размещенных в документе HTML, которые пользователь пока не посетил.
- vlinkColor — содержимое атрибута VLINK. Это свойство определяет цвет ссылок, размещенных в документе HTML, которые пользователь уже посетил ранее.
- bgColor — содержимое атрибута BGCOLOR. Это свойство применяется для создания цветного фона HTML-документа. Цвет задается либо в шестнадцатеричном виде (например, #F0F8FF), либо с использованием названий цветов (например, red или white).
- fgColor — содержимое атрибута TEXT, определяющего цвет текста. Задается таким же образом, что и фон веб-страницы (с помощью атрибута BGCOLOR).
- lastModified — дата последнего изменения документа HTML.
- location — полный URL-адрес текущей веб-страницы.
- referrer — полный URL-адрес страницы, с которой была открыта данная веб-страница.
- title — заголовок документа, заданный с помощью оператора <TITLE>.
- URL — полный URL-адрес документа HTML.

Многие из перечисленных свойств объекта Document позволяют изменять значения HTML-элементов, расположенных в блоке <BODY>, например фон страницы, цвет ссылок, а также содержание заголовка документа.

Большинство свойств объекта Document доступно сценарию JavaScript как для чтения, так и для записи. Следующий пример демонстрирует, как с помощью сценария JavaScript можно изменить свойства документа HTML: фон и цвет текста, а также цвета посещенных, не посещенных и выбранных пользователем ссылок (листинг 2.38).

Листинг 2.38. Свойства объекта Document

```
<HTML>
<HEAD>
<TITLE> Свойства объекта Document</TITLE>
```

```
<SCRIPT LANGUAGE="JavaScript">
  document.bgColor = "#00FF80";
  document.fgColor = "#800080";
  document.linkColor  = "#000000";
  document.alinkColor = "#FF0000";
  document.vlinkColor = "#4000FF";
</SCRIPT>
</HEAD>
<BODY BGCOLOR=white>
<H1>Изменение цветового оформления страницы</H1>
<H3>Щелкните по этим ссылкам: </H3>
<P><A HREF="http://www.yandex.ru">Yandex</A>
<P><A HREF="http://www.mail.ru">Бесплатная электронная почта</A>
<P><A HREF="http://www.microsoft.com">Сервер Microsoft</A>
</BODY>
</HTML>
```

Результат действия сценария вы видите на рис. 2.26.

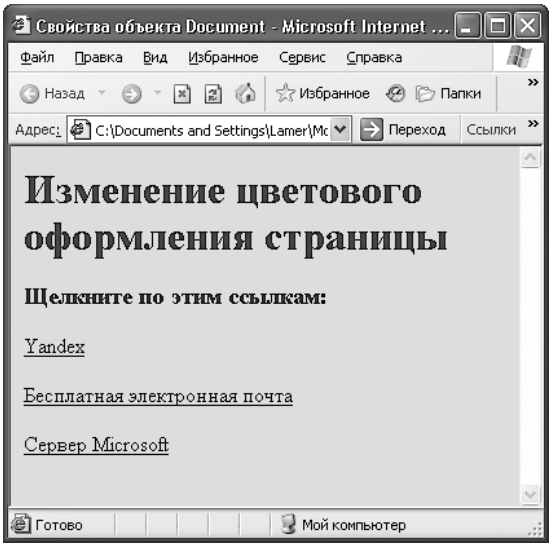


Рис. 2.26. Изменение фона, цвета текста и ссылок

Обратите внимание, что данный сценарий переопределяет фон веб-страницы, заданный параметром `BGCOLOR=white` в теге `<BODY>`.

Рассмотрим еще один пример сценария, в котором используется свойство `lastModified` объекта `Document` (листинг 2.39).

Листинг 2.39. Использование свойства lastModified

```
<HTML>
<HEAD>
<TITLE> Свойство lastModified</TITLE>
</HEAD>
<BODY>
<center>
<SCRIPT LANGUAGE="JavaScript">
document.write("Последний раз страница редактировалась:<br>" +document.
lastModified)
</SCRIPT>
</center>
</BODY>
</HTML>
```

Свойство `lastModified` позволяет узнать дату внесения последних изменений в содержание HTML-документа. В сценарии это реализуется командой `document.lastModified`. Строка `document.write()` помещает полученную дату на веб-страницу (рис. 2.27).

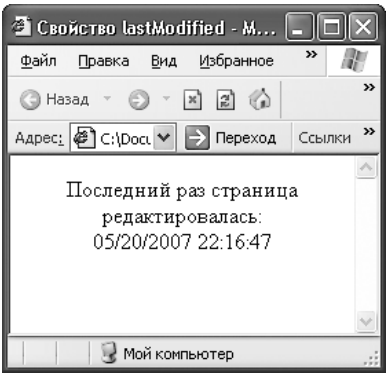


Рис. 2.27. Использование свойства lastModified

Во многих рассмотренных ранее примерах сценариев JavaScript не раз использовалась строка `document.write`. Кроме этого широко используемого метода, сценарии JavaScript могут вызывать и другие методы, определенные для объекта `Document`. Рассмотрим их кратко.

- `getElementsByName({Имя_элемента})` — возвращает элемент, специфическое имя которого передается в качестве параметра.
- `getElementById({ID})` — возвращает элемент, имя которого передается в качестве параметра. Имя элемента страницы задается атрибутом `ID`. Единственный параметр этого метода — имя элемента страницы.

- `getElementsByTagName({Имя тега})` — возвращает тег, имя которого передается в качестве параметра.
- `write` — записывает текст или HTML-код в текущее место документа.

Метод `getElementsByTagName` используется в сценариях JavaScript особенно часто. Рассмотрим пример такого сценария (листинг 2.40).

Листинг 2.40. Использование метода getElementsByTagName

```
<HTML>
<HEAD>
<TITLE> Метод getElementsByTagName</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function getElements()
  {
  var x=document.getElementsByTagName("input");
  alert(x.length);
  }
</SCRIPT>
</HEAD>
<BODY>
<input name="myInput" type="text" size="30" /><br />
<input name="myInput" type="text" size="30" /><br />
<input name="myInput" type="text" size="30" /><br />
<br />
<input type="button" onclick="getElements()" value="Посчитаем элементы
input" />
</BODY>
</HTML>
```

Данный сценарий содержит функцию `getElements()`, которая подсчитывает все элементы типа `input` на веб-странице, а затем выводит результат в окне Alert (рис. 2.28).



Рис. 2.28. Окно с количеством элементов

Функция срабатывает при нажатии кнопки в форме. Как вы видите, в этом HTML-документе содержится четыре элемента, обозначенных тегом `input`: три поля для ввода и кнопка (рис. 2.29).

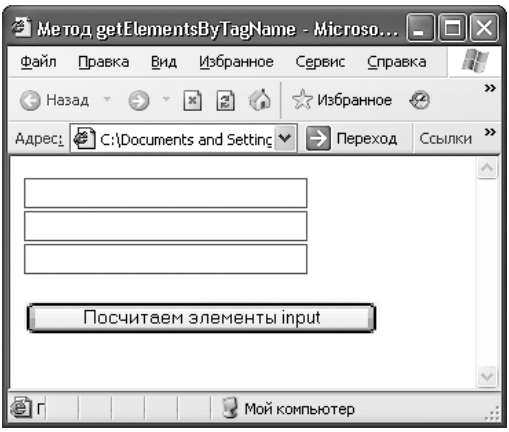


Рис. 2.29. Элементы input

Благодаря определенным свойствам и методам через объект `Document` можно получить доступ к любым другим объектам, расположенным на веб-странице.

Доступ к объектам документа

Сценарии JavaScript можно применять для создания динамических веб-страниц, способных получать от пользователя произвольную информацию и обрабатывать ее. Для этого необходимо организовать доступ сценария к определенным элементам веб-страницы.

Доступ к различным объектам HTML-документа в языке JavaScript организовывается в строгом соответствии с иерархией объектов. Каждый объект иерархической структуры имеет имя.

Например, на веб-странице может находиться несколько изображений с именами `img1`, `img2` и `img3`. Если вы хотите обратиться к первому рисунку, то должны сориентироваться в иерархии объектов и начать с ее вершины. Главным объектом на веб-странице является объект `Document`. Все изображения на странице представляются как коллекция изображений `images`. Причем первый рисунок всегда обозначается `images[0]`, второй — `images[1]`, третий — `images[2]` и т. д. Как вы заметили, отсчет объектов в коллекции начинается не с единицы, а с нуля. Таким образом, вы можете получить доступ к первому изображению с помощью объекта `document.images[0]`.

К ссылкам и формам применяется такой же принцип. Если вы хотите получить доступ к какому-либо элементу формы, необходимо начать отсчет снова с самой вершины иерархии объектов.

Например, чтобы узнать, какой текст ввел посетитель в поле формы вашей веб-страницы, следует обратиться к значению `value` данного поля. Для этого в сценарии JavaScript можно записать следующую строку:

```
Name = document.forms[0].elements[0].value
```

Полученная строка со значением первого элемента первой формы заносится в переменную `name`. Теперь вы можете работать с этой переменной, используя ее в других строках сценария JavaScript.

Однако процедура адресации к различным объектам по номеру весьма затруднительная. Например, довольно неудобно обращаться к объекту через строку `document.forms[4].elements[15]`, так как можно запутаться в количестве объектов на странице и неправильно указать номер нужного объекта.

Во избежание подобной проблемы в JavaScript можно не только пользоваться существующими коллекциями объекта `Document`, но и самим присваивать объектам уникальные имена. Например, для формы можно задать имя с помощью атрибута `name`: `<form name="myform">`. Эта запись означает, что первая форма на вашей веб-странице, соответствующая объекту `forms[0]`, получает уникальное имя `myform`.

Точно так же можно поступить по отношению к любому элементу формы. Вместо `elements[0]` вы можете указать атрибут `name`, вместо `name = document.forms[0].elements[0].value` — `name = document.myform.element_name.value`.

Такой прием значительно упрощает доступ к элементам HTML-документа, а следовательно, и программирование на JavaScript, особенно в случае с большими веб-страницами, содержащими множество объектов.

ВНИМАНИЕ

! При написании имен объектов внимательно следите за положением регистра. Нельзя писать Myform вместо myform, поскольку язык JavaScript чувствителен к регистру и будет воспринимать их как имена разных объектов.

Таким образом, доступ к объектам HTML-документа можно получить двумя способами:

- используя коллекции и нумерацию объектов;
- присваивая каждому объекту уникальное имя.

В любом из этих случаев необходимо определить сам объект и всех его предков. Объект высшего уровня всегда находится слева в выражении, и переход к дочерним

объектам происходит слева направо. По правилам языка JavaScript родительские и дочерние объекты отделяются друг от друга точками.

Рассмотрим несколько примеров, иллюстрирующих способы доступа к объектам веб-страницы.

В листинге 2.41 приводится текст сценария, при помощи которого данные, введенные пользователем в поле формы, передаются в функцию для дальнейшей обработки.

Листинг 2.41. Передача данных в функцию

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function doit()
{
   var greeting="Мне нравится "
   alert(greeting + document.aform.color.value
      + " " + document.aform.geometr.value)
   var prich="Потому, что "
   alert(prich + document.aform.prichina.value)
   alert("Количество букв в названии вашей фигуры"
     + document.aform.geometr.value.length)
}
</SCRIPT>
</HEAD>
<BODY>
<center>
<FORM NAME="aform">
  Ваша любимая геометрическая фигура:
      <INPUT TYPE="text" NAME="geometr"><p>
  Ваш любимый цвет:
      <INPUT TYPE="text" NAME="color"><p>
Почему вам нравится эта фигура:
    <INPUT TYPE="text" NAME="prichina"><p>
    <INPUT TYPE="button" VALUE="Ответить" onClick="doit()">
</FORM>
```

```
</center>
</BODY>
</HTML>
```

В данном HTML-документе имеется форма с полями ввода и кнопкой (рис. 2.30). При ее нажатии обработчик событий `onClick` запускает функцию `doit()`, обрабатывающую данные, которые пользователь ввел в поля формы.

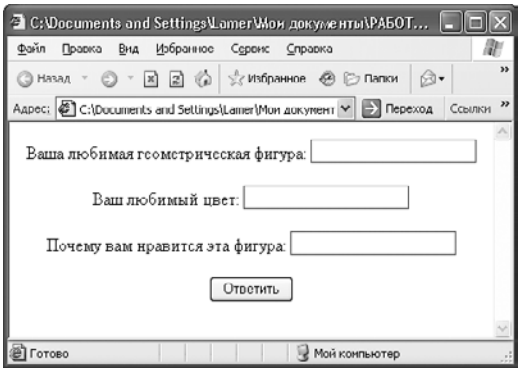


Рис. 2.30. Элементы формы

Доступ к объектам формы в данном сценарии организуется путем присвоения каждому объекту формы уникального имени и их перечисления в соответствии с их иерархией в документе.

Самой форме присвоено имя `aform`. Каждому полю ввода также заданы уникальные имена: `geometr`, `color` и `prichina`. Введенные пользователем данные передаются в функцию как значение `value`. В соответствии с иерархией объектов на первом месте стоит объект `Document`, далее идет объект `форма`, затем поле формы и, наконец, значение поля — текст, введенный пользователем.

В сценарии такая цепочка объектов выглядит следующим образом: `document.aform.geometr.value`. Таким образом, функция `doit()` получает данные пользователя из значения `value` поля `geometr`, которое находится в форме `aform` HTML-документа `document`. Аналогично данные передаются в функцию из других полей формы.

Функция отображает полученные от формы данные в окне Alert (рис. 2.31).

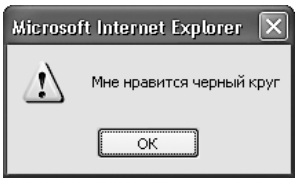


Рис. 2.31. Результат работы сценария

Итак, мы рассмотрели объекты HTML-документа: саму веб-страницу и расположенные на ней элементы. Все они обладают некоторыми свойствами. К каждому из них можно применить определенные действия, или методы, которые позволяют управлять содержимым веб-страницы и динамически его изменять.

Объекты, свойства и методы являются фундаментальными понятиями языка JavaScript. Теперь вы сможете использовать в сценариях различные элементы окна

браузера и веб-страницы, изменять их свойства и управлять ими при помощи JavaScript.

Работа с формами

Сейчас я расскажу об одном из самых важных механизмов, с помощью которого можно придать веб-странице интерактивность, — формы. Этот механизм позволяет любому пользователю передавать данные на сервер. До появления JavaScript большинство разработчиков для передачи данных пользователю от веб-сервера после предоставления формы вызывали определенные серверные программы (рассмотрим их в главе 3). Пользователь вводил свои данные в форму и передавал ее серверу. Этот процесс занимал достаточно много времени, что было вызвано задержкой при обмене данными между сервером и браузером пользователя, так как сервер должен был обработать данные и переслать их обратно.

JavaScript расширил сферу применения форм: стали возможны предварительная обработка данных формы и их верификация. Подобная обработка приводит к снижению загрузки сервера и уменьшает время, затрачиваемое на передачу данных.

Конечно, JavaScript не заменяет полностью серверные программы обработки данных. Сценарии на клиентской стороне являются дополнением серверных программ.

Форма как объект

Объект `Form` служит контейнером для пересылки данных от пользователя к серверу. Все элементы управления формы находятся внутри тега `<form>` и являются дочерними по отношению к нему.

В спецификации HTML объект `Form` определяется следующим образом:

```
<form
  [name="ИмяФормы"]
  [accept="ТипСодержимого"]
  [accept-charset="НаборСимволов"]
  [action="URLСервера"]
  [enctype="ТипКодировки"]
  [metod=get|post]
  [target="ИмяОкна"]
```

```
  [id="Идентификатор"]
  [class="ИмяКласса"]
  [style="ИмяСтиля"]
  [title="Заголовок"]
  [lang="КодЯзыка"]
  [dir="Каталог"]
  [onclick="КодСценария"]
  [ondblclick="КодСценария"]
  [onmousedown="КодСценария"]
  [onmouseup="КодСценария"]
  [onmouseover="КодСценария"]
  [onmousemove="КодСценария"]
  [onkeypress="КодСценария"]
  [onkeydown="КодСценария"]
  [onkeyup="КодСценария"]
  [onsubmint="КодСценария"]
  [onreset="КодСценария"]
</form>
```

Объект `Document` является контейнером всех форм на странице. Обычно объект `Form` и элементы управления именованы (с помощью атрибута `Name`). Чтобы обратиться к отдельному элементу в конкретной форме, необходимо воспользоваться конструкцией `document.ИмяФормы.ИмяЭлемента`. Например, для доступа к текстовому полю `myelement` в форме `myform` можно воспользоваться следующим кодом:

```
document.myform.myelement.value="Мое текстовое поле";
```

Результат обработки этого кода вы видите на рис. 2.32.

Кроме того, объект `Document` содержит свойство `forms`, которое можно использовать для ссылки на форму. В свою очередь, объект `Form` имеет свойство `elements`, позволяющее обращаться по индексу к определенному элементу. Следовательно, такого же эффекта можно достичь, воспользовавшись кодом

```
document.forms[0].element[0].value="Мое текстовое поле";
```

При этом должно соблюдаться одно условие: форма `myform` является первой формой на странице (`forms[0]`), а элемент управления `myelement` — первым элементом в этой форме (`element[0]`).

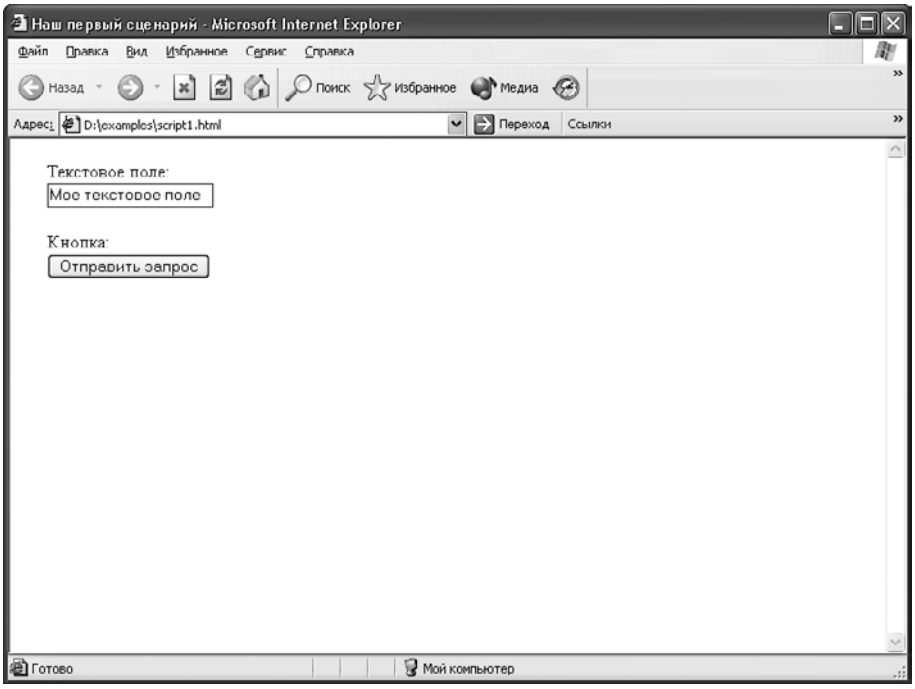


Рис. 2.32. Доступ к элементу

Форма и ее элементы управления

Формы состоят из элементов управления и разметки, необходимой для создания структуры формы и представления ее элементов на экране. В число элементов формы входят поля ввода данных, флажки, переключатели, списки, кнопки и т. д.

Рассмотрим пример использования формы. Предположим, требуется проверить, что все ее обязательные поля заполнены. Свойство `elements` позволяет организовать цикл для проверки заполнения полей. В листинге 2.42 приводится код, который содержит основные элементы управления и демонстрирует упрощенную проверку полей.

Листинг 2.42. Пример использования формы

```
<html>
<head>
<title>Регистрация на сайте</title>
<script>
//Функция проверки формы на наличие пустых полей
  function validateForm(){
```

```
    var validateFlag=true;  //возвращаемая функцией логическая переменная
    for (var i=0; i<6; i++) //Проверяем первые шесть элементов
    {
     //В действительности переменная countElem не нужна. Она введена для
     //улучшения читабельности кода, в нее записывается очередной элемент
      var countElem=document.regform.elements[i];
      if (countElem.value=="") //Текущий элемент пустой?
      { //Если да, то
        validateFlag=false;    //Устанавливаем флаг в false
        //Выдаем предупреждение для текущего элемента
        alert ("Введите значение в поле "+countElem.name);
        break;   //Цикл уже можно не продолжать
      }
    }
    // возвращаемое значение зависит от наличия пустых элементов
    return validateFlag;
  }
</script>
</head>
<body>
<h2>Регистрация на сайте</h2>
<!-- Создаем форму -->
<form name="regform" metod="post" onsubmit="return validateForm();">
<!--Для удобства отображения формы ее элементы располагаем в таблице-->
<table width="100%">
  <tr>
<!--Создаем поля ввода данных-->
    <td>Имя:<sup>*</sup></td>
<!-- Поле Имя -->
    <td><input type="text" size="35" name="firstname"></td>
  </tr>
  <tr>
    <td>Фамилия:<sup>*</sup></td>
<!-- Поле Фамилия -->
```

```
    <td><input type="text" size="35" name="lastname"></td>
  </tr>
  <tr>
    <td>Адрес Email:<sup>*</sup></td>
<!-- Поле Email -->
    <td><input type="text" size="35" name="email"></td>
  </tr>
  <tr>
    <td>Логин:<sup>*</sup></td>
<!-- Поле Логин -->
    <td><input type="text" size="35" name="login"></td>
  </tr>
  <tr>
    <td>Пароль:<sup>*</sup></td>
<!-- Поле Пароль -->
    <td><input type="password" size="20" name="pass1"></td>
  </tr>
  <tr>
    <td>Повтор пароля:<sup>*</sup></td>
    <!--Поле повтор пароля -->
    <td><input type="password" size="20" name="pass2"></td>
  </tr>
  <tr>
    <td>Страна:</td>
<!-- Далее список -->
    <td><select  size="1" name="country">
      <!-- Варианты списка -->
      <option value="Russia">Россия</option>
      <option value="Ukraine">Украина</option>
      <option value="Belorussia">Белоруссия</option>
      <option value="USA">США</option>
      <option value="France">Франция</option>
      <option value="Great Britain">Великобритания</option>
```

```
      <option value="other">Другая</option>
    </td>
  </tr>
  <tr>
    <td>Ваш пол:</td>
    <td>
<!-- Формируем переключатели -->
      <input type="radio" name="sex" value="male">Мужской<br />
      <input type="radio" name="sex" value="female">Женский<br/>
    </td>
  </tr>
  <tr>
    <td colspan=2>
<!-- Флажок со своим описанием занимает всю строку в таблице -->
    <input type="checkbox" name="consent">
       С условиями регистрации согласен.</td>
  </tr>
  <tr>
   <td colspan=2> Поля, помеченные звездочкой ( <sup>*</sup> ), являются
обязательными.</td>
 </tr>
 <tr>
<!-- Заканчивают форму две стандартные кнопки -->
    <td><input type="submit" value="Зарегистрировать"></td>
    <td><input type="reset" value="Очистить"></td>
  </tr>
</table>
</form>
</body>
</html>
```

Представленная форма (рис. 2.33) является типичной формой регистрации на сайтах. Вся проверка сводится к запуску функции `validateForm()` при отправке формы на сервер.

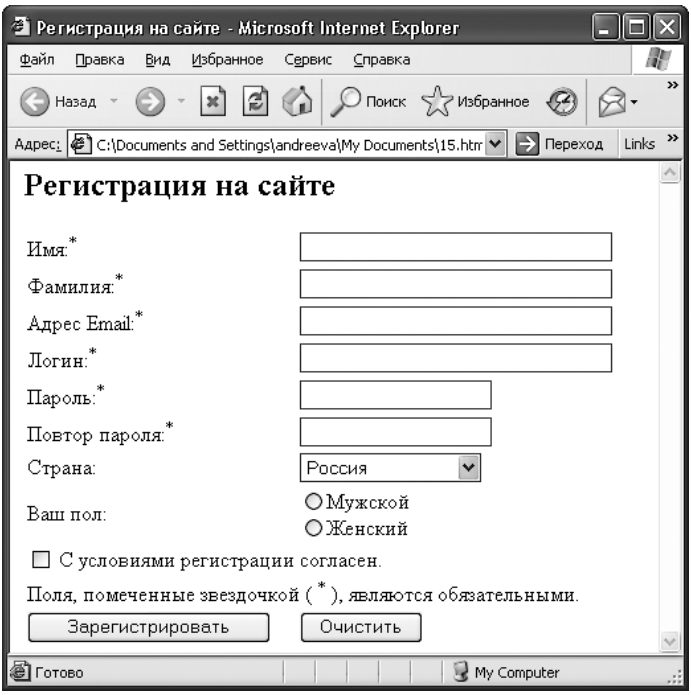


Рис. 2.33. Простая форма

Свойства, методы и события формы

Рассмотрим основные свойства, методы и события формы (табл. 2.17).

Таблица 2.17. Свойства, методы и события формы

Свойства	Методы	События
action	reset()	onReset
enctype	submit()	onSubmit
elements[]		
length		
method		
name		
target		

Свойство action

Свойство `action` задает URL-адрес сценария на сервере, которому отправляется форма:

```
<form name="myform"
  action=http://www.myserver.com/cgi-bin/registation.asp
  metod="post">
  <!-- Объекты формы -->
</form>
```

Но вместо URL-адреса можно указать и обработчик JavaScript:

```
<script>
  function handler (){
    alert('Функция JavaScript');
    void(0);
  }
</script>
<form name="myform"
  action="handler();"
  metod="post">
  <!-- Объекты формы -->
<input type=submit value="Обработка JavaScript в action">
</form>
```

Таким образом, после заполнения формы и нажатия кнопки запустится JavaScript.

Свойство enctype (encoding)

Свойство `enctype (encoding)` указывает тип MIME-кодирования формы. Значение по умолчанию — `application/x-www-form-urlencoded`.

Свойство elements[]

В массив `elements[]` включаются все элементы формы. К нему можно обратиться с помощью имени формы или массива `forms`. На элементы формы можно также ссылаться по их именам. Например, если элемент типа `text` с именем `email` является третьим элементом формы на странице HTML, то его значение можно получить тремя способами:

```
formName.elements[2].value
formName.elements["email"].value
formName.email.value
```

Свойство length

Свойство `length` определяет количество элементов на форме. Например, для определения количества элементов на третьей форме в документе необходимо воспользоваться следующей переменной:

```
document.forms[2].elements.length
```

Свойство method

Свойство `method` определяет способ отправки формы серверу. В зависимости от того, как автор HTML-страницы собирается получать и обрабатывать данные из формы, он может выбрать тот или иной метод доступа. Сервер можно настроить на использование различных методов, однако на практике чаще всего используются методы GET и POST.

Методы доступа GET и POST применяются в качестве типа передачи данных формы, однако их использование несколько отличается. JavaScript-программа может в любой момент изменить значение этого свойства. Ниже приводится код, в котором используются методы GET и POST.

```
<form name="myform">
<!-- Кнопка присваивает значение post свойству method-->
  <input type=button
    onclick="document.myform.method='post';
    alert(document.myform.method);" value="Метод post">
<!-- Кнопка присваивает значение get свойству method-->
  <input type=button
    onclick="document.myform.method='get';
    alert(document.myform.method);" value="Метод get">
  <input type=submit value="Отправить данные">
</form>
```

На данной форме имеются три кнопки (рис. 2.34), две из которых изменяют свойство `method`, а третья служит для отправки данных формы.

Свойство name

Свойство `name` возвращает строку со значением атрибута `name` формы. Это свойство в любой момент можно изменить.

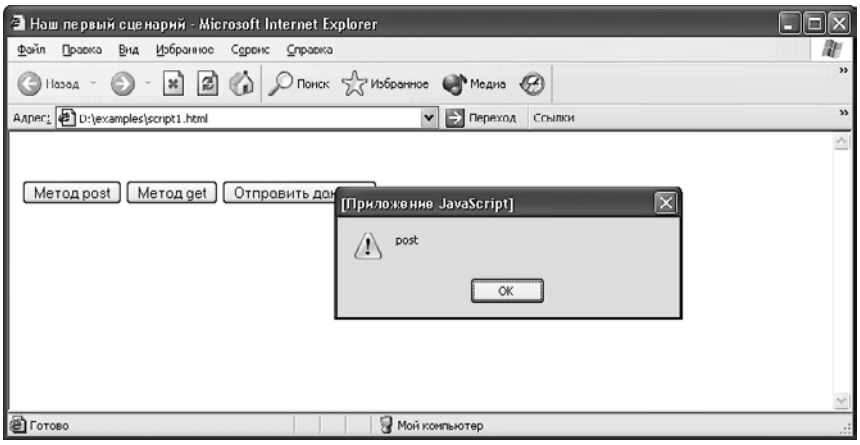


Рис. 2.34. Смена свойства metod

Свойство target

Свойство `target` указывает окно назначения браузера, в котором должен отображаться результат отправки формы. При использовании набора фреймов это свойство может быть фреймом. Кроме того, в качестве значения данного свойства могут выступать зарезервированные слова `_self`, `_blank`, `_top` и `_parent`.

Метод reset()

Метод `reset()` сбрасывает значения всех элементов формы в исходные состояния. При этом не нужно использовать кнопку `Reset`. Данный метод используется в следующем коде:

```
<form name="myform">
  Поле 1: <input type="text" SIZE=60
    value="Значение по умолчанию1" name="text1"></br>
  Поле 2: <input type="text" SIZE=60
    value="Значение по умолчанию1" name="text1"></br>
  Поле 3: <input type="text" SIZE=60
    value="Значение по умолчанию1" name="text1"></br>
  <!-- Кнопка, изменяющая значения полей -->
  <input type=button value="Изменим текст"
    onClick="document.myform.elements[0].value='Новый текст';
      document.myform.elements[1].value='Новый текст';
      document.myform.elements[2].value='Новый текст';">
```

```
</form>
<a href="javascript:document.myform.reset();void(0);">
Установить значение по умолчанию</a>
```

В этом примере возврат к форме значений происходит по гипертекстовой ссылке по умолчанию.

Метод submit()

Метод `submit()` используется для передачи данных формы. Действие этого метода аналогично эффекту от щелчка на кнопке `Submit`. Это позволяет передать данные на сервер по ссылке, а не с помощью кнопки, что иллюстрирует следующий код:

```
<form name="myform"
  action="http://myserver.com/cgi-bin/script.cgi">
  Поле 1: <input type="text" SIZE=60 name="text1"></br>
  Поле 2: <input type="text" SIZE=60 name="text1"></br>
  Поле 3: <input type="text" SIZE=60 name="text1"></br>
</form>
<a href="javascript:document.myform.submit();">
  Отправить данные</a>
```

Метод onReset

Событие `reset` восстанавливает значения по умолчанию в полях формы. Оно возникает при щелчке на кнопке `Reset` или при выполнении метода `reset()`. Функцию обработки данного события можно переопределить следующим образом:

```
<form name="myform"
  onReset="javascript:alert('Вы сбросили значения');
    return true;">
  Поле 1: <input type="text" size=60
    value="Значение по умолчанию1" name="text1"></br>
  Поле 2: <input type="text" size =60
    value="Значение по умолчанию1" name="text2"></br>
  Поле 3: <input type="text" size =60
    value="Значение по умолчанию1" name="text3"></br>
  <input type=reset value="Восстановить">
</form>
```

Если в обработчике события этого примера заменить возвращаемое логическое значение `true` на `false`, то стандартный обработчик не будет вызываться и событие будет обрабатываться только в новом обработчике.

Метод onSubmit

Событие `submit` возникает при щелчке на кнопке `Submit` или при вызове метода `submit()`. Функцию обработки данного события можно переопределить. Обработчик события `onSubmit` запускается тогда, когда пользователь отправляет данные формы на сервер. Любое отличное от `false` возвращаемое значение, в том числе пропуск оператора `return`, приводит к отправке данных на сервер. Это позволяет отменить отправку данных, если пользователь заполнил форму некорректно. Ниже приводится код, в котором используется событие `submit`.

```
<script>
  //Функция, вызываемая при отправке формы
  function checkSubmit() {
    //Проверяем значение поля
    if (document.myform.text1.value==""){
      //Если поле пустое, предупреждаем и отменяем вызов
      //стандартного обработчика
      alert('Введите имя');
      return false; }
    else {//иначе, вызываем стандартный обработчик
      return true }
  }
</script>
<form name="myform" onSubmit="return checkSubmit();"
  action="http://myserver.com/cgi-bin/reg.pl>
  Имя: <input type="text" size=60 name="text1"></br>
  <input type=submit value="Отправить">
</form>
```

Примеры сценариев

Рассмотрим два простых примера, иллюстрирующих практическое применение языка сценариев JavaScript в формах.

Решение квадратного уравнения

В следующей программе (листинг 2.43) показывается решение квадратного уравнения. Напомним, что квадратным уравнением называется уравнение вида

$ax^2+bx+cx=0$.

Корни этого уравнения равны

$x_1=(-b+\sqrt{D})/2a$, $x_2=(-b-\sqrt{D})/2a$,

где D — дискриминант, определяемый следующим образом:

$D=b^2-4ac$.

В текстовые поля a, b и c необходимо ввести соответствующие коэффициенты квадратного уравнения.

Листинг 2.43. Решение квадратного уравнения

```
<html>
  <head>
    <title>Решение квадратного уравнения</title>
    <script>
      //Эта функция высчитывает корни уравнения
      //Ее аргумент – объект формы.
      function Calculate(FormObj){
        //Умножаем значения полей на единицу, чтобы преобразовать
        //их в числа
        var a=FormObj.a.value*1;
        var b=FormObj.b.value*1;
        var c=FormObj.c.value*1;
        //Проверяем поля a, b и c
        //Если там нет чисел, то inNaN возвращает true
        if (isNaN(a)){
          alert("a должно быть числом");
          return;
        }
        if (isNaN(b)){
          alert("b должно быть числом");
          return;
        }
```

```
if (isNaN(c)){
  alert("c должно быть числом");
  return;
}
//Проверяем, квадратное ли это уравнение
//В квадратном уравнении a не должно быть нулем
//А если и a и b равны нулю, то это вообще не уравнение.
if (a==0){
  if (b==0){
    //Это вообще не уравнение и не имеет смысла при c<>0
    FormObj.x1.value="Любой";
    FormObj.x2.value="Любой";
    FormObj.D.value="";
    return;
  }
  //Это обычное линейное уравнение, у которого один корень
  FormObj.x1.value=c/b;
  FormObj.x2.value=c/b;
  FormObj.D.value="";
  return;
}
//Находим дискриминант
var Dis=b*b-4*a*c;
FormObj.D.value=Dis;
if (Dis<0){
  alert("Дискриминант отрицательный");
  FormObj.x1.value="Нет действительных корней";
  FormObj.x2.value="Нет действительных корней";
  return;
}
else{
  //Находим квадратный корень дискриминанта
  Dis=Math.sqrt(Dis);
  //Находим корни уравнения
  FormObj.x1.value=(-b+Dis)/(2*a);
```

```
          FormObj.x2.value=(-b-Dis)/(2*a);
        }
      }
    </script>
  </head>
  <body>
    <h2></h2>
    <form name="cal">
       <table>
         <tr>
           <td>a:</td>
           <td><input type="text" size=30 name="a"></td>
         </tr>
         <tr>
           <td>b:</td>
           <td><input type="text" size=30 name="b"></td>
         </tr>
         <tr>
           <td>c:</td>
           <td><input type="text" size=30 name="c"></td>
         </tr>
         <tr>
           <td>D:</td>
           <td><input type="text" size=30 name="D"></td>
         </tr>
         <tr>
           <td>x1:</td>
           <td><input type="text" size=30 name="x1"></td>
         </tr>
         <tr>
           <td>x2:</td>
           <td><input type="text" size=30 name="x2"></td>
         </tr>
      </table>
```

```
      <input type="button" Value="Посчитать"
        onClick="Calculate(this.form)">
    </form>
  </body>
</html>
```

Помимо определения дискриминанта и корней уравнения, в функции `Calculate()` проверяются введенные данные. Это делается для того, чтобы не позволить пользователю ввести некорректные данные и избежать деления на нуль (рис. 2.35).

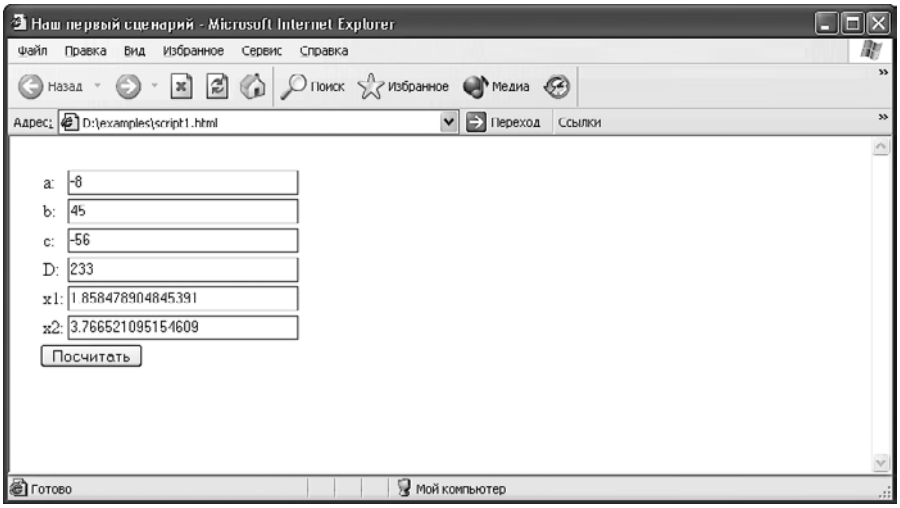


Рис. 2.35. Решение квадратного уравнения

Таким образом, основная часть кода сводится к проверке данных, а не к расчетам. Если же форма отправляется на сервер, то ее также необходимо проверить, так как пользователь может отослать в форме некорректные данные.

Использование поисковых систем

Многие сайты предоставляют возможность поиска информации через различные поисковые системы. В листинге 2.44 представлен код формы, которая позволяет осуществить поиск с помощью популярных поисковых систем.

Листинг 2.44. Использование поисковых систем

```
<html>
  <head>
    <title>Поиск</title>
```

```
    <script>
      //Строки для поиска
      UrlYandex="http://www.yandex.ru/yandsearch?rpt=rad&text=";
      UrlRambler="http://www.rambler.ru/srch?words=";
      UrlAport="http://sm.aport.ru/scripts/template.dll?That=std&r=";
      UrlGoogle="http://www.google.ru/search?hl=ru&newwindow=1&q=";
      function FindText(Url){
        var FullUrl=Url+document.find.searchtext.value;
         //Проверяем, установлен ли флажок
         if (document.find.newwin.checked){
           var FindWin=window.open(FullUrl);
         }
         else{
           location.href = FullUrl;
         }
      }
    </script>
  </head>
  <body>
    <form name="find">
      Найти:<br>
      <input type="text" name="searchtext" size=40><br>
      <input type="checkbox" name="newwin">Открыть в новом окне<br>
      <br>С помощью поисковой системы:<br>
      <input type="button" value="Yandex"
        onClick="FindText(UrlYandex)">
      <input type="button" value="Rambler"
        onClick="FindText(UrlRambler)">
      <input type="button" value="Aport"
        onClick="FindText(UrlAport)">
      <input type="button" value="Google"
        onClick="FindText(UrlGoogle)">
    </form>
  </body>
</html>
```

Здесь по щелчку на любой из кнопок появляется функция `FindText()`, в которой вызывается соответствующая страница (рис. 2.36).

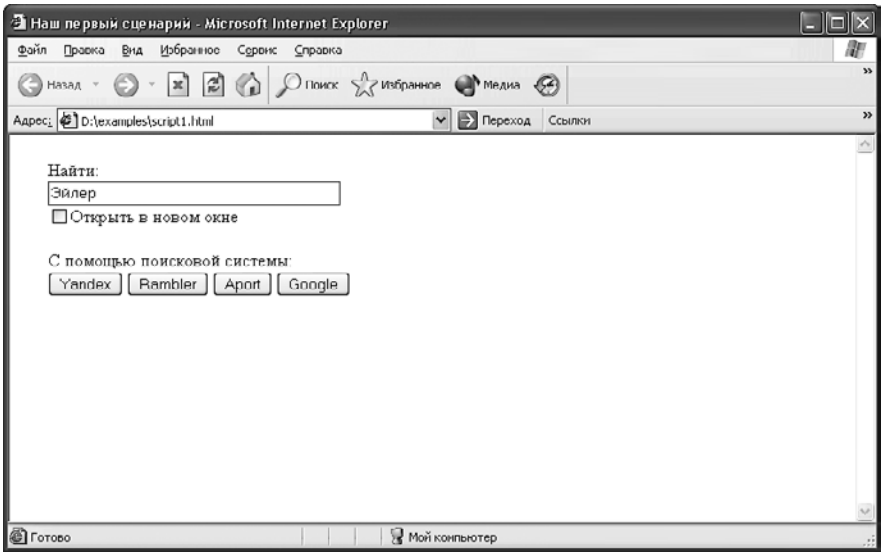


Рис. 2.36. Поиск в поисковых системах

Чтобы найти строки для поиска поисковых машин, нужно зайти на сайт и начать поиск какого-либо слова в адресной строке браузера. В результате вы получите требуемый адрес за вычетом введенного слова.

При вводе текста поиска и щелчке на одной из кнопок загрузится страница с соответствующей поисковой системой. Если же установить флажок Открыть в новом окне, то страницы будут открываться в новом окне и работу с этим документом можно продолжить.

Следует отметить, что все поисковые машины позволяют проводить поиск на одном сайте, что, в свою очередь, ограничивает поиск только по вашему сайту. К примеру, в Яндексе для этого к запросу нужно добавить параметр `site=www.mysite.ru`:

```
http://www.yandex.ru/yandsearch?site=www.mysite.ru&rpt=rad&text=
```

Динамический HTML

JavaScript позволяет придавать документу новое качество — возможность динамического изменения страницы без перезагрузки документа. Динамический HTML позволяет языку JavaScript полностью контролировать все элементы документа,

параметры их оформления и размещения, а также текст страницы. Благодаря этому страницы могут динамически изменять стили, содержание и оформление. Кроме того, можно создавать интерактивные документы, взаимодействующие с пользователем в режиме реального времени.

Управление текстом

Несмотря на все мультимедийные возможности, основным источником информации в Интернете остается текст. HTML и другие подобные веб-технологии имеют ограничения, не позволяющие управлять разметкой страницы, как в издательском деле. Кроме того, пользователь может изменить разрешение экрана, размер страницы, шрифт, цвета и т. д. Различные браузеры по-разному отображают одну и ту же страницу. Использование графических форматов вместо текста пока недопустимо для решения этой проблемы, так как размер страницы увеличивается в сотни раз. Дизайнер должен проектировать сайт, учитывая эти ограничения.

Вы уже познакомились с объектной моделью документа (DOM). Теперь рассмотрим, каким образом она используется для управления текстом.

Язык JavaScript позволяет управлять любым элементом на странице в режиме реального времени: скрывать, удалять либо добавлять элементы, копировать их, изменять такие свойства элементов, как цвет, ширина, высота и др.

Каждая составляющая дерева DOM-модели называется *узлом*. Узлом может быть элемент, текст, атрибут и т. д. Узел описывается объектом `Node`, который является первичным типом данных DOM. Объект `Node` является основополагающим, поэтому приведем его основные свойства и методы (табл. 2.18).

Таблица 2.18. Свойства и методы объекта Node

Тип	Имя	Описание
Свойства	nodeName	Имя текущего узла — строка только для чтения
	nodeValue	Значение текущего узла — строка только для чтения
	nodeType	Тип объекта текущего узла — целое число только для чтения, равное одной из констант объекта Node
	parentNode	Родительский узел текущего узла — объект Node
	childNodes	Список узлов, содержащий всех потомков текущего узла, — объект NodeList

Тип	Имя	Описание
	firstChild	Первый дочерний узел текущего узла — объект Node
	lastChild	Последний дочерний узел текущего узла — объект Node
	previousSibling	Предшествующий узел — объект Node
	nextSibling	Последующий узел — объект Node
	attributes	Коллекция узлов, содержащая все атрибуты данного узла, — объект NamedNodeMap
	ownerDocument	Объект Document, связанный с текущим узлом
	namespaceURI	URI пространства имен данного узла — строка только для чтения
	prefix	Префикс пространства имен данного узла
	localName	Локальная часть квалифицированного имени текущего узла — строка только для чтения, например имя тега
	baseURI	Базовый URI — строка только для чтения, адрес документа
	textContent	Текстовое содержимое текущего узла — текст в дочернем узле #text
Методы	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 узел newChild в конец списка потомков текущего узла
	hasChildNodes()	Возвращает true, если текущий узел имеет потомков, иначе возвращает false
	cloneNode(deep)	Возвращает копию данного узла. Если логическая переменная deep равна true, то копируются и текущий узел, и его потомки
	normalize()	Нормализует текущий узел — перемещает узлы Text вглубь поддерева текущего узла
	isSupported(feature, version)	Возвращает true, если текущий узел поддерживает возможность feature версии version, иначе возвращает false
	hasAttributes()	Возвращает true, если текущий узел имеет атрибуты, иначе возвращает false

Это далеко не полный перечень аргументов объекта `Node`, но даже они не все поддерживаются различными браузерами.

Доступ к узлам

Рассмотрим, каким образом можно получить доступ к DOM-модели с помощью языка JavaScript. Эта задача реализуется за счет использования следующих способов:

- метода `getElementsByTagName()`, возвращающего список узлов с указанным именем тега;
- метода `getElementById()`, который передает уникальный атрибут `id` необходимого элемента;
- свойства `documentElement` объекта `Document` и древовидной структуры DOM, которые требуются для доступа к необходимому элементу.

В листинге 2.45 демонстрируются все способы, реализованные в функциях `View1()`, `View2()` и `View3()`, которые совершают идентичные действия — возвращают имя и текст соответствующего тега. Работа JavaScript в различных браузерах немного отличается, поэтому и коды для них нужно выполнять разные.

Листинг 2.45. Доступ к узлам

```
<html>
  <head>
    <title>Доступ к узлам</title>
    <script>
      //Функция для первого тега <h2>
      function View1(){
        //получаем список всех тегов <h2>в документе
        var H2List=document.getElementsByTagName("h2");
        //Обращаемся к первому узлу в списке
        var Header1=H2List.item(0);
        //Выводим сообщение с текстом и именем тега
        alert(Header1.firstChild.nodeValue+" "+Header1.tagName);
      }
      //Функция для второго тега <h2>
      function View2(){
        //Обращаемся ко второму тегу с id="h2Tag2"
        var Header2=document.getElementById("h2Tag2");
        //Выводим сообщение с текстом и именем тега
```

```
        alert(Header2.firstChild.nodeValue+" "+Header2.tagName);
      }
      //Функция для третьего тега <h2>
      function View3(){
        //Специфика DOM-модели браузера Mozilla Firefox
        if (navigator.appName=="Netscape") {
          //Обращаемся к тегу <body>
          var BodyNode= document.documentElement.childNodes[2];
          //Обращаемся к третьему тегу <h2> в теле документа
          var Header3=BodyNode.childNodes[7];
          //Выводим сообщение с текстом и именем тега
          alert(Header3.firstChild.nodeValue+" "+Header3.tagName);
       }
        //Специфика DOM-модели браузера Opera
       if (navigator.appName=="Opera") {
          //Обращаемся к тегу <body>
          var BodyNode= document.documentElement.childNodes[1];
          //Обращаемся к третьему тегу <h2> в теле документа
          var Header3=BodyNode.childNodes[7];
          //Выводим сообщение с текстом и именем тега
          alert(Header3.firstChild.nodeValue+" "+Header3.tagName);
       }
        //Специфика DOM-модели браузера Microsoft Internet Explorer
       if (navigator.appName=="Microsoft Internet Explorer") {
          //Обращаемся к тегу <body>
          var BodyNode= document.documentElement.childNodes[1];
          //Обращаемся к третьему тегу <h2> в теле документа
          var Header3=BodyNode.childNodes[3];
          //Выводим сообщение с текстом и именем тега
          alert(Header3.firstChild.nodeValue+" "+Header3.tagName);
       }
}
    </script>
  </head>
```

```
  <body>
    <!-- Заголовки, к которым мы обращаемся -->
    <h2 id="h2Tag1">Первый заголовок</h2>
    <h2 id="h2Tag2">Второй заголовок</h2>
    <h2 id="h2Tag3">Третий заголовок</h2>
    <form>
      <input type=button value="Показать1" onClick="View1();">
      <input type=button value="Показать2" onClick="View2();">
      <input type=button value="Показать3" onClick="View3();">
    </form>
  </body>
</html>
```

После щелчка на кнопке Показать1 вызывается метод `getElementsByTagName()` для некоторого списка тегов `<h2>`. Данный список содержит три элемента. Нас интересует первый. Для доступа к нему вызываем метод `item()` для объекта `NodeList`.

ПРИМЕЧАНИЕ



Обратите внимание, что работа с текстом немного отличается от работы с другими элементами DOM. Каждый фрагмент текста на странице помещается в невидимый узел #text. Так, текст тега ‹h2› хранится в дочернем узле, и для доступа к строке этого текста необходимо использовать свойство firstChild объекта Node.

После щелчка на кнопке Показать2 вызывается метод `getElementById()` с соответствующим идентификатором `h2Tag2` для второго заголовка. Как видите, это самый простой и быстрый метод (рис. 2.37).

После щелчка на кнопке Показать3 вызывается функция `View3()`, в которой учитывается специфика работы с DOM-моделью различных браузеров. Алгоритм в каждом случае одинаковый. Свойство `documentElement` документа ссылается на объект `Element`, который представляет сам документ. Определяем его дочерний узел, соответствующий тегу `<body>` в переменной `BodyNode`, а затем находим нужный дочерний узел тега `<h2>`. Отличия в работе функции `View3()` связаны с тем, что браузеры Opera и Mozilla Firefox учитывают пустые текстовые строки между тегами заголовков, введенные для удобства чтения. Кроме того, изменения документа повлекут за собой изменения объектов, а соответственно, и кода. Таким образом, этот способ является довольно трудоемким в реализации.

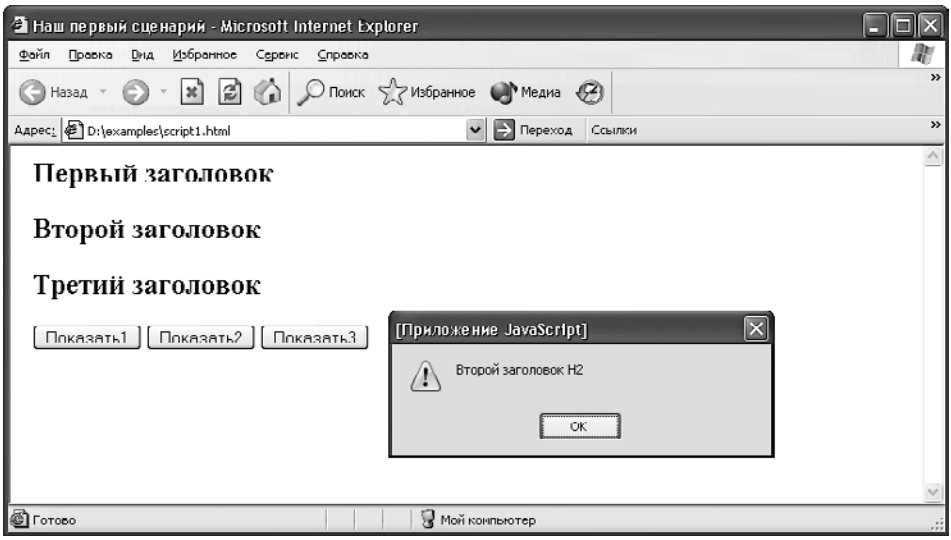


Рис. 2.37. Доступ к узлам

Доступ к атрибутам

Атрибуты элемента — это переменная и ее значение, которые указываются в угловых скобках (<>) тега. Например, тег `<img>` имеет атрибут `src`, указывающий на графический файл. Чтобы получить значение атрибута, необходимо воспользоваться методом `getAttribute()` объекта `Element`. Параметром данного метода выступает соответствующее имя атрибута элемента (листинг 2.46).

Листинг 2.46. Доступ к атрибутам

```
<html>
  <head>
    <title>Доступ к атрибутам</title>
  </head>
  <body>
    <h2 id="h2Tag" align="center">Заголовок</h2>
    <form>
      Текст <input type="text" id="text1"><br>
      <input type=submit id="But" >
    </form><br>
    <script>
      //Обращаемся к заголовку
      var Elem=document.getElementById("h2Tag");
```

```
      //Получаем атрибуты элемента
      document.write("Элемент: "+Elem.tagName+", id=");
      document.write(Elem.getAttribute("id")+", выравнивание-"
        +Elem.getAttribute("align"));
      document.write("<br>");
      //Обращаемся к текстовому полю
      Display(document.getElementById("text1"));
      //Обращаемся к кнопке
      Display(document.getElementById("But"));
      //Функция, выводящая атрибуты элементов формы
      function Display(El){
        document.write("Элемент: "+El.tagName+", id=");
        document.write(El.getAttribute("id")+", тип-"
          +El.getAttribute("type"));
        document.write("<br>");
      }
    </script>
  </body>
</html>
```

В этом сценарии с помощью метода `getAttribute()` выводятся значения некоторых атрибутов заголовка, текстового поля и кнопки (рис. 2.38). Интересующие нас атрибуты кнопки и текстового поля совпадают, поэтому вывод их значений объединяется в функцию `Didsplay()`, которая в качестве параметра получает соответствующий объект `Element`.

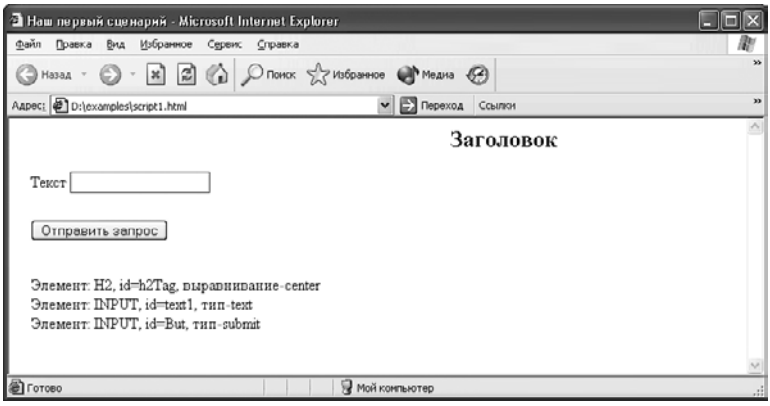


Рис. 2.38. Доступ к атрибутам

Изменение текста

На странице можно изменять не только текст, но и его атрибуты. Для этого необходимо воспользоваться методом `setAttribute()` объекта `Element`. Параметрами данного метода выступают имя атрибута элемента и соответствующее ему значение (листинг 2.47).

Листинг 2.47. Изменение текста

```
<html>
  <head>
    <title>Изменение текста</title>
    <script>
      //Функция, изменяющая текст на странице
      function Change(){
        //Обращаемся к заголовку
        var Elem=document.getElementById("h2Tag");
        //Назначаем атрибуту align новое значение
        Elem.setAttribute("align","left");
        //Обращаемся к тексту, выделенному курсивом
        var Elem=document.getElementById("iTag");
        //Изменяем текст
        Elem.firstChild.nodeValue="Текст2";
      }
    </script>
  </head>
  <body>
    <h2 id="h2Tag" align="center">Заголовок</h2>
    <i id="iTag">Текст</i>
    <form>
      <input type=button id="But" value="Изменить" onClick="Change();">
    </form>
  </body>
</html>
```

Щелчок на кнопке данной страницы вызывает функцию `Change()`, с помощью которой можно изменить текст. Метод `setAttribute()` назначает выравнивание по левой стороне для заголовка. Обращаясь к дочернему узлу тега `<i>`, можно изменить текст, выделенный курсивом (рис. 2.39).

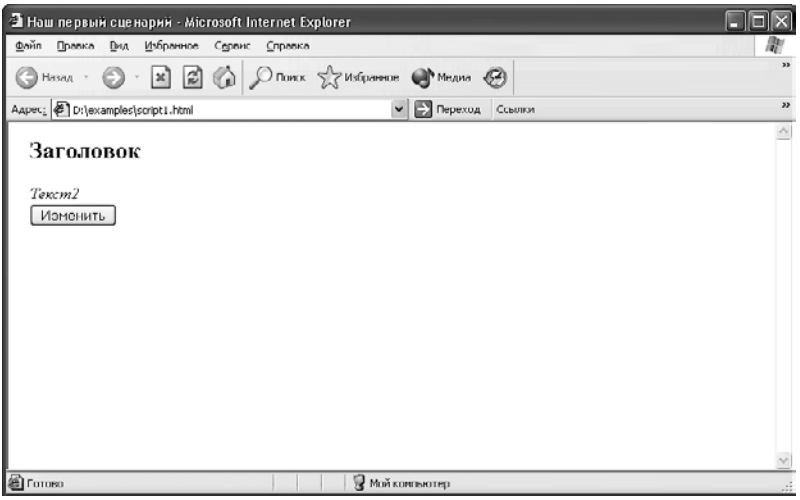


Рис. 2.39. Изменение текста

Графика на веб-страницах

Текст на веб-страницах необходим, но порой один рисунок позволяет донести больше информации, чем сотни строк текста. Помимо наглядности, графика имеет и другое преимущество — всевозможные эффекты, которые могут улучшить привлекательность сайта.

События

Наиболее часто используемой возможностью визуальных эффектов является применение событий.

Для визуальных эффектов накатов (rollover) используются четыре типа событий:

- `onMouseDown` — обработчик события, который активизируется при щелчке кнопкой мыши на элементе;
- `onMouseUp` — обработчик события, который активизируется при отпускании кнопки мыши после щелчка на элементе;
- `onMouseOver` — обработчик события, который активизируется при наведении указателя мыши на элемент;
- `onMouseOut` — обработчик события, который активизируется при покидании указателем мыши элемента.

Эти события применяются в листинге 2.48.

Листинг 2.48. События

```
<html>
  <head>
    <title>События</title>
   </head>
  <body>
    <h2>Демонстрация событий</h2>
    <a href="javascript:void(0)"
      onMouseDown="document.pic.src='onMouseDown.gif'"
      onMouseUp="document.pic.src='onMouseUp.gif'"
      onMouseOver="document.pic.src='onMouseOver.gif'"
      onMouseOut="document.pic.src='onMouseOut.gif'">
      <img src="onMouseOut.gif" border=0 name="pic">
    </a>
  </body>
</html>
```

На странице выводится изображение onMouseOut.gif, которое при наступлении какого-либо события сменяется на другую картинку (рис. 2.40).



Рис. 2.40. Демонстрация событий

Смена изображения

Только что мы рассмотрели, как происходит обработка событий. Однако таким образом, как указывается в листинге 2.48, в реальных веб-страницах динамика не

реализуется, так как медленная загрузка картинок с сервера не позволит создать быструю смену изображений. Поэтому перед использованием изображений их надо загрузить. Наиболее просто эта задача реализуется при создании двух массивов, в одном из которых хранятся статичные изображения, а в другом — те, которые используются для наката (листинг 2.49).

Листинг 2.49. Смена изображений

```
<html>
  <head>
    <title>Смена изображений</title>
    <script>
      //Создание массивов для хранения изображений
      //Массив с 6 элементами, так как 0 мы не будем использовать
      numerals=new Array(6);
      letters=new Array(6);
      //В цикле создаем объекты изображений и загружаем рисунки
      for(var i=1;i<6;i++){
        //Создаем объекты изображений
        numerals[i] =new Image();
        letters[i] =new Image();
        //Загружаем все рисунки
        numerals[i].src ="img"+i+".gif";
        letters[i].src ="pic"+i+".gif";
      }
      //Функция, сменяющая изображение при событии onMouseOut
      function ImageOut(img){
        document.images[img-1].src=numerals[img].src;
      }
      //Функция, сменяющая изображение при событии onMouseOut
      function ImageOver(img){
        document.images[img-1].src=letters[img].src;
      }
    </script>
  </head>
  <body>
```

```
    <h2>Смена изображений</h2>
    <!-- Создаем пять ссылок с рисунками -->
    <a href="javascript:void(0)"
      onMouseOver="ImageOver(1)"
      onMouseOut="ImageOut(1)">
      <img src="img1.gif" width=81 height=81>
    </a>
    <a href="javascript:void(0)"
      onMouseOver="ImageOver(2)"
      onMouseOut="ImageOut(2)">
      <img src="img2.gif" width=81 height=81>
    </a>
    <a href="javascript:void(0)"
      onMouseOver="ImageOver(3)"
      onMouseOut="ImageOut(3)">
      <img src="img3.gif" width=81 height=81>
    </a>
    <a href="javascript:void(0)"
      onMouseOver="ImageOver(4)"
      onMouseOut="ImageOut(4)">
      <img src="img4.gif" width=81 height=81>
    </a>
    <a href="javascript:void(0)"
      onMouseOver="ImageOver(5)"
      onMouseOut="ImageOut(5)">
      <img src="img5.gif" width=81 height=81>
    </a>
  </body>
</html>
```

После выполнения данного сценария создается страница, на которой располагаются пять рисунков с цифрами от 1 до 5. При наведении указателя мыши на любой из них изображение сменяется на картинку соответствующей латинской буквы от A до E (рис. 2.41).

Во время загрузки страницы изображения загружаются в массивы `numerals` и `letters`. Мы создали эти массивы с количеством элементов, равным 6, нулевой элемент при этом не используется. Следовательно, для доступа к массиву изображений документа `document.images[]` нужно вычесть единицу из передаваемого параметра в функциях обработки событий `ImageOut()` и `ImageOver()`, так как нумерация картинок в документе начинается с нуля.

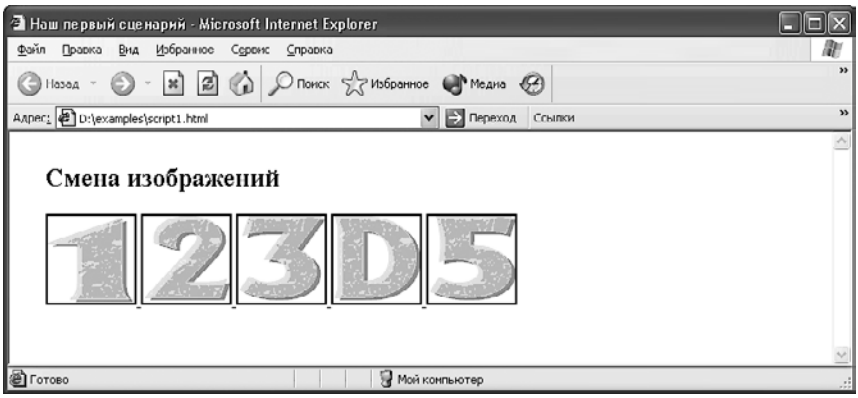


Рис. 2.41. Смена изображений

Анимированные кнопки

В вышеприведенном примере для перехвата событий использовались ссылки. Такого же эффекта можно достичь, если перехватывать события самого изображения. Более того, можно вообще отказаться от ссылок, а перехватывать событие щелчка на кнопку с изображением. Таким образом, вместо ссылок в листинге 2.49 можно использовать код следующего вида:

```
<img src="img1.gif" width=81 height=81 onMouseOver="ImageOver(1)"
  onMouseOut="ImageOut(1)" onClick="javascript:void(0)">
```

Анимация изображения

Для создания анимации изображения необходимо через равные промежутки времени изменять атрибут `src` в изображении. Для этого используется одна из функций объекта `Window`: `setTimeout()` или `setInterval()`.

Листинг 2.50. Анимация изображения

```
<html>
  <head>
    <title>Анимация изображения</title>
```

```
    <script>
      //Создаем массив для хранения изображений
      numerals=new Array(6);
      //В цикле создаем объекты изображений и загружаем рисунки
      for(var i=1;i<6;i++){
        //Создаем объекты изображений
        numerals[i] =new Image();
        //Загружаем все рисунки
        numerals[i].src ="img"+i+".gif";
      }
      //Функция, осуществляющая анимацию
      function animation(count){
        //Загружаем текущий рисунок
        document.images[0].src="img"+count+".gif";
        count--;//Уменьшаем счетчик
        //Пока есть изображения, рекурсивно вызываем функцию animation()
        if (count>=0){
          //Откладываем вызов функции на секунду
          setTimeout("animation("+count+");",1000);
        }
      }
    </script>
  </head>
  <body>
    <h2>Анимация изображения</h2>
      <img src="img5.gif" width=81 height=81  onClick="animation(5)">
  </body>
</html>
```

Функция `animation()` получает управление после щелчка на рисунке, затем она изменяет рисунок, уменьшает счетчик и порождает другой свой экземпляр, выполнение которого откладывается на 1 с. Текущая функция уничтожается. Когда счетчик становится отрицательным, то функция уже не порождается и анимация прекращается. В результате выполнения этого кода на странице отображается рисунок с цифрой 5, на котором после щелчка последовательно изменяются изображения цифр от 5 до 0 (рис. 2.42).

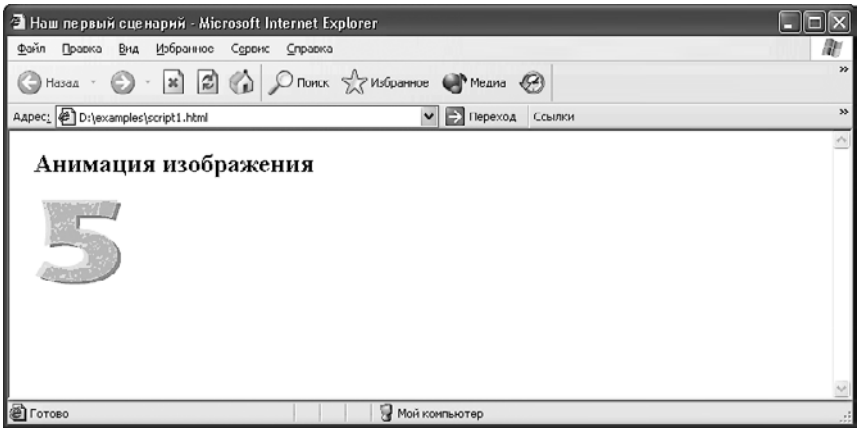


Рис. 2.42. Анимация изображения

Слои

Сейчас я расскажу о размещении блочных элементов HTML-разметки в рабочей области браузера с точностью до пиксела, а также о других возможностях слоев.

Для создания слоев обычно используется элемент `<div>`, являющийся блочным элементом разметки. Этот элемент просто определяет данные, но не указывает браузеру, как их отображать.

Таблицы стилей позволяют по-всякому манипулировать слоями. Из предыдущей главы вы узнали о таких свойствах слоев, как `position` и `z-index`. Рассмотрим пример, позволяющий управлять свойством `z-index` (листинг 2.51).

Листинг 2.51. Управление порядком слоев

```
<html>
  <head>
    <title>Управление порядком слоев</title>
    <script>
      //Функция, осуществляющая смену порядка слоев
      function move(){
        //Получаем доступ к слоям
        var Layer1=document.getElementById('first');
        var Layer2=document.getElementById('second');
```

```
        //Выясняем текущий порядок слоев
        if (Layer1.style.zIndex<Layer2.style.zIndex){
          //Если сверху 2 слой
          Layer1.style.zIndex=1;
          Layer2.style.zIndex=0;
        }
        else{
          //Если сверху первый слой
          Layer1.style.zIndex=0;
          Layer2.style.zIndex=1;
        }
      }
    </script>
  </head>
  <body>
    <h2>Управление порядком слоев</h2>
    <div id=first style="position:absolute; top:50px;left:50px;
      height:100px;width:250px; background-color:orange; z-index:0">
      1 слой
    </div>
    <div id=second style="position:absolute; top:80px;left:80px;
      height:100px;width:250px; background-color:yellow; z-index:1">
      2 слой
    </div>
    <!-- Вспомогательный блок, в котором находится ссылка, -->
    <!-- вызывающая функцию move() -->
    <div id=control style="position:absolute; top:200px;left:80px;">
      <a href="javascript:move();">Переложить слои</a>
    </div>
  </body>
</html>
```

В этом примере для доступа к слоям использовался метод `getElementById()`. На странице, сформированной с помощью данного кода, отображаются два слоя. Щелчок на ссылке изменяет их порядок (рис. 2.43).

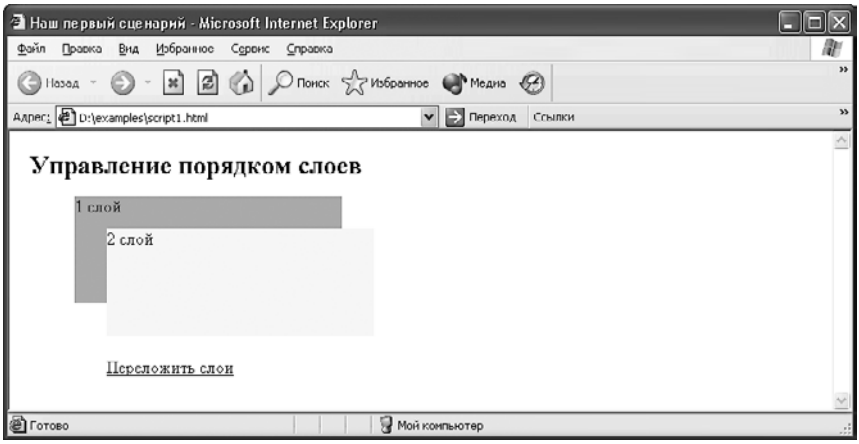


Рис. 2.43. Управление порядком слоев

Рассмотрим также полезное свойство `visibility`, которое позволяет скрывать и делать видимыми слои. Оно может принимать следующие значения:

- `hidden` — скрывает слой;
- `inherit` — указывает, что видимость наследуется от родителя (значение по умолчанию);
- `visible` — делает слой видимым.

В листинге 2.52 задается управление видимостью слоев.

Листинг 2.52. Управление видимостью слоев

```
<html>
  <head>
    <title>Управление видимостью</title>
    <script>
      //Функция, осуществляющая изменение видимости слоя
      function vis(){
        //Получаем доступ к слою
        var Layer=document.getElementById('lay');
        //Выясняем текущее состояние слоя
        if (Layer.style.visibility=='hidden'){
          //Если слой невидим, то делаем видимым
          Layer.style.visibility='visible';
        }
```

```
        else{
          //Если слой видим, то делаем невидимым
          Layer.style.visibility='hidden';
        }
      }
    </script>
  </head>
  <body>
    <h2>Управление видимостью</h2>
    <div id=control style="position:absolute; top:50px;left:50px;">
      <a href="javascript:vis();">Скрыть/Показать слой</a>
    </div>
    <div id=lay style="position:absolute; top:70px;left:50px;
      height:100px;width:250px; background-color:orange;
      visibility:hidden">
      Этот слой по умолчанию невидим, чтобы его скрыть или увидеть,
      необходимо щелкнуть на ссылке
    </div>
  </body>
</html>
```

Этот код позволяет создать невидимый слой, который можно отобразить или вновь скрыть щелчком на ссылке (рис. 2.44).

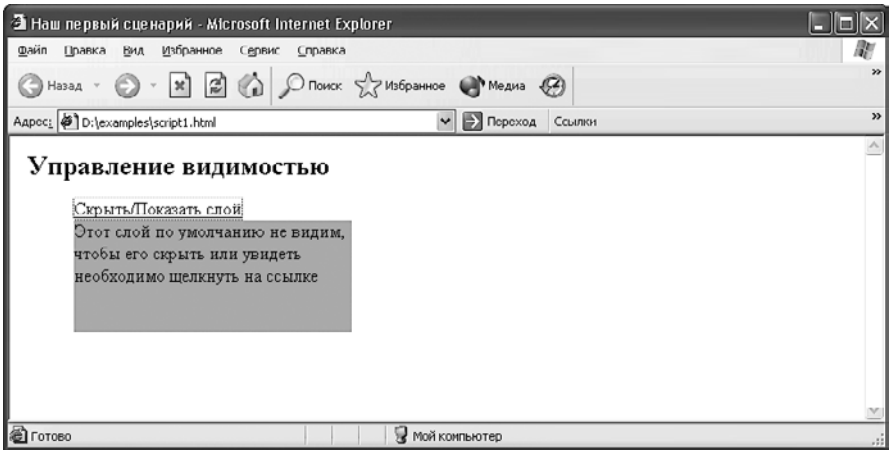


Рис. 2.44. Управление видимостью слоев

Доступ к стилям с помощью JavaScript

Стили очень важны при оформлении, а следовательно, и для реализации динамики на странице.

Для доступа к элементу в целях его изменения можно воспользоваться методом `getElementById()`. Но можно воспользоваться более простым способом и обращаться к свойству элемента напрямую по идентификатору (листинг 2.53).

Листинг 2.53. Доступ к свойствам таблицы стилей

```
<html>
  <head>
    <title>Доступ к свойствам таблицы стилей</title>
    <style type="text/css">
      p {
        color:black;
        font-size: 14pt;
        font-style:normal;
      }
    </style>
    <script>
      //Функция, изменяющая стиль при наведении указателя мыши
      function Change1(){
        par.style.color="red"; //Цвет текста красный
        par.style.fontSize="16pt"; //Размер шрифта 16 пунктов
        par.style.fontStyle="italic"; //Курсив
      }
      //Функция, восстанавливающая стиль при покидании указателя мыши
      function Change2(){
        par.style.color="black";
        par.style.fontSize="14pt";
        par.style.fontStyle="normal";
      }
    </script>
  </head>
  <body>
    <h2>Доступ к свойствам таблицы стилей</h2>
```

```
    <p>Это обычный текст.<p>
    <p id="par" onMouseOver="Change1()" onMouseOut="Change2()">
      Текст, изменяющий свой вид при наведении указателя мыши.
    </p>
  </body>
</html>
```

В этом листинге абзац с идентификатором `par` реагирует на два события: наведение указателя мыши на элемент и его покидание. События обрабатываются в двух почти идентичных функциях, которые задают значения свойств стиля данного элемента (рис. 2.45).

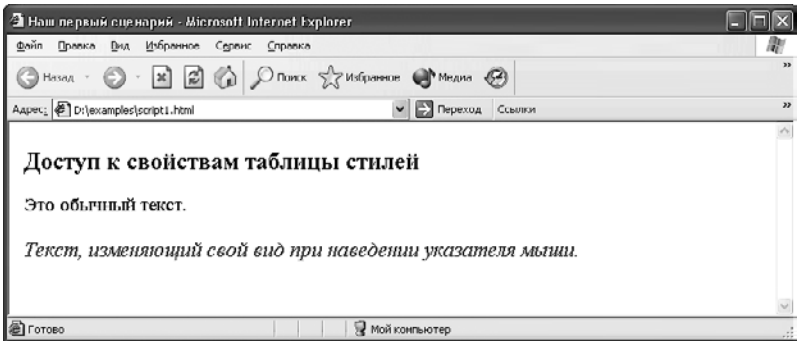


Рис. 2.45. Доступ к свойствам таблицы стилей

ПРИМЕЧАНИЕ



Если в сценариях JavaScript используются атрибуты, в названиях которых есть дефис, то уберите его и пишите слова слитно, причем второе слово должно начинаться с заглавной буквы. Например, backgroundColor и textDecoration.

Пример создания динамической игры

Работа — дело нужное, но иногда можно и расслабиться. Рассмотрим работу со слоями на примере простой игры. Цель этой игры состоит в том, чтобы попасть в случайно двигающийся квадрат (листинг 2.54).

Листинг 2.54. Мишень

```
<html>
  <head>
    <title>Мишень</title>
    <script>
```

```
var vx;//Скорость по горизонтали
var vy;//Скорость по вертикали
var flag;//Флаг, попадания в цель
//Функция инициализации
function init(){
  flag=false;//Сброс флага
  lay.style.backgroundColor="orange";//Цвет квадрата – оранжевый
  //Задаем случайные координаты для квадрата в пределах
  //поля игрового
  lay.style.top=Math.round((Math.random()*450));
  lay.style.left=Math.round((Math.random()*750));
  //Задаем случайное значение и направление скоростей
  vx=Math.round((Math.random()*10))-5;
  vy=Math.round((Math.random()*10))-5;
  move();//Запуск игрового процесса
}
//Функция, реализующая перемещение квадрата
function move(){
  //Если флаг установлен, то новый экземпляр функции не нужен
  if (flag==true){return}
  //Вспомогательные переменные с координатами квадрата
  var x=lay.style.left;
  var y=lay.style.top;
  //Случайное изменение скоростей с 10% вероятностью
  if (Math.round((Math.random()*1000))<10){
    vx=Math.round((Math.random()*10))-5;
    vy=Math.round((Math.random()*10))-5;
  }
  //Изменение координат
  x=parseInt(x)+vx;
  y=parseInt(y)+vy;
  //Проверки на встречу с границей поля
  if (x<=0){
    x=0;
    vx=-vx;
```

```
      }
      if (x>=750){
        x=750;
        vx=-vx;
      }
      if (y<=0){
        y=0;
        vy=-vy;
      }
      if (y>=450){
        y=450;
        vy=-vy;
      }
      //Назначаем новые координаты квадрату
      lay.style.left=x;
      lay.style.top=y;
      //Вызов функции через 30 мс
      setTimeout("move();",30);
    }
    //Функция, реагирующая на попадание
    function vis(){
      //Увеличиваем счетчик попаданий
      document.forms[0].Yes.value++;
      //Устанавливаем флаг, чтобы не обрабатывалась функция move()
      flag=true;
      //Меняем цвет квадрата
      lay.style.backgroundColor="red";
      //Через 400 мс вновь запускаем игру
      setTimeout("init();",400);
    }
  </script>
</head>
<body onLoad=init()>
  <!-- Игровое поле -->
  <div id=field style="position:absolute; top:0px;left:0px;
```

```
      height:500px;width:800px; background-color:yellow;"
      onClick='javasript:document.forms[0].No.value++;'>
    </div>
    <!-- Слой с квадратом -->
    <div id=lay style="position:absolute; top:50px;left:50px;
      height:50px;width:50px; background-color:orange;" onClick='vis()'>
    </div>
    <!-- Поле для счета -->
    <div id=control style="position:absolute; top:0px;left:800px;
      height:500px;width:200px; background-color:green;">
      <form>
        <h2>Счет</h2>
        Попал: <br>
        <input id="Yes"> <br>
        Мимо:<br><input id="No">
      </form>
    </div>
  </body>
</html>
```

При щелчке на квадрате увеличивается значение в текстовом поле Попал, при промахе — в текстовом поле Мимо (рис. 2.46).

Игровое поле размером 800 × 500 пикселов желтого цвета занимает основную часть страницы. При щелчке на нем увеличивается количество промахов. Справа от него находится поле счета, в котором располагаются два текстовых поля. Квадрат, который перемещается по игровому полу, является слоем размером 50 × 50 пикселов. После загрузки страницы запускается функция `init()`, которая задает начальные значения. С учетом размера квадрата координаты его левой стороны могут принимать значения от 0 до 750, а верхней — от 0 до 450. Скорость по горизонтали и вертикали может изменяться от –5 до 5. После задания случайных начальных данных управление передается функции `move()`. В этой функции осуществляется вечное движение квадрата по инерции. Для увеличения непредсказуемости с вероятностью около 10 % меняются скорости перемещения квадрата. Через каждые 30 мс функция `move()` вызывает новый экземпляр самой себя. При щелчке на квадрате обработчик события передает управление функции `vis()`, в которой увеличивается счетчик попаданий и изменяется цвет квадрата. Далее, через 400 мс, управление вновь передается функции `init()`.

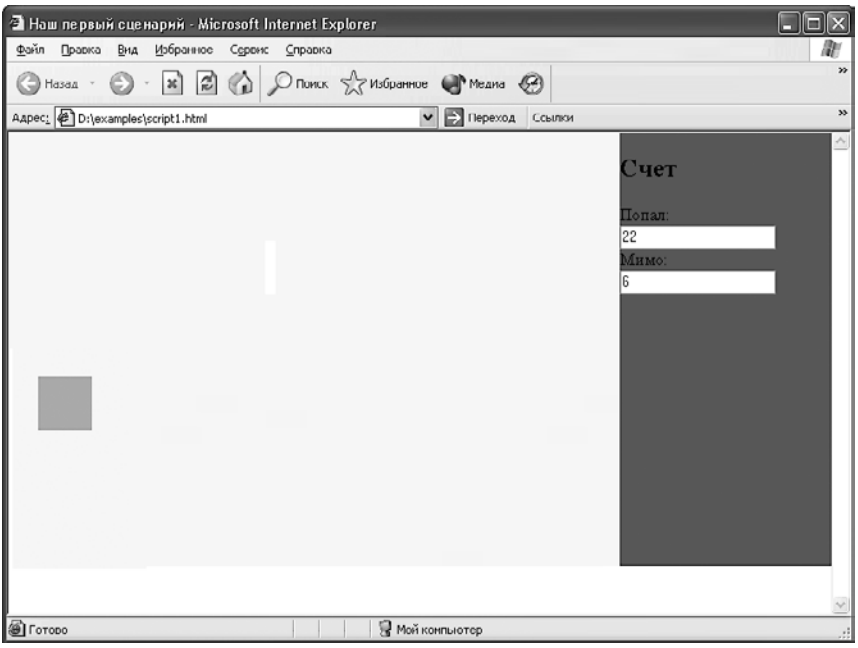


Рис. 2.46. Игра Мишень

Конечно, эту игру можно улучшить. Например, вместо кубика ввести изображение шарика, защитить текстовые поля от изменений, добавить динамику в передвижение мишени и т. д.

Резюме

В этой главе вы познакомились с возможностями языка сценариев JavaScript. С его помощью можно существенно разнообразить дизайн страниц, упростить навигацию для пользователя и сделать веб-сайт более привлекательным.

Отдельно мы рассмотрели динамические страницы. Применение динамического HTML дает возможность управлять содержимым страницы при ее загрузке и работе с ней. Этот весьма мощный инструмент позволит веб-дизайнеру воплотить в жизнь свои разнообразнейшие идеи.

Зная HTML и JavaScript, вы сможете создавать полноценные веб-сайты.

Глава 3

Apache, Perl и MySQL

Сервер Apache

Язык Perl

СУБД MySQL

Резюме

Данная глава посвящается использованию веб-сервера Apache и базы данных MySQL в веб-приложениях, написанных на языках Perl. Я расскажу вам о том, как установить все необходимые программные продукты и начать их использовать.

Сервер Apache

Одной из наиболее распространенных платформ для запуска веб-серверов является Apache. Свое распространение он получил, во-первых, за счет высокого качества, а во-вторых, благодаря своей бесплатности. Более того, исходный код Apache открыт и доступен всем, поэтому при необходимости в него можно добавлять различные модули, а также настраивать взаимодействие с тем или иным приложением.

Apache работает не как пользовательское приложение типа текстового редактора или браузера, которые имеют главное окно, меню и т. д. Он функционирует целиком в фоновом режиме, а его сервисами пользуются другие программы и приложения — браузеры, базы данных и интерпретаторы Perl.

Для чего нужен Apache

Apache обеспечивает работу веб-серверов. Он получает запросы от браузеров и в соответствии со своими настройками выдает им ответ — как правило, HTML-страницу или же другие данные.

Когда вы набираете в своем браузере какой-нибудь адрес, например `http://www.google.ru`, то ваш компьютер устанавливает соединение с веб-сервером, который, обрабатывая ваш запрос, выдает HTML-страницу.

Таким образом, можно сформулировать задачи Apache (а также всех веб-серверов).

- Получение запросов от браузеров через Интернет. Для передачи таких запросов существуют специальные сетевые протоколы — http, https и др.
- Анализ запросов. На этом этапе веб-сервер должен определить, какие подсистемы задействованы и какие данные поступили от пользователя.
- Подготовка данных (в других приложениях) для передачи пользователю. В простейшем случае, когда веб-сервер работает со статическими сайтами, пользователю передается готовая html-страница, уже лежащая где-то на сервере. Для обработки динамических страниц вызываются различные подсистемы — PHP, Perl, MySQL и др.

- Передача подготовленных данных пользователю. Программа пользователя, как правило, браузер, обеспечивает представление переданных сведений пользователю.
- Обеспечение безопасности работы сервера. Так или иначе веб-сервер предоставляет пользователям доступ к данным на нем, поэтому он должен иметь хорошую защиту от злоумышленников.

Мы рассмотрели работу серверов в общем случае.

Веб-сервер Apache работает в сети, обеспечивая доступ к данным с других компьютеров. Под сетью в данном случае может подразумеваться как локальная сеть, так и Интернет. В частном случае обращение браузера к веб-серверу необязательно происходит через Интернет. Если вы установите Apache к себе на машину, то сможете превратить свой компьютер в веб-сервер. Это позволит реализовать две задачи: обеспечить функционирование сервера на своей машине (что может использоваться для отладки) и гарантировать доступ к данным на вашем компьютере для других пользователей.

Таким образом, Apache позволяет организовать полноценный веб-сервер на любом компьютере, подключенном к Интернету. Правда, не все так просто, и одного Apache для этого недостаточно: нужно обеспечить доступ к вашему компьютеру из Интернета. Для этого вам потребуются выделенный IP-адрес и запись в сервере DNS. Такая запись нужна для того, чтобы пользователи имели возможность обращаться к вашему серверу по имени, а не по цифрам IP-адреса. Однако на этом мы останавливаться не будем.

Apache реализует взаимодействие с клиентами по так называемому *протоколу передачи гипертекста* — HTTP.

Установка Apache

Чтобы начать работу с Apache, его нужно скачать из Интернета, установить и настроить. Увы, это не так просто, как кажется.

Как уже говорилось, веб-сервер Apache является абсолютно бесплатным. Его можно скачать, например, с сайта http://httpd.apache.org. Он имеет немного запутанную систему скачивания: нужно выбрать источник для скачивания — один из зеркальных сайтов — зеркало для скачивания, а затем один из следующих файлов:

- `UNIX Source: httpd-2.2.9.tar.gz [PGP] [MD5];`
- `Win32 Source: httpd-2.2.9-win32-src.zip [PGP] [MD5];`
- `Win32 Binary without crypto (no mod_ssl) (MSI Installer);`

- `Win32 Binary including OpenSSL 0.9.8h (MSI Installer)`;
- другие файлы.

Нас интересует простая версия для Windows (третья сверху). Она представляет собой стандартный инсталлятор, который, задав несколько типичных вопросов, установит Apache на ваш компьютер.

Итак, приступим.

1. Запустим мастер установки (Setup Wizard) — файл `apache_2.2.9-win32-x86-no_ssl-r2.msi`, который мы скачали с сервера.
2. В начальном окне мастера установки нажмем кнопку Next (Далее).
3. После этого необходимо принять лицензионное соглашение. Здесь следует согласиться с утверждением (I accept the terms in the license agreement) и нажать кнопку Next.
4. Затем нужно указать режим работы Apache (рис. 3.1): как сервис (служба) или как приложение. Первый вариант (as a Service) предпочтительнее.

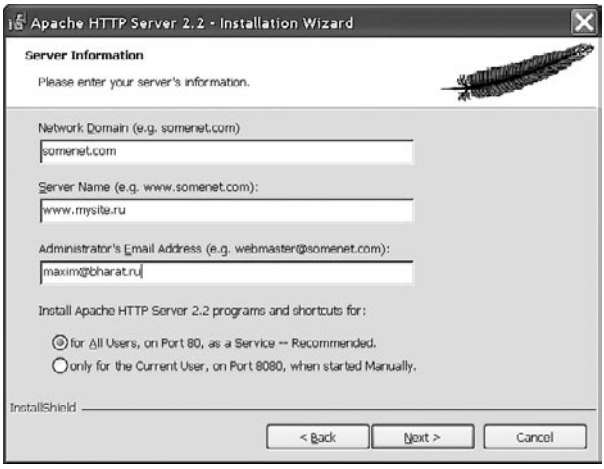


Рис. 3.1. Выбор режима работы

5. Выберем один из следующих типов установки :
   - `Typical` (Обычная) — установятся все необходимые компоненты Apache;
   - `Custom` (Выборочная) — вы сможете указать путь к каталогу, в котором установится Apache, и компоненты, требуемые для инсталляции. Кстати, в типе `Typical` тоже можно выбрать директорию для установки (в следующем окне).

Теперь нужно настроить Apache.

Следует отметить, что после установки Apache вы не заметите какого-либо результата, так как этот сервер не является приложением с пользовательским интерфейсом, оконными формами и красивыми заставками. Установка сервера просто завершится — и все (рис. 3.2).

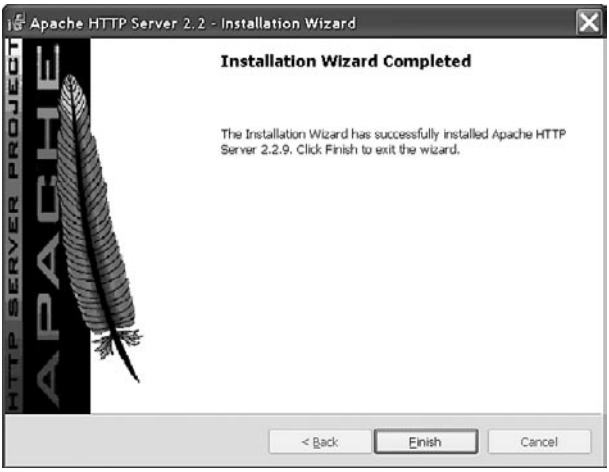


Рис. 3.2. Установка завершена

Убедиться в том, что сервер работает, очень просто: достаточно запустить браузер (к примеру, Internet Explorer) и написать адрес вашего компьютера — как правило, это 127.0.0.1 и 192.168.0.1 (можно также ввести адрес http://localhost/).

Если Apache установился успешно, то в окне браузера появится сообщение `It works!` (рис. 3.3).



Рис. 3.3. Установка Apache завершилась успешно

Теперь вам предстоит самое главное — сконфигурировать и настроить Apache.

Основные настройки Apache

Настройка Apache, пожалуй, немного неудобна — для нее не предусматривается никаких специальных графических средств. Для изменения каких-либо параметров работы Apache необходимо править текстовые файлы, содержащие настройки. Однако не отчаивайтесь: как правило, нет нужды регулярно менять файлы, да и не так это сложно, как может показаться.

Основные настройки Apache хранятся в файле `httpd.conf`. Обычно он находится в директории `Program Files\Apache Software Foundation\Apache2.2\conf`, его можно вызвать из меню Пуск.

Откройте файл `httpd.conf` в редакторе Notepad (рис. 3.4).

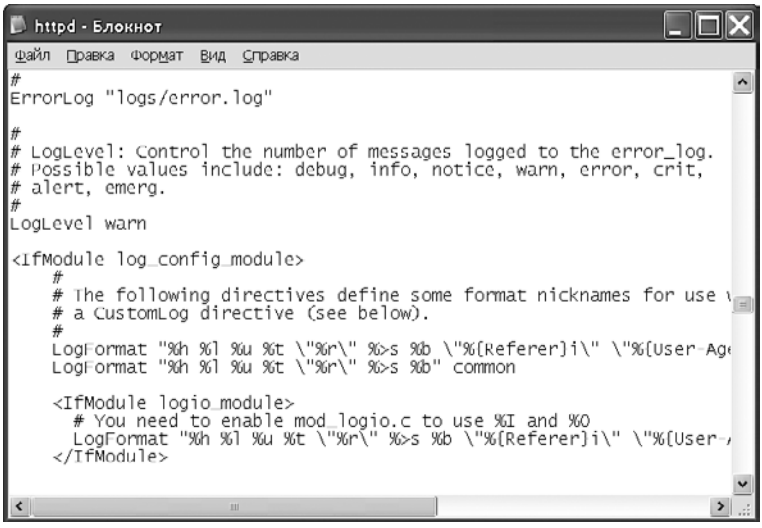


Рис. 3.4. Файл httpd.conf

Как вы видите, некоторые строки здесь начинаются с символа #. Он означает, что данная строка будет игнорироваться Apache (это называется комментарием). Рядом с этим символом разработчики Apache написали свои комментарии и инструкции конфигурационного файла, которые в данный момент выключены.

Чтобы изменения в этом файле вступили в силу, необходимо перезапустить Apache. Как я рекомендовал, Apache лучше устанавливать в режиме службы, так как он всегда будет работать в фоновом режиме. Это можно сделать либо из панели управления («Администрирование/Службы»), либо с помощью сервисного окна — оно открывается при нажатии значка Apache около часов. В этом окне (рис. 3.5) нужно выбрать команду `Restart`.

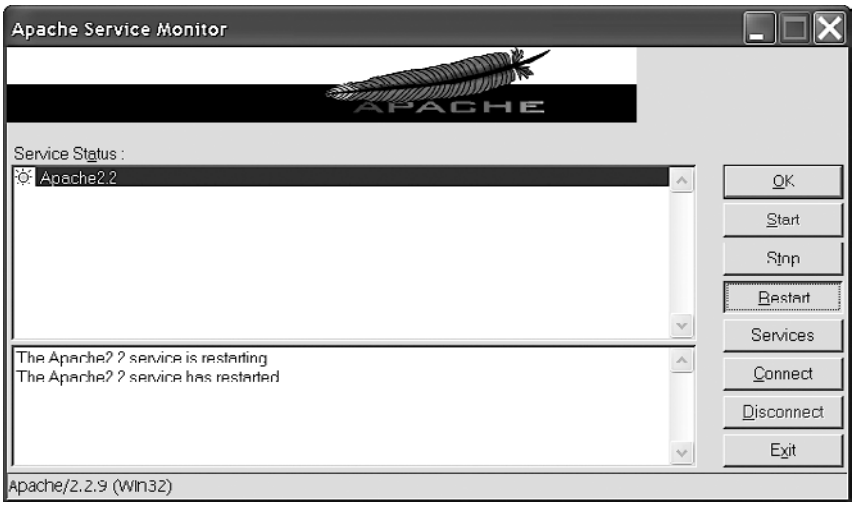


Рис. 3.5. Окно службы Apache

Какие же основные команды (директивы) нас интересуют?

В первую очередь это директивы основного сервера, которые указывают Apache, какие адреса следует обрабатывать. По умолчанию это адрес компьютера — 127.0.0.1 по порту 80, однако это может быть и любой другой IP-адрес.

Для этого существует директива `Listen`:

```
Listen 12.34.56.78:80
```

Или же просто

```
Listen 80
```

В первом случае директива указывает IP-адрес и порт, по которому будет работать Apache, а во втором — только порт (используется IP-адрес компьютера). Чтобы ваш сервер был доступен не только с вашего компьютера, здесь нужно указать адрес, под которым он будет в распоряжении пользователей «снаружи».

Следующая команда — это `ServerRoot`. Она указывает, где на самом деле будут находиться файлы вашего виртуального сервера. По умолчанию она выглядит приблизительно так:

```
ServerRoot "D:/Program Files/Apache Software Foundation/Apache2.2"
```

Ее можно переписать таким образом:

```
ServerRoot "D:/МувебSite"
```

Означает это примерно следующее: при обращении браузера к адресу `http://12.34.56.78/file.html` Apache начнет поиск файла `file.html` по адресу `D:\МувебSite`.

Если вы посмотрите адрес, указанный при установке Apache, то обнаружите, что там находится файл `index.html`, в котором написано следующее:

```
<html><body><h1>It works!</h1></body></html>
```

Именно эти слова — `It Works!` — вы видели в начале работы Apache.

Apache позволяет работать сразу с несколькими такими виртуальными серверами. Для этого существует директива `VirtualHost`, которая в общем случае выглядит так:

```
<VirtualHost 127.0.0.1>
    ServerAdmin вебmaster@dummy-host.example.com
    DocumentRoot "d:/mysite/1"
    ServerName dummy-host.example.com
    CustomLog dummy-host.example.com-access_log common
</VirtualHost>
```

Самые важные параметры здесь — это IP-адрес, порт (не может быть двух серверов, находящихся на одном и том же IP и порте, поэтому адреса нужно задавать разные, к примеру 127.0.0.2, 127.0.0.3...) и уже упоминавшийся `DocumentRoot`.

Адреса вида 127.0.0.* зарезервированы для локальных данных компьютера, их без проблем можно использовать. Если же вы напишете там произвольный адрес, например 213.180.204.11, то, возможно, вам понадобится этот адрес. В частности, по указанному адресу находится сервер Яндекса. При обращении к нему из браузера управление перехватит Apache и представит вам не содержимое сайта Яндекса, а ваш собственный сервер.

Вам наверняка пригодятся директивы ErrorLog и CustomLog. Директива Errorlog указывает на файл, который содержит запись обо всех ошибках, возникающих при обращении к вашим виртуальным серверам, а CustomLog — просто информацию об обращениях к серверам. Первый файл полезен при отладке веб-сервера, а второй — при анализе истории посещений.

Следует отметить, что если вам неудобно набирать в адресной строке длинные адреса из цифр, то можете отредактировать файл `hosts` из директории Windows и назначить имена своим виртуальным серверам.

Последняя директива, которую мы рассмотрим, — это `LoadModule`. Она выглядит так:

```
LoadModule alias_module modules/mod_name.so
```

Эта директива позволяет загружать в Apache дополнительный модуль.

Язык Perl

Язык программирования Perl — это один из популярных языков программирования, используемый в Интернете. Сегодня Perl является неотъемлемым инструментом при разработке веб-серверов наравне с PHP и Java.

Этот язык программирования появился еще в 80-х гг. XX в., когда о WWW и не слышали. Perl отвечает всем требованиям, которые предъявляются к языку программирования при разработке веб-приложений: он одновременно простой и эффективный, к тому же обладает богатыми возможностями для работы с текстом.

В чем же отличие Perl от языков программирования JavaScript и VBScript, которые прочно связаны с разработкой сайтов? Эти языки сценариев выполняются браузером. Они могут взаимодействовать с пользователем, который открыл страницу. А вот сценарии Perl выполняются сервером, то есть на выходе они предоставляют какую-то информацию, которая и отправляется пользователю. После отправки данных сценарии прекращают свою работу.

Таким образом, браузер не может «догадаться», откуда у него появилась веб-страница: то ли это просто файл с сервера, то ли динамически сгенерированные каким-то сценарием данные.

Установка Perl и подключение его к Apache

Язык Perl можно скачать абсолютно бесплатно, например с сайта www.activestate.com. Как и всякий «современный» веб-сайт, он крайне запутан, однако пусть вас это не смущает — выбирайте пункт Downloads, а затем — ActivePerl. На вопросы же о ваших контактных данных можете смело не отвечать.

Вам нужен дистрибутив ActivePerl для Windows с расширением MSI, который является самораспаковывающимся.

Итак, приступим к установке.

Как всегда, в самом начале вы сможете узнать про сам ActivePerl (рис. 3.6).



Рис. 3.6. Начало установки

Затем вы имеете возможность выбрать отдельные компоненты для инсталляции. Я рекомендую выбрать их все, так как дополнительные компоненты занимают очень мало места (рис. 3.7).

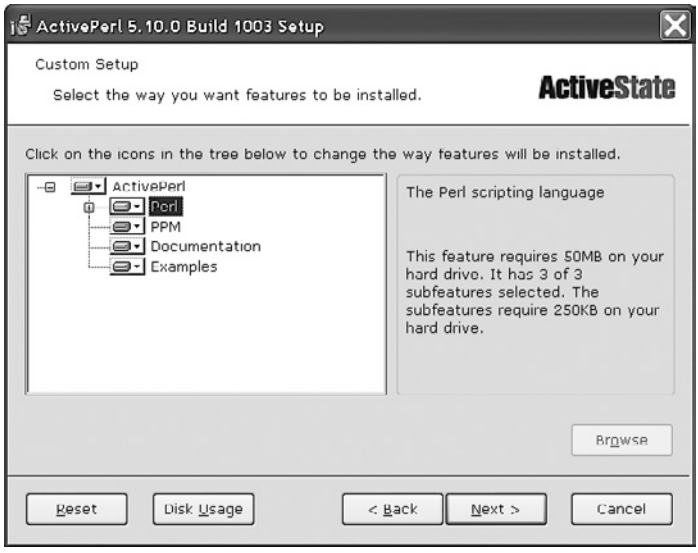


Рис. 3.7. Выбор компонентов

Теперь самое сложное: нужно связать Apache и Perl. Для этого откроем файл `httpd.conf` и добавим строку `Options FollowSymLinks ExecCGI`.

Найдем и раскомментируем строку `#AddHandler cgi-script .cgi`, убрав символ #. Вместо `.cgi` можно написать `.pl`, так как эти расширения используются для языка Perl в равной мере.

Добавим следующую директиву:

```
<FilesMatch "\.cgi$">
Options ExecCGI
</FilesMatch>
```

Эти изменения позволяют Apache определить, когда следует обращаться к Perl.

После изменения файла `httpd.conf` нужно перезапустить сервер Apache.

Теперь можно проверить работу первого сценария. Для этого создадим файл `printenv.cgi` в той же директории, которая указывается в директиве `DocumentRoot` в файле `httpf.conf`, и поместим в него следующее:

```
#!c:/Perl/bin/perl.exe
print "Content-type: text/plain; charset=iso-8859-1\n\n";
foreach $var (sort(keys(%ENV))) {
    $val = $ENV{$var};
    $val =~ s|\n|\\n|g;
    $val =~ s|"|\\"|g;
    print "${var}=\"${val}\"\n";
}
```

В Perl, так же как и в файле `httpd.conf`, комментарии отмечаются знаком решетки, однако первая строка должна содержать адрес установленного perl-интерпретатора.

При установке программа не спрашивает, куда записать файлы: она записывает их на тот диск, на котором находится дистрибутив Windows.

Проверим работу нашей небольшой программы. Для этого запустим браузер и откроем там страницу http://192.168.0.1/printenv.cgi.

Если все в порядке, то вы увидите данные о текущих переменных (рис. 3.8).

В противном случае это означает, что вы выполнили что-то не так.

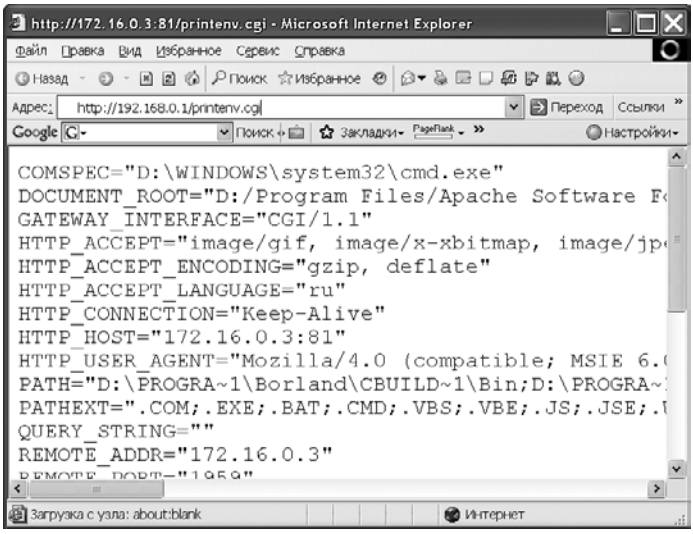


Рис. 3.8. Результат работы сценария

Проблем может быть несколько: во-первых, вы могли просто забыть перезапустить веб-сервер, а во-вторых, создать файл не там, указать не тот путь или забыть что-то еще. Чтобы правильно диагностировать проблему, нужно посмотреть лог-файлы Apache (они находятся в директории `logs` — там же, где и сам Apache).

Нас интересуют там два файла — `access.log` и `error.log`. В первом мы можем узнать характер ошибки. Например, строка

```
192.168.0.1 - - [10/Jul/2008:13:06:38 +0400] "GET /printenv.cgi HTTP/1.1"
404 209
```

говорит о коде ошибки 404 — то есть файл `printenv.cgi` просто отсутствует: вы его записали в какую-то другую директорию.

Строка

```
192.168.0.1 - - [10/Jul/2008:13:07:25 +0400] "GET /printenv.cgi HTTP/1.1"
500 535
```

с кодом ошибки 500 говорит о том, что ошибка — в конфигурации.

Ну и, наконец, строка

```
192.168.0.1 - - [10/Jul/2008:13:07:39 +0400] "GET /printenv.cgi HTTP/1.1"
200 1410
```

с кодом 200 свидетельствует о том, что обращение к файлу прошло успешно.

При возникновении ошибки нужно изучить файл `error.log`. Ошибка

```
[Thu Jul 10 13:06:38 2008] [error] [client 172.16.0.3] D:/Program Files/
Apache Software Foundation/Apache2.2/htdocs/config.cgi is not executable;
ensure interpreted scripts have "#!" first line
```

говорит о том, что вы забыли указать местоположение интерпретатора Perl.

Если файлы слишком большие и в них сложно разобраться, то их нужно стереть, предварительно остановив Apache. После повторного запуска Apache и обращения к вашей странице они создадутся заново и в них поместятся только самые последние ошибки.

Основы языка и синтаксис

Рассмотрим базовые принципы Perl, не вдаваясь в детали.

Как известно, Perl — это язык-интерпретатор. Это обеспечивает его кросс-платформенность: программа, написанная на данном языке, будет работать в любой операционной системе, в которой есть программа-интерпретатор (в нашем случае — ActivePerl). Поэтому мы можем смело разрабатывать на домашнем компьютере программы, тестируя их под Windows, чтобы позже положить их на сервер хостинг-провайдера. Как правило, хостинг-провайдеры работают под управлением ОС UNIX.

Язык Perl не является строго типизированным, как, например, Java или Си. Это означает, что не обязательно объявлять переменную перед ее использованием. Вполне допускается следующий код:

```
$a = 'Привет';
$a = 12;
print $a;
```

Здесь переменная `a` (переменные всегда предваряет знак $, а массивы — знак @) сначала принимает значение строкового типа, а затем — числового. В более «взрослых» языках программирования такое не допускается, потому что это может вызвать множество ошибок. Однако программы на Perl не такие большие, чтобы нельзя было уследить за подобного рода ошибками.

Команда `print` выводит значение своего аргумента (в данном случае — число 12). Каждое выражение должно заканчиваться точкой с запятой.

Строки в Perl заключаются в кавычки, тип которых крайне важен. Одиночные кавычки означают просто строку текста, а двойные кавычки допускают использование

специальных символов и переменных. Например, код `print '$a\n'` выведет строку `$a\n`. А вот код print `"$a\n"` выведет значение переменной `a` и символ перевода строки. Нередко неправильное использование кавычек является источником ошибок, отыскать которые не так легко.

Важная часть любого языка программирования — блоки и циклы.

Как и любой современный алгоритмический язык программирования, Perl поддерживает циклы — для этого предназначены операторы `for`, `while` и `do`. Они аналогичны циклам, реализуемым в языке Си: тело цикла, то есть код, который должен выполняться циклически, заключается в фигурные скобки:

```
for($i = 0; $i< 100;$i++)
{ print $i; }
```

Данный цикл присваивает переменной `i` значения от 0 до 99 и выводит их на экран в одну строку: `012345678910111213141 5...`

Немного по-другому, чем в Си, выглядит оператор `while`:

```
$i=10;
while ($i > 0)
{
print $i;
} continue {$i-}
```

В результате выполнения этого кода числа будут отображаться в обратном порядке, начиная с 10.

Отметим, что здесь, в отличие от языка Си, фигурные скобки нужно использовать обязательно. Кроме того, Perl не содержит оператора `break`, а оператор `continue` выполняет совершенно иную функцию. В Perl применяются следующие функции для обеспечения работы циклов:

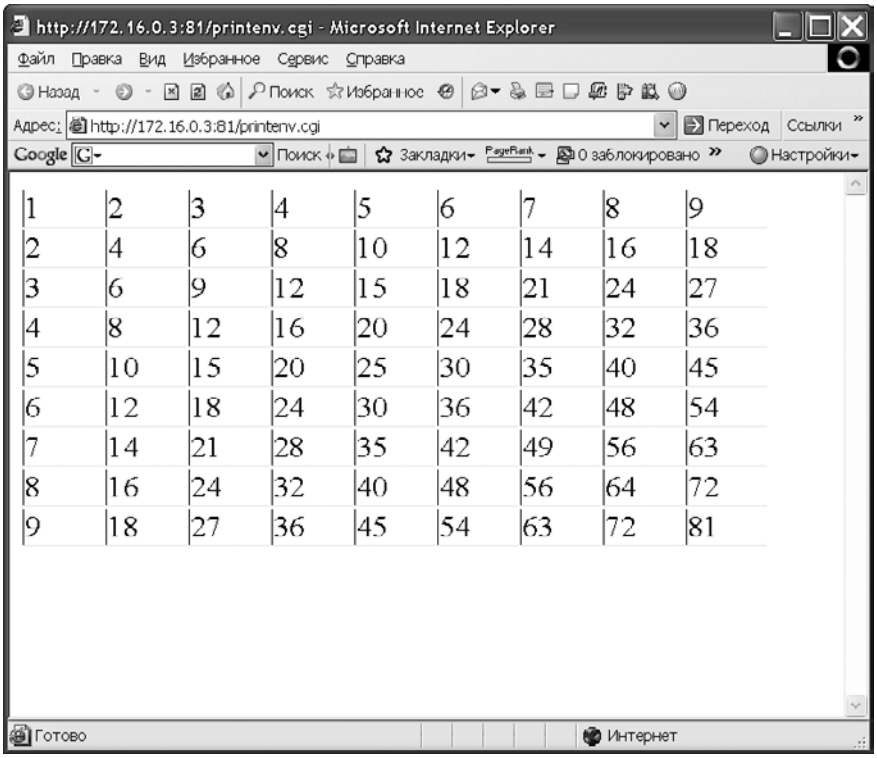
- `Last` — досрочный выход из цикла;
- `Next` — начало новой итерации;
- `Redo` — повтор текущей итерации.

Ниже приводится первый Perl-сценарий, который выводит на экран не числа, а HTML-код.

```
#!c:/Perl/bin/perl.exe
print "Content-type: text/plain; charset=iso-8859-1\n\n";
```

```
print "<html><body><table>";
for($i = 1; $i< 10;$i++)
{
print "<tr>";
for($j = 1; $j< 10;$j++)
{
 print "<td width=70 style='border-left:1px solid #707070;border-bottom:1px
solid #f0f0f0;'>";
 print $i*$j, "</td>";
}
print "</tr>";
}
print "</table></body></html>";
```

В результате работы этого кода на экран будет выведена таблица умножения (рис. 3.9).



1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Рис. 3.9. Таблица умножения

Если вы посмотрите исходный код страницы, то увидите, что там вместо вашей программы находится полноценный HTML-код. Именно таким образом Perl и может интегрироваться в сайт: программы формируют часть кода или код целиком. Это позволяет менять содержимое страниц в зависимости от любых факторов, которые можно запрограммировать.

Рассмотрим цели, для которых чаще всего используется язык Perl.

- Использование шаблонов. При создании сайта в простом, статическом HTML мы создаем страницы непосредственно в коде. Когда нужно задать иной дизайн сайта, следует изменить код всех страниц. С помощью Perl мы можем создать только одну страницу — программу, которая будет подставлять необходимое содержание, взяв его, например, из базы данных. Таким образом, дизайн будет храниться отдельно от содержания сайта.
- Вывод динамической информации. С помощью Perl содержимое страницы можно адаптировать под каждого конкретного пользователя — выводить список его заказов на сайте, отображать интересующую этого пользователя информацию (последние просмотренные статьи, новые сообщения и т. д.).
- Вывод расчетной информации, когда на момент создания сайта мы не знаем точно, что будет отображаться в том или ином месте, однако знаем, где брать информацию для отображения. Таким образом отображаются, например, новости других сайтов, информация о курсе валют и погоде. Все это можно запрограммировать.
- Взаимодействие с пользователем и обработка форм.

Однако рассмотренным выше методом вводить HTML-код очень неудобно: приходится каждый раз при выводе тегов писать команду `print`. В Perl есть следующая конструкция, позволяющая приводить код напрямую:

```
print <<HTML;
<HTML>
<HEAD><TITLE>Страничка</TITLE></HEAD>
<BODY>
<H1><CENTER>Привет всем! </CENTER></H1>
HTML
Print    "</BODY></HTML>";
```

Здесь между командами `print <<HTML` и `HTML` идет непосредственно html-код, который будет передаваться в браузер.

Примеры использования Perl для взаимодействия с пользователем

Язык Perl чаще всего применяется для организации форм ввода информации.

Формы — это специальный механизм, заложенный в HTML и предназначенный для получения какой-либо информации от пользователя. Разнообразные действия, начиная от банальной галочки «не открывать в новом окне» и заканчивая заполнением опросного листа для оформления какого-нибудь заказа, выполняются с помощью форм.

Сами формы реализуются средствами HTML, и Perl здесь может использоваться только в целях генерации этого HTML. Код для задания формы может выглядеть так:

```
<form method=post action="form.cgi">
<b>Введите имя:</b>
<input name="user_name" size=20>
<b>и фамилию:</b> <input name="user_fam" size=20>
<input type="submit" value="OK">
</form>
```

Данный код отображает форму из двух текстовых полей и кнопки OK, при нажатии которой информация передается в сценарий `form.cgi` (рис. 3.10).

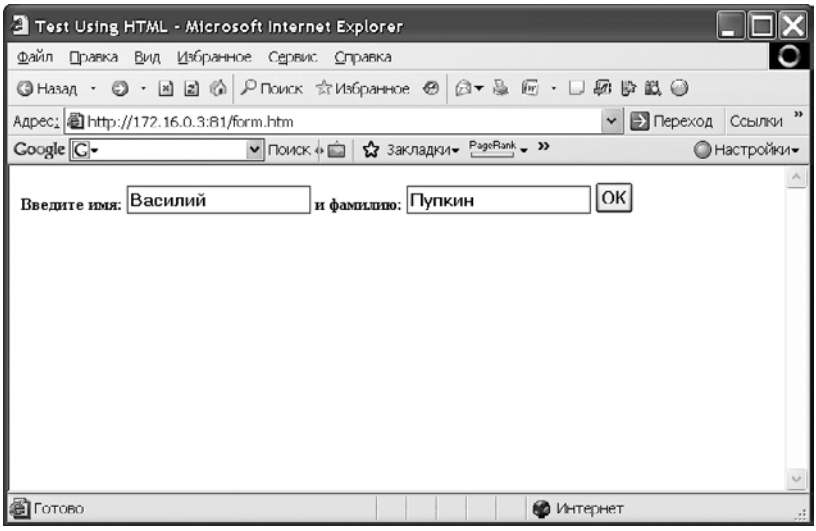


Рис. 3.10. Простейшая форма

Присланная формой информация должна быть обработана. Для этого и используется Perl, так как язык HTML не дает этого сделать. Сценарии на Perl позволяют выполнить с данными формы следующие действия:

- записать данные в текстовый файл или базу данных;
- отослать данные по электронной почте;
- принять их к сведению, то есть отложить использование данных, например при выборе товара в интернет-магазине вы также используете формы, однако содержимое корзины «запоминается» до тех пор, пока вы не завершите выбор и не оформите заказ;
- использовать их на следующей странице («Благодарим за заказ!» или «Указан неправильный e-mail») и многое другое.

Для обработки введенных в форму данных нужно использовать имена полей — в нашем случае это `user_name` и `user_fam`, указанные в тегах `input`.

Для их обработки в сценарии `form.cgi` (именно это имя указывается в HTML-коде в теге `FORM`) нужно разместить следующий код:

```
#!c:/Perl/bin/perl.exe
print "Content-type: text/plain; charset=iso-8859-1\n\n";
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
@pairs = split('&', $buffer);
foreach $pair (@pairs) {
($name, $value) = split(/=/, $pair);
$value =~ tr/+/ /;
$value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
$value =~ s/<!--(.|\n)*-->//g;
$input{$name} = $value;
print "$name : $value<br>";
}
print "<h3>",$input{'user_name'},", спасибо, что заполнили форму</h3>";
print 'Нажмите <a href="/proceed.html">сюда</a>, чтобы увидеть Ваш ввод.';
```

В данном коде сценарий записывает в переменную `$buffer` значения переменных, перенесенных в Perl стандартным образом (это делает команда `read`). Переменные передаются следующим образом: `имя1=значение1&имя2=значение2&...` Нам нужно из этой строки вычленить отдельные переменные. Это делается командой

split, которая разделяет строку на элементы массива и помещает результат в массив @pairs. При вызове этой функции указывается символ-разделитель (в нашем случае — &).

Команда цикла foreach обрабатывает каждую пару «переменная = значение» в образовавшемся массиве @pairs. В начале цикла пара разбивается на две переменные — $name и $value. Затем переменная $value из одного представления преобразовывается в другое. Дело в том, что при передаче данных в протоколе HTTP используется специальное кодирование, в результате которого имя, например «Василий», преобразовывается в так называемое 7-битное представление: %C2%E0%F1%E8%EB%E8%E9.

Предпоследняя строка цикла — $input{$name} = $value — заполняет переданными значениями переменную input.

Результат обработки сценария вы видите на рис. 3.11.

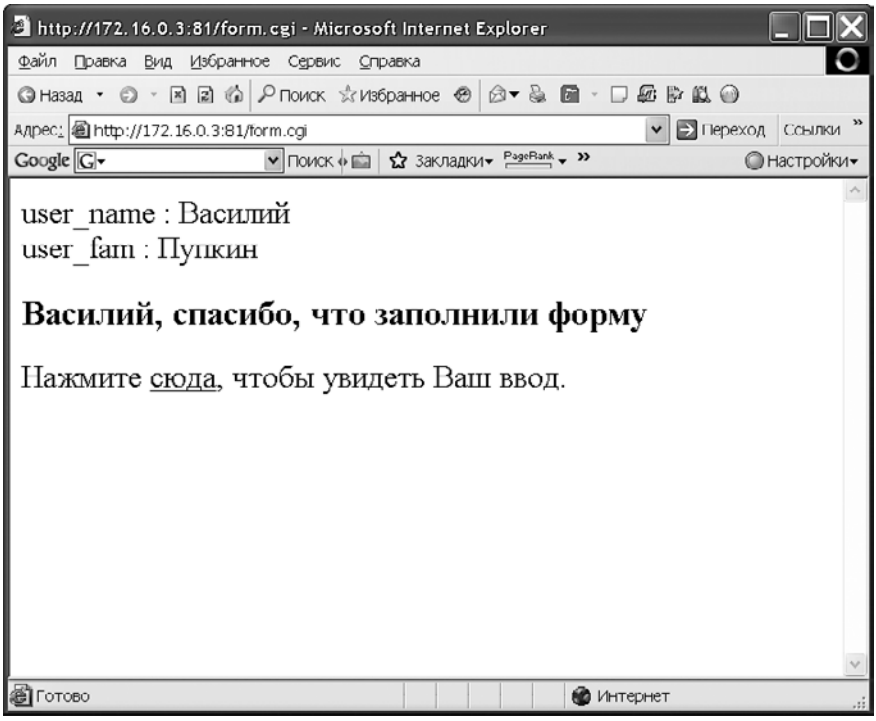


Рис. 3.11. Результат обработки простейшей формы

Мы рассмотрели, как данные из формы могут попасть на следующую страницу. Теперь разберем механизм записи данных в файл.

С файлами в Perl работать очень просто: откройте файл, запишите в него информацию, а потом закройте:

```
open (HANDLE, ">>myfile");
print HANDLE "Имя: ", $input{'user_name'},"\n";
close (HANDLE);
```

Эти три команды открывают файл `myfile` и дописывают в его конец строку «Имя: Василий», после чего добавляют символ перехода на новую строку. Перед тем как записать в файл данные из формы, их нужно обработать указанным выше способом.

Чтобы прочитать весь файл (и, к примеру, отобразить его на странице), следует воспользоваться следующим кодом:

```
open (HANDLE, "myfile");
while($line=<HANDLE>)
{
 print $line;
}
```

Теперь рассмотрим процедуру отправки данных по электронной почте.

Часто требуется отослать данные, введенные в форму. Например, при оформлении заказа нужно отправить два сообщения: одно — менеджеру интернет-магазина, чтобы он сформировал заказ и созвонился с заказчиком, а второе — заказчику, чтобы тот получил подтверждение о заказе (перечень, номер заказа и т. д.).

Методов отправки сообщений из Perl несколько, мы рассмотрим лишь самый простой — другие вы сможете найти в Интернете. Для отправки почты простым способом требуется программа посылки почты с интерфейсом командной строки — обычно это программа Sendmail (ее можно найти в Интернете). Код для отправки сообщений следующий:

```
$sendto = 'manager@mysite.ru';
open (MAIL, "| /usr/bin/sendmail $sendto")
print MAIL "From: веб-сервер <zakaz@mysite.ru>\n";
print MAIL "To: $sendto\n";
print MAIL "Subject: Новый заказ";
print MAIL "Нам поступил новый заказ:";
print MAIL "Заказчик: ", $input{'user_name'},"\n";
print MAIL "Телефон: ", $input{'phone'},"\n";
close (MAIL);
```

Здесь используется переменная `$input`, которая должна содержать корректные значения из формы заказа. Как обрабатывать эти значения, мы рассматривали ранее.

Механизм поиска

Язык Perl был создан для упрощения работы со строками. Для операций в Интернете он стал применяться позже и лишь потому, что ничего лучше на тот момент не было — ни Java, ни PHP. Поэтому Perl и примечателен работой со строками — поиском и заменой.

Регулярные выражения (от англ. regular expressions) — это современная система поиска текстовых фрагментов, основанная на специальной системе записи образцов для поиска. Это ключевое понятие при работе со строками. Образец, задающий правило поиска, иногда называют *шаблоном*, или *маской*.

Регулярные выражения являются мощным и гибким языком описаний для поиска строк по шаблону. С их помощью можно выполнять приведенные ниже операции.

- Проверять, соответствует ли вся строка заданному шаблону. Например, определять корректность введенного электронного адреса или даты.
- Находить в строке подстроки, удовлетворяющие заданному шаблону. Например, проверять, содержит ли HTML-текст теги.
- Извлекать из строки подстроки, соответствующие заданному шаблону. Например, собрать все ссылки из HTML-текста в один массив.
- Изменять в строке подстроки, соответствующие шаблону. Например, заменять сразу все ссылки.

Работа с регулярными выражениями уже демонстрировалась выше, когда мы дешифровывали переданные из формы параметры. Рассмотрим, какими средствами обладает Perl для работы с этими выражениями.

Для начала сформируем шаблон. Он заключается в косые скобки и содержит определенным образом составленную строку. Например, шаблон `/бор/` будет отвечать таким строкам, как `борис`, `собор` и `бордовый`. Однако реальное преимущество шаблоны дают, когда в них используются так называемые *метасимволы* — специальные символы, интерпретируемые Perl в соответствии с некоторыми правилами. Ниже приводятся основные метасимволы.

- `^` — соответствует началу строки.
- `$` — соответствует концу строки.

- `.` — соответствует любому символу, кроме разрыва строки.
- `*` — соответствует повторению предыдущего символа нуль или более раз.
- `+` — соответствует повторению предыдущего символа один или более раз.
- `?` — соответствует повторению предыдущего символа нуль или один раз.
- `x|y` — соответствует *x* или *y*.
- `{n}`, где `n` — неотрицательное число. Соответствует ровно *n* вхождениям предыдущего символа.
- `{n,}`, где `n` — неотрицательное число. Соответствует *n* или более вхождениям предыдущего символа. Например, `/x{1,}/` эквивалентно `/x+/`, а `/x{0,}/` эквивалентно `/x*/`.
- `{n,m}`, где `n и m` — неотрицательные числа. Соответствует не менее чем *n* и не более чем *m* вхождениям предыдущего символа. Например, `/x{0,1}/` эквивалентно `/x?/`.
- `[xyz]` — соответствует любому символу из заключенных в квадратные скобки.
- `[^xyz]` — соответствует любому символу, кроме заключенных в квадратные скобки.
- `[a-z]` — соответствует любому символу в указанном диапазоне.
- `[^a-z]` — соответствует любому символу, кроме лежащих в указанном диапазоне.
- `\w` — соответствует латинской букве, цифре или подчеркиванию.

Например, шаблон `/^\w+([\.\w]+)*\w@\w((\.\w)*\w+)*\.\w{2,4}$/` определяет электронный адрес — строку, в которой есть символ @, нет пробелов и последнее выражение после точки содержит от 2 до 4 символов (например, .ru или .info).

Помимо шаблонов, существуют *модификаторы* — уточнения для шаблонов. Рассмотрим основные модификаторы.

- `I` — игнорирует регистр символов при сопоставлении с образцом.
- `M` — рассматривает исходную строку как буфер из нескольких строк текста, разделенных разрывами строк. Это означает, что метасимволы `^` и `$` соответствуют не только началу и концу всей строки, но и началу и концу строки текста, ограниченной разрывами строк.
- `S` — рассматривает исходную строку как единую строку текста, игнорируя разрывы строк. Это означает, что метасимвол `.` соответствует любому символу, включая разрыв строки.

- x — разрешает использование пробелов и комментариев. Пробелы, не имеющие предшествующего символа \ и не заключенные в квадратные скобки, игнорируются. Символ # начинает комментарий, который также игнорируется.

Простейший пример регулярных выражений следующий:

```
if($email=~ /^\w+([\.\w]+)*\w@\w((\.\w)*\w+)*\.\w{2,4}$/)
{ print "Адрес введен правильно";}
else
{ print "Адрес введен неправильно";}
```

Использование круглых скобок в регулярном выражении приводит к тому, что подстрока, соответствующая образцу в скобках, запоминается в специальном буфере и мы можем разбить строку на подстроки:

```
$_ = 'Время: 12:34:25';
if (/(\d\d):(\d\d):(\d\d)/) {
  $hours = $1;
  $minutes = $2;
  $seconds = $3;
}
```

Здесь также применяется специальная переменная $_ для хранения основной строки.

Работать с регулярными выражениями достаточно сложно, поэтому мы лишь кратко остановились на этой теме. Если требуются более сложные, чем указанные выше, коды, я рекомендую обратиться к специальной литературе.

Получение информации о посетителях

Мы рассмотрели передачу данных от пользователя через форму и анализ этих данных на корректность с помощью регулярных выражений. Однако один из самых главных вопросов остался пока нераскрытым.

Допустим, вы получили от пользователя данные о его заказе и записали их куда-нибудь. Но как узнать, что это все тот же пользователь, если он перешел на другую страницу? Ведь нельзя же просить его регистрироваться на каждой странице, а вместо ссылок ставить кнопки OK.

Для решения этой задачи — идентификации посетителей и передачи информации о них — создан специальный механизм под названием cookies.

Cookies — это данные в виде пар «ключ = значение», которые сервер выдает браузеру, а тот, в свою очередь, передает их обратно при открытии следующей страницы. В целях безопасности эти данные могут получаться в общем случае только тем сервером, который их выдал, или тем, для которого они предназначались выдающим сервером.

Обмен этой информацией между браузером и сервером может происходить следующим образом: в самом начале HTTP-ответа сервер устанавливает в браузер cookie с помощью поля заголовка Set-cookie. Если требуется установить несколько cookies, то в заголовке ответа должны быть соответственно несколько полей Set-cookie — отдельное для каждой пары cookie.

При следующих запросах этого пользователя браузер передает cookie, и они доступны через переменную среды `HTTP_COOKIE` в следующем виде:

```
NAME=VALUE; [expires=DATE;] [path=PATH;] [domain=DOMAIN_NAME;] [secure]
```

где `NAME` — это имя ключа, а `VALUE` — значение ключа. Остальные параметры являются вспомогательными. Веб-сервер, к которому они относятся, определяет эти параметры и срок хранения данных у пользователя.

Рассмотрим простейший пример:

```
#!d:/Perl/bin/perl.exe

print "Content-type: text/plain; charset=iso-8859-1\nSet-Cookie: cookie-name=test-value; path=/;\n\n";
print @ENV{'HTTP_COOKIE'};
```

Этот код устанавливает один cookie с именем `cookie-name` в значение `test-value` и выводит на экран все установленные cookie. Разобрать переменную `@ENV{'HTTP_COOKIE'}` можно так же, как мы разбирали переменную `@ENV{'CONTENT_LENGTH'}`).

Что же не следует хранить в cookie? Во-первых, никогда не храните в них пароли — они могут стать известны третьим лицам. Во-вторых, нельзя хранить информацию о факте регистрации, например «Зарегистрировано = да», потому что пользователь может подделать cookie и тем самым обойти процедуру регистрации.

В cookie лучше всего хранить произвольный код, про который сторонним людям ничего не может быть известно. В то же время ваша программа имеет возможность обращаться по этому коду к базе данных или временному файлу, который хранит информацию о данном пользователе. В следующем разделе мы рассмотрим, как именно Perl взаимодействует с базами данных.

СУБД MySQL

MySQL — это очень популярная и к тому же бесплатная система управления базами данных (СУБД), разработанная компанией MySQL AB.

Эта СУБД стала популярна в первую очередь благодаря тому, что содержит множество программных интерфейсов, которые позволяют подключаться к базе данных приложениям, созданным с помощью различных языков и средств разработки. Это кроссплатформенная система, то есть она может использоваться как под UNIX в качестве стандартного средства разработки веб-приложений, так и под Windows или, к примеру, Mac OS.

Ну и, конечно, потребительские качества этой системы находятся на высоком уровне: она быстро работает, прекрасно масштабируется и обеспечивает многопользовательский доступ. Она гарантирует безопасность данных за счет механизма разграничения доступа.

MySQL — это реляционная СУБД, то есть система управления *реляционными* базами данных. Для построения базы данных в MySQL необходимо знать основы теории реляционных баз данных.

Рассмотрим строение реляционной базы данных — таблицы, ключевые столбцы и связи между таблицами.

Введение в СУБД MySQL

Все данные, хранящиеся в реляционной базе данных, представляются в виде таблиц. Любая таблица имеет имя и состоит из строк и столбцов. На пересечении каждого столбца и строки располагается значение.

В качестве примера рассмотрим таблицу, содержащую сведения о посетителях сайта (табл. 3.1).

Таблица 3.1. Таблица Visitors (Посетители)

id	name (имя)	surname (фамилия)	phone (телефон)	address (адрес)
73	Вася	Пупкин	313-48-48	Vasya@pupkin.com
74	Гена	Пупкин	7(929)112-14-15	Gena@pupkin.com
75	Ваня	Иванов	—	Ivan_p@yandex.ru

Каждый столбец таблицы имеет имя и *тип данных* — этому типу данных соответствуют все значения в столбце. Так, в нашем примере столбец с именем

`id` — числовой, а поля `name` (адрес), `surname` (фамилия), `phone` (телефон) и `address` (адрес) — символьные.

Таблица данных представляет собой набор сведений об однотипных объектах. При этом каждая строка содержит информацию об одном объекте, а в каждом столбце содержатся значения некоторого атрибута этих объектов.

Важно понимать, что строки таблицы не упорядочены и могут храниться в базе данных в произвольной последовательности. У строк нет порядковых номеров. Для удобства работы с таблицами разработчики баз данных используют *первичный ключ* (primary key) — минимальный набор столбцов, совокупность значений которых однозначно определяет строку. В таблице не должно быть строк, у которых значения во всех столбцах первичного ключа совпадают. В нашем примере первичный ключ состоит только из одного столбца — `id`. Как правило, этого достаточно. Если бы мы сделали первичным ключом столбец `surname`, то таблица не могла бы содержать данных о двух Пупкиных, что, конечно, некорректно.

Первичные ключи применяются также для организации *связей* между таблицами. Связи позволяют объединить фрагменты информации в единую, целостную структуру данных, в которой строка одной таблицы сопоставляется строке (или нескольким строкам) другой таблицы. Это позволяет извлекать информацию одновременно из нескольких таблиц (например, выводить с помощью одного запроса и сведения о клиенте, и сведения о заказах этого клиента), избегать дублирования информации (например, не требуется в каждом заказе хранить адрес клиента) и многое другое.

Допустим, у нас имеются таблица *A* и таблица *B*, которые мы хотим связать. Для этого поместим в каждую строку таблицы *A* некую информацию, позволяющую идентифицировать связанную с ней строку таблицы *B*. Эта информация называется *ссылкой*, а поля таблицы *A*, содержащие эту ссылку, называются *внешними ключами*. В качестве ссылки используется первичный ключ таблицы *B*. Таблица *A* называется *дочерней*, а таблица *B* — *родительской*.

Связи бывают трех типов.

- Связь «один ко многим».

  Это наиболее распространенный тип связи. Каждая строка таблицы *A* ссылается на одну из строк таблицы *B*, причем на одну строку таблицы *B* могут ссылаться сразу несколько строк таблицы *A*. К примеру, в таблице *A* находятся статьи сайта, а в таблице *B* — названия разделов. Таким образом, в одном разделе может быть несколько статей.

Для установки связи между таблицами в дочернюю таблицу добавляется *внешний ключ* (foreign key) — столбец или набор столбцов, содержащий значения первичного ключа родительской таблицы (иными словами, во внешнем ключе хранятся *ссылки* на строки родительской таблицы).

- Связь «один к одному».

  Эта связь используется, когда каждой строке одной таблицы соответствует только одна строка другой таблицы, и наоборот. Это удобно, например, для хранения расширенных данных о клиентах: в одной таблице хранится краткая информация (например, имя и фамилия), а в другой — полная информация (номер паспорта и дата рождения).

  Таблицы, соединенные связью «один к одному», можно при необходимости преобразовать в одну. Однако использование двух и более таблиц вместо одной в некоторых случаях оправдывается, например из соображений конфиденциальности (когда некоторые данные недоступны пользователям без определенного уровня доступа) или удобства (если в таблице слишком много столбцов, что затрудняет работу с ней). Часто в отдельную таблицу выносят поля, которые во многих случаях остаются пустыми.

  Связь «один к одному» может организовываться так же, как связь «один ко многим», — с помощью первичного ключа родительской таблицы и внешнего ключа дочерней таблицы. Другой вариант — связь посредством первичных ключей обеих таблиц, при этом связанные строки имеют одинаковое значение первичного ключа.

- Связь «многие ко многим».

  Этот тип связи применяется, например, для того, чтобы составить таблицу заказов на сайте интернет-магазина. Каждый заказ должен содержать несколько товаров, причем каждый товар — в определенном количестве.

  Непосредственно задать такую связь в реляционной базе данных нельзя, поэтому она реализуется путем построения третьей таблицы. В нашем случае эта таблица должна иметь отдельные строки заказа, в каждой из которых содержатся наименование товара и количество, а также ссылка на номер заказа.

  Таким образом, таблица со строками заказа окажется дочерней по отношению и к таблицам заказов, и к таблицам продуктов магазина.

Но вообще реляционная база данных может иметь довольно запутанную и сложную структуру с различными видами связей. Поэтому неудивительно, что иногда возникают нарушения в структуре, вызванные либо ошибками операторов, либо неправильной работой программ. Это грозит нарушением базовых функций СУБД.

К примеру, если вы храните в базе сведения о заказах своих клиентов, то эти заказы могут перепутаться; если же материалы сайта, то они могут выдаваться с ошибками или не те, которые запрашивал пользователь.

В связи с этим в теории реляционных баз данных есть понятие *целостности данных*, которое означает корректность и непротиворечивость хранимых в СУБД сведений.

Базовыми требованиями целостности, которые должны выполняться в любой реляционной базе данных, являются *целостность сущностей* и *целостность связей* (ссылочная целостность).

Первое означает, что в каждой таблице есть первичный ключ, который не содержит повторяющихся и неопределенных значений. Второе — внешний ключ в дочерней таблице не содержит значений, отсутствующих в первичном ключе родительской таблицы. Другими словами, если в рассмотренном выше примере у нас есть таблица клиентов и таблица с их паспортными данными, то каждая строка из таблицы с паспортными данными обязательно ссылается на реально существующую строку из таблицы клиентов. Ни одна из строк не «висит в воздухе».

Если целостность данных нарушена, то с ней нельзя корректно работать. Поддержание целостности данных является одной из основных функций любой СУБД.

Чтобы поддержать целостность сущностей, MySQL автоматически проверяет корректность значения первичного ключа при добавлении и изменении строк.

Поддержать ссылочную целостность сложнее, так как помимо того, что следует проверить корректность значения внешнего ключа при добавлении и изменении строк дочерней таблицы, необходимо также внимательно удалять и изменять строки родительской таблицы.

Ссылочная целостность автоматически поддерживается некоторыми средствами: их несколько, и используются они в разных случаях. В MySQL тот или иной способ указывается при создании дочерней таблицы. Рассмотрим эти способы.

- Запрет (RESTRICT): удаление родительской строки и изменение значения первичного ключа запрещаются, если на строку родительской таблицы ссылается хотя бы одна строка дочерней таблицы. Это означает, что нельзя удалить данные о клиенте, если не удалены данные о его заказах.
- Каскадное удаление/обновление (CASCADE): при удалении строки из родительской таблицы автоматически удаляются все ссылающиеся на нее строки дочерней таблицы; при изменении значения первичного ключа в строке родительской

таблицы автоматически обновляется значение внешнего ключа в ссылающихся на нее строках дочерней таблицы. В этом случае при удалении данных о клиенте удаляются все данные о его заказах.

- Обнуление (SET NULL): при удалении строки и изменении значения первичного ключа в строке значение внешнего ключа во всех строках, ссылающихся на данную строку, автоматически становится неопределенным (NULL).
- При удалении записи о клиенте заказы этого клиента «теряются» — становятся «ничьими».

Возможно, вам это кажется немного сложным, однако, как только вы займетесь практическими решениями, все окажется легче.

При создании таблиц в MySQL можно и не указывать методы поддержания целостности. Однако в этом случае вы сами должны следить за целостностью БД, и это придется делать самостоятельно, целиком полагаясь на свое мастерство программиста.

Работа с базами данных только на первый взгляд кажется трудной и запутанной. На том уровне, на котором это может вам понадобиться в практических задачах, все оказывается совсем не сложным.

Использование базы данных при создании веб-сайта позволяет воспользоваться следующими преимуществами.

- Простота ввода и модификации данных. Исходя из того что данные в БД хранятся отдельно от дизайна сайта, их проще менять — для этого предназначено большое количество интерфейсов редактирования данных (например, phpMyAdmin, предоставляемый многими хостинг-компаниями).
- Скорость работы. Если вы решите организовать собственный метод хранения данных, то наверняка сделаете это либо хуже, чем в СУБД, либо не так удобно и практично.
- Удобство аналитики. Несложные SQL-команды позволяют выделить из БД лишь необходимые вам данные.

Установка и настройка программы. Инструменты графического интерфейса. Командная строка

Дистрибутив СУБД MySQL можно бесплатно скачать с сайта компании-разработчика. Windows-версию MySQL вы найдете на веб-странице http://dev.mysql.com/

downloads/ по ссылке MySQL Community Server. Нужно выбрать платформу (Windows или Windows x64) и один из трех вариантов дистрибутива:

- Essentials — базовый вариант без опциональных компонентов;
- ZIP/Setup.EXE — полный вариант, включающий опциональные утилиты, документацию и др.;
- Without installer — полный вариант, не включающий мастер настройки (требует ручной установки и настройки). Используйте его, если вы хорошо знаете, как работает MySQL.

Мы рекомендуем использовать первый или второй вариант дистрибутива. При скачивании вас попросят зарегистрироваться, однако этого можно избежать, щелкнув кнопкой мыши на ссылке «No thanks, just take me to the downloads!» («Нет, спасибо, я просто скачать пришел!»).

Как и многие другие программы такого рода, MySQL предназначается для работы с командной строкой. Однако для него есть и пакет графических утилит MySQL GUI Tools (http://dev.mysql.com/downloads/gui-tools/5.0.html), позволяющий работать с графическим интерфейсом. Этот пакет включает три программы:

- MySQL Administrator — инструмент администрирования, конфигурирования, мониторинга, запуска/остановки сервера MySQL, а также управления пользователями (различные настройки учетных записей, разрешения доступа и проч.) и соединениями;
- MySQL Query Browser — инструмент создания, выполнения и оптимизации запросов в графической среде;
- MySQL Migration Toolkit — инструмент для переноса данных в MySQL из других реляционных баз данных (Oracle, Microsoft SQL Server, Microsoft Access, Sybase и др.).

Первые две программы очень полезны, а третья может понадобиться непосредственно в тех случаях, когда вам нужно только перенести данные. В штатной работе с программой в ней нет необходимости.

Скачав необходимые дистрибутивы (рекомендуется версия с инсталлятором, то есть не «Without installer»), приступим к установке СУБД MySQL 5.0. Сам процесс установки ничем особо примечательным не отличается.

Чтобы установить Windows-версию программы MySQL на свой компьютер, выполните следующие действия.

1. Запустите мастер установки (Setup Wizard):
   - если вы скачали базовый вариант дистрибутива MySQL, дважды щелкните на значке файла `mysql-essential-5.0.xxx-win32.msi`;
   - если вы скачали полный вариант дистрибутива MySQL, извлеките из архива файл `Setup.exe` и запустите его, дважды щелкнув на значке файла.
2. В начальном окне мастера установки (рис. 3.12) нажмите кнопку Next (Далее).



Рис. 3.12. Начало установки

3. Выберите тип установки (рис. 3.13):
   - `Typical` (Обычная) — установятся только основные компоненты MySQL: сервер и утилиты командной строки. Я рекомендую выбрать именно этот тип, так как он отвечает большинству требований;
   - `Complete` (Полная) — установятся все компоненты MySQL, в том числе библиотеки и заголовочные файлы;
   - `Custom` (Выборочная) — вы сможете указать путь к каталогу, в котором установится программа MySQL, и компоненты, необходимые для установки.
4. В следующем окне, которое непосредственно предваряет саму установку, нужно проверить правильность выбранного типа и каталога установки. Если параметры необходимо изменить, нажмите кнопку Back (Назад). Если параметры указаны верно, для запуска установки нажмите кнопку Install (Установить).
5. После окончания установки на экране появится информационное окно. В этом и последующих аналогичных окнах просто нажимайте кнопку Next (Далее).



Рис. 3.13. Выбор типа установки

6. В заключение укажите необходимость запуска мастера настройки, установив флажок Configure the MySQL Server now (Конфигурировать сервер MySQL сейчас). Затем нажмите кнопку Finish (Готово).

Запустится конфигуратор MySQL (рис. 3.14). Выберите вариант Standard Configuration.

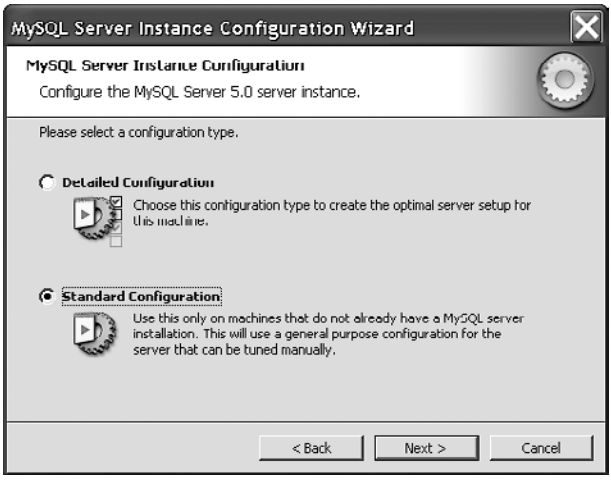


Рис. 3.14. Выбор режима конфигурации

В этом режиме нужно указать, что MySQL будет работать как сервис, а затем установить основной пароль (рис. 3.15).



Рис. 3.15. Ввод основного пароля

Как правило, доступ под основным паролем разрешается только с того компьютера, на котором стоит сервер MySQL, — это сделано в целях безопасности. Пользователю вовсе необязательно иметь доступ к MySQL для получения информации — за него это будет делать сценарий, который посетитель вызовет со своего компьютера. Но сам сценарий будет выполняться на компьютере с веб-сервером, предоставляя пользователю не доступ к базе данных, а результат своей работы.

Когда все действия выполнены, появляется окно, в котором сообщается об успешной установке (рис. 3.16).



Рис. 3.16. Установка успешно завершена

Установка GUI Tools также начинается с выбора режима установки — рекомендую выбрать Complete (рис. 3.17).



Рис. 3.17. Выбор режима установки MySQL Tools

Проверить правильность установки очень просто. Запустите MySQL Administrator, введя в окне авторизации в поле Server Host адрес localhost (это обозначение IP-адреса вашего компьютера), имя пользователя — root и пароль — тот, который вы ввели при установке MySQL (рис. 3.18).



Рис. 3.18. Ввод данных при авторизации

Если все сделано верно, откроется окно администратора (рис. 3.19).

Эта программа позволяет отслеживать состояние сервера MySQL, включать и выключать сервис, делать архивные копии базы данных и многое другое.

Если вы не установили пароль root при инсталляции, то сделать это можно сейчас. Откройте окно командной строки (для этого нажмите кнопку Пуск, выберите пункт Выполнить, в окне Запуск программы в поле Открыть введите команду cmd и нажмите кнопку OK), а затем введите команду

```
mysqladmin -u root password <пароль>
```

(или `C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -u root password <пароль>`, если путь к подкаталогу bin не был добавлен в значение системной переменной `Path` при настройке сервера) и нажмите клавишу Enter.

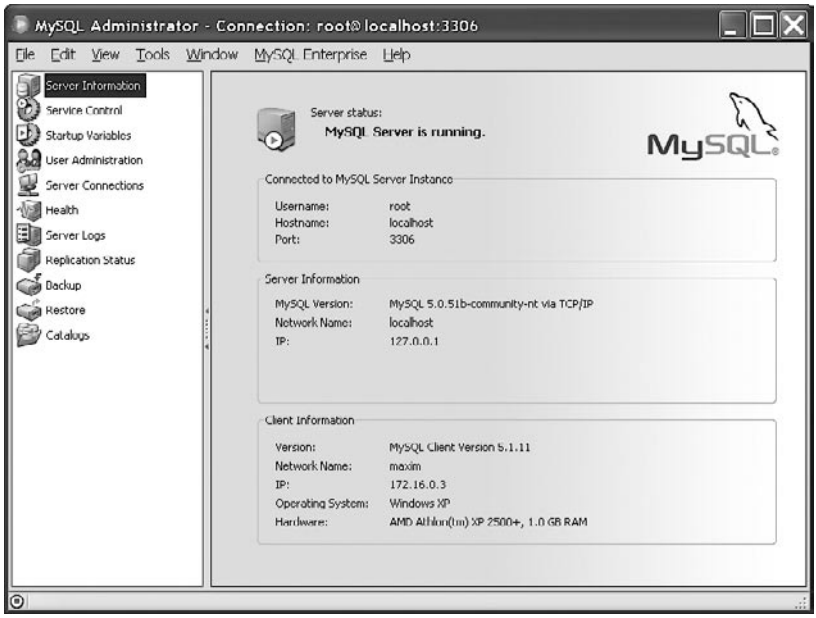


Рис. 3.19. Администрирование MySQL

В дальнейшем, если потребуется сменить пароль пользователя root, выполните такую же команду, только используйте параметр `-p`:

```
mysqladmin -u root -p password <новый пароль>
```

Затем после появления приглашения `Enter password` (Введите пароль) укажите прежний пароль и нажмите клавишу Enter.

Чтобы подключиться к серверу MySQL из командной строки, нужно выполнить следующие действия.

1. Открыть окно командной строки Windows.
2. В командной строке ввести команду

   ```
   mysql -h <Имя компьютера> -u <Имя пользователя> -p
   ```

   (где `<Имя компьютера>` — это имя компьютера, на котором работает сервер) и нажать клавишу Enter. После появления приглашения `Enter password` (Введите пароль) ввести пароль пользователя.

Если требуется подключиться к серверу MySQL, работающему на этом же компьютере, имя компьютера (localhost) можно не указывать, например:

```
mysql -u root -p
```

После подключения к серверу приглашение командной строки изменится на `mysql>` (рис. 3.20).

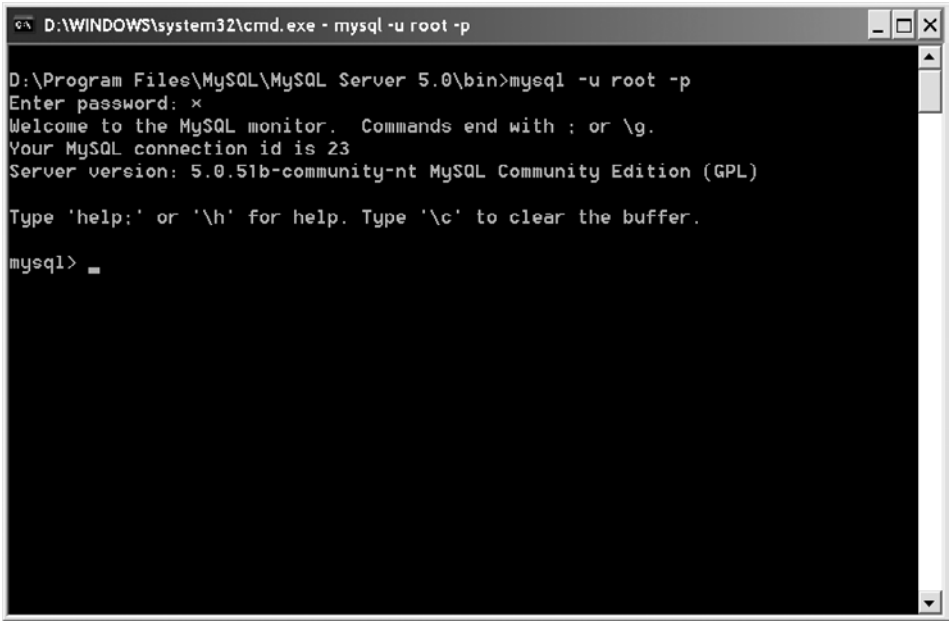


Рис. 3.20. Работа с MySQL из командной строки

Теперь можно приступать к работе с базой данных: добавлять таблицы, вводить и запрашивать данные, регистрировать новых пользователей и др.

Чтобы отключиться от сервера, нужно набрать в командной строке команду `exit` и нажать клавишу Enter.

Альтернативный способ подключения к серверу MySQL предоставляет графическая утилита MySQL Query Browser, которую вы уже установили. В ней также нужно ввести имя пользователя и пароль для подключения к серверу, после чего появится основное окно программы (рис. 3.21).

Работа с MySQL ведется, как правило, в два этапа. На первом этапе задается структура базы данных — создаются таблицы и связи, определяются типы данных, некоторые таблицы наполняются данными (содержимым сайта, списком товаров и т. д.).

На втором этапе происходит, собственно, работа с БД. Если речь идет о сайте, то к базе данных обращаются сценарии: они считывают информацию из таблиц и могут записывать туда данные. Важно организовать работу так, чтобы при выполнении штатных работ не было необходимости создавать новые таблицы или менять структуру имеющихся. Для этого структуру БД следует продумать заранее.

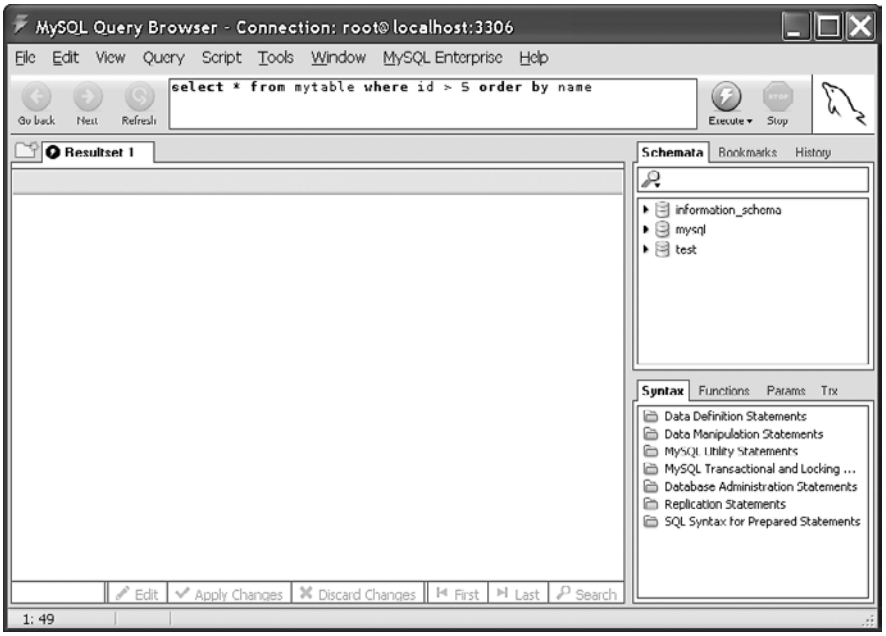


Рис. 3.21. Основное окно MySQL Query Browser

SQL-команды: синтаксис команд, примеры запросов из командной строки, создание базы данных

Прежде чем выполнять SQL-команды, необходимо подключиться к работающему серверу MySQL (как это делать, мы рассмотрели выше).

Если используется командная строка, то для выполнения SQL-команды ее текст вводится в окне командной строки. Для отправки команды на сервер следует нажать клавишу Enter. Чтобы избежать проблем с кодировкой русскоязычных данных, перед началом работы с данными выполните команду

```
SET NAMES cp866;
```

На рис. 3.22 представлен вид этой команды в окне командной строки.

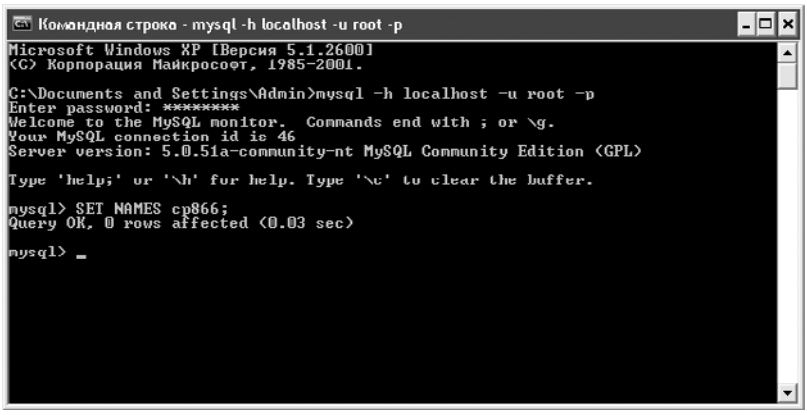


Рис. 3.22. Выбор кодировки

Эту команду необходимо повторять *при каждом* подключении к серверу с помощью командной строки. Она указывает серверу, что используется кодировка CP-866 (это кодировка командной строки Windows).

Сообщения о результате выполнения команд, а также запрашиваемые данные выводятся в окне командной строки. Команда может занимать несколько строчек, причем она должна заканчиваться символом точки с запятой.

Команда `SHOW DATABASES` показывает, какие базы данных находятся под управлением MySQL. Баз может быть несколько (рис. 3.23).

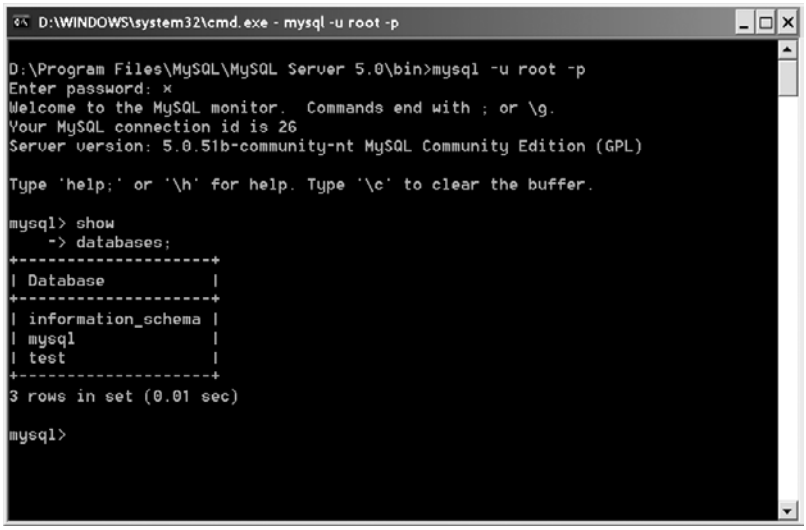


Рис. 3.23. Список имеющихся БД

Чтобы создать базу данных, нужно выполнить команду

```
CREATE DATABASE <Имя базы данных>;
```

Например, команда

```
CREATE DATABASE MySite;
```

создает базу данных с именем `MySite`.

Чтобы удалить ненужную или ошибочно созданную базу данных, выполните команду

```
DROP DATABASE <Имя базы данных>;
```

Удаление базы данных — ответственная операция: она приводит к удалению всех таблиц и данных, хранящихся в таблицах, этой БД.

Для непосредственной работы рекомендуется выбрать в качестве текущей одну базу данных с помощью команды

```
USE <Имя базы данных>;
```

После этого можно выполнять операции с таблицами этой базы данных, не добавляя имя базы данных в виде префикса к имени таблицы.

В MySQL по умолчанию есть три базы данных.

- `INFORMATION_SCHEMA` — информационная база данных, из которой вы можете получить сведения обо всех остальных базах данных.
- `mysql` — служебная база данных, которую использует сервер MySQL. В ней хранятся сведения о зарегистрированных пользователях и их правах доступа, справочная информация и др.
- `test` — пустая база данных, которую можно использовать для «пробы пера», а затем просто удалить.

Базы данных содержат таблицы. Чтобы создать таблицу, выполните команду, представленную в листинге 3.1.

Листинг 3.1. Команда создания таблицы

```
CREATE TABLE <Имя таблицы>
 (<Имя столбца 1> <Тип столбца 1> [<Свойства столбца 1>],
  <Имя столбца 2> <Тип столбца 2> [<Свойства столбца 2>],
  ...
```

```
  [<Информация о ключевых столбцах и индексах>])
 [<Опциональные свойства таблицы>];
```

Команда создания таблицы может включать множество параметров. Многие из них задавать необязательно (в листинге 3.1 такие параметры заключаются в квадратные скобки). В действительности для создания таблицы достаточно указать ее имя, а также имена и типы всех столбцов; остальные параметры используются при необходимости.

Создадим для примера три таблицы: `Customers` (Клиенты), `Products` (Товары) и `Orders` (Заказы).

В листинге 3.2 представлена команда создания таблицы `Customers`.

Листинг 3.2. Команда создания таблицы Customers

```
CREATE TABLE Customers
 (id SERIAL,
  name VARCHAR(100),
  phone VARCHAR(20),
  address VARCHAR(150),
  PRIMARY KEY (id))
 CHARACTER SET utf8;
```

Поясню, какие параметры использовались в этой команде. Во-первых, мы задали название таблицы, а во-вторых, перечислили названия и типы столбцов, из которых будет состоять таблица. Рассмотрим эти параметры подробнее.

- `id` — идентификатор записи. Этому столбцу мы назначили тип `SERIAL`, позволяющий автоматически нумеровать строки таблицы. Ключевое слово `SERIAL` означает `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`. Данное ключевое слово свидетельствует о том, что в столбец можно вводить большие целые (`BIGINT`) положительные (`UNSIGNED`) числа, при этом автоматически контролируется отсутствие неопределенных и повторяющихся значений (`NOT NULL UNIQUE`). Если при добавлении строки в таблицу вы не укажете значение для этого столбца, то программа MySQL внесет в него очередной порядковый номер (`AUTO_INCREMENT`). Можно сказать, что это серийный номер записи.
- `Name` — имя клиента, `phone` — номер телефона и `address` — адрес. Мы присвоили этим столбцам тип `VARCHAR`, поскольку они будут содержать символьные значения. В скобках указывается максимально допустимое количество символов в значении столбца.

В-третьих, мы указали, что столбец `id` будет первичным ключом таблицы, включив в команду создания таблицы определение `PRIMARY KEY (id)`.

Итак, таблица создана. Впоследствии ее структуру можно изменить, однако старайтесь с самого начала задавать ее правильно. Для модификации ранее созданной таблицы используется команда `ALTER TABLE` — она позволяет добавлять, изменять и удалять столбцы.

Работа с таблицами: добавление и удаление записей, выбор с условиями

Как правило, работа с БД включает три основные операции: извлечение данных из БД, ввод новых данных и их модификацию.

Начнем с рассмотрения ввода данных. Для добавления одной или нескольких строк в таблицу можно использовать команду

```
INSERT [INTO] <Имя таблицы>
 [(<Список столбцов>)]
 VALUES
  (<Список значений 1>),
  (<Список значений 2>),
  ...
  (<Список значений N>);
```

В команде `INSERT` используются следующие основные параметры:

- имя таблицы, в которую добавляются строки;
- список имен столбцов, для которых будут задаваться значения (если это делается для всех столбцов таблицы, то приводить список необязательно).

Если имя столбца не включается в список, то в этом столбце при добавлении строки будет автоматически устанавливаться значение по умолчанию. Например, если вы добавляете строку в таблицу с автоинкрементом, то MySQL автоматически поставит в соответствующий столбец новое число.

Набор значений для каждой добавляемой строки заключается в скобки. Эти значения внутри каждой пары скобок должны соответствовать указанному списку столбцов, если же этот список не указывается — то упорядоченному списку всех столбцов, составляющих таблицу. Значения внутри набора и сами наборы отделяются друг от друга запятыми.

Символьные значения, а также значения даты и времени заключаются в одинарные кавычки. Для числовых значений кавычки необязательны. Десятичным разделителем для чисел с дробной частью служит точка. Даты и время вводятся соответственно в формате «YYYY-MM-DD» и «HH:MM:SS».

Чтобы ввести в столбец неопределенное значение, необходимо указать вместо значения ключевое слово NULL *без кавычек* (слово в кавычках рассматривается как обычная символьная строка).

Вместо значения можно указать ключевое слово DEFAULT *без кавычек*, тогда в столбец будет вводиться значение по умолчанию (если оно задается для этого столбца).

Пример использования ключевых слов без кавычек следующий:

```
INSERT INTO Customers (nqme, phnone, address)
 VALUES
 ('Иван Васильевич',
    '555-33-12'
      'Тверская-ямская, 56'),
 ('Василий Иванович,
    NULL,
      'Колхозная, 56');
```

Для изменения данных в таблице применяется, как правило, команда UPDATE, которая позволяет установить новые значения в одной или нескольких строках:

```
UPDATE <Имя таблицы>
 SET <Имя столбца 1>=<Значение 1>,
  ...,
  <Имя столбца N>=<Значение N>
 [WHERE <Условие отбора>]
 [ORDER BY <Имя столбца> [ASC или DESC]]
 [LIMIT <Количество строк>];
```

Например, если у клиента по фамилии Петров изменился номер телефона, то обновить соответствующую информацию в базе данных можно с помощью команды

```
UPDATE Customers SET phone='444-25-27' WHERE id=536;
```

Эту команду можно использовать, если известен код ID Петрова (как правило, он известен).

Ниже приводятся основные параметры команды UPDATE.

- Имя редактируемой таблицы.
- `SET <Имя столбца 1>=<Значение 1>, ..., <Имя столбца N>=<Значение N>` — выбор списка столбцов и новых значений для этих столбцов. Задать новое значение вы также можете с помощью выражения, использующего прежние значения в строке. Например, удвоить рейтинги для всех клиентов можно с помощью команды `UPDATE Customers SET rating=rating*2;`
- `WHERE <Условие отбора>` — выбор условия отбора. Изменения будут применяться только к тем строкам таблицы, которые удовлетворяют этому условию. Если условие отбора не задается, изменения будут применяться ко всем строкам таблицы.
- `ORDER BY <Имя столбца> [ASC или DESC]` — выбор порядка, в котором изменения будут применяться к строкам таблицы. Этот параметр указывается при необходимости. Обычно порядок применения изменений не влияет на результат выполнения операции. Однако в некоторых случаях последовательность действий может быть важна. Например, если вы установили предельное количество изменяемых строк (см. следующий пункт), то некоторые строки, удовлетворяющие условию отбора, могут остаться неизменными, а какие именно это будут строки — зависит от последовательности применения изменений.
- `LIMIT <Количество строк>` — выбор максимального количества строк таблицы, которые могут изменяться командой UPDATE. Этот параметр также указывается при необходимости. Если количество измененных строк достигнуто, то операция завершается, даже если в таблице еще остались строки, которые удовлетворяют условию отбора и не изменились.

Для получения информации из таблиц базы данных применяются запросы. Для создания запроса используется SQL-команда SELECT:

```
SELECT * FROM <Имя таблицы>;
```

Например, в результате выполнения запроса

```
SELECT * FROM Customers;
```

вы получите всю информацию из таблицы.

Вместо звездочки можно указать список столбцов таблицы, из которых нужно получить информацию. Например, чтобы вывести только имя, телефон и рейтинг каждого клиента, выполните запрос

```
SELECT name, phone, rating FROM Customers;
```

Запросы позволяют получать не только значения столбцов, но и значения, вычисленные с помощью выражений. Например, запрос

```
SELECT name, phone, rating/1000 FROM CUSTOMERS;
```

возвращает результат, аналогичный предыдущему, только значения рейтинга разделены на 1000.

Результат запроса может содержать повторяющиеся строки. Например, клиенты могут иметь одинаковые рейтинги, поэтому запрос

```
SELECT phones FROM Customers;
```

выдает результат, в котором перечисляются все телефоны из таблицы.

Бывает, что у разных клиентов телефоны совпадают. Если нужно исключить повторения из результата запроса, то в его текст нужно добавить ключевое слово DISTINCT:

```
SELECT DISTINCT rating FROM Customers;
```

Чтобы упорядочить выведенные по запросу строки по значениям одного из столбцов, добавьте в текст запроса выражение

```
ORDER BY <Имя столбца> [ASC или DESC]
```

Ключевое слово ASC означает, что сортировка выполняется по возрастанию значений, а DESC — по убыванию.

В запрос можно также добавить условие отбора — WHERE. Это делается точно так же, как и при выполнении команды UPDATE. В условии отбора можно использовать любые операторы и функции языка SQL, в том числе логические операторы AND и OR для создания составных условий отбора. Например, запрос

```
SELECT name,phone,rating FROM Customers
 WHERE name LIKE 'ООО%' OR rating>1000
 ORDER BY rating DESC;
```

выводит информацию о клиентах, чье имя начинается с «ООО», и о тех, чей рейтинг превосходит 1000, упорядочивая строки в порядке убывания значения рейтинга. Символ % означает любое количество произвольных символов.

Операторы языка SQL

Функции и операторы позволяют создавать *выражения* — формулы, вычисляющие какое-либо значение (числовое, логическое, символьное и др.). Наиболее часто

выражения используются в SQL-запросах: как для вычисления значений, выводимых запросом, так и в условиях отбора. Выражения также позволяют задавать условия отбора в SQL-командах UPDATE и DELETE, значения, добавляемые в таблицу, в командах INSERT и UPDATE и многое другое.

Вначале рассмотрим наиболее часто используемую группу операторов, которые предназначаются для создания условий отбора. Это операторы <, >, =, <>, !=, служащие для сопоставления двух операндов, например:

```
SELECT name,phone,rating1,rating2 FROM Customers
 WHERE rating1 > rating2 and name != `Петров`
```

Здесь выбираются все строки таблицы, в которых одно поле содержит значение, превышающее значение другого поля, за исключением той строки, в которой приводятся сведения о Петрове.

Есть и более сложные операторы сравнения, но на них мы подробно останавливаться не будем.

Оператор сравнения с шаблоном LIKE сравнивает элементы таблицы с шаблоном. В последнем можно использовать два специальных подстановочных символа:

- % — на месте знака процента может быть любое количество произвольных символов;
- _ — на месте знака подчеркивания может быть только один произвольный символ.

Например, следующий запрос выводит данные о клиентах, чьи имена содержат двойные кавычки:

```
SELECT * FROM Customers WHERE name LIKE '%"%"%';
```

Если требуется включить в шаблон знак процента или подчеркивания, которые должны рассматриваться не как подстановочные, а как обычные символы, то перед ними нужно поставить обратную косую черту («\%», «_»). Если же шаблон должен содержать символ обратной косой черты, то ее нужно удвоить («\\»).

По умолчанию сравнение с помощью оператора LIKE выполняется без учета регистра символов. Для сравнения с учетом регистра необходимо указать ключевое слово BINARY или правило сравнения (COLLATE). Например, выражение 'Крылов' LIKE 'крылов' истинно, а выражения 'Крылов' LIKE BINARY 'крылов' и 'Крылов' LIKE 'крылов' COLLATE utf8_bin ложны.

Настройка взаимодействия MySQL и Perl

Теперь, когда вы знаете, что такое Apache, Perl и SQL, попробуем наладить взаимодействие этих продуктов.

Чтобы программы на Perl могли обращаться к СУБД MySQL, необходимо настроить ActivePerl. Для этого из командной строки (Пуск ▸ Выполнить ▸ CMD) нужно выполнить следующие команды:

```
ppm install DBI
ppm install DBD-mysql
```

Если по какой-то причине они выдают ошибку, то нужно написать следующую команду:

```
ppm install http://theoryx5.uwinnipeg.ca/ppms/DBD-mysql.ppd
```

Если и это вызывает трудности, то можно скачать один из готовых пакетов, который уже включает в себя интегрированные Apache, Perl и MySQL, например «Денвер».

«Денвер» — это российский проект, его можно скачать с сервера http://www.denwer.ru/. Он очень удобен для работы, поэтому остановимся на нем подробнее.

Установка проекта «Денвер» и работа с ним

Скачав «Денвер», запустите программу установки. Для этого нужно выполнить следующие действия.

1. На первый вопрос (рис. 3.24) ответьте положительно — Да.

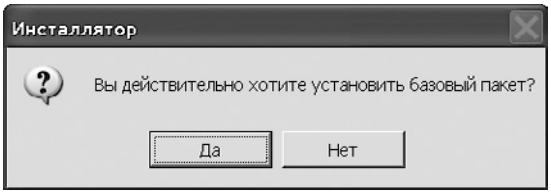


Рис. 3.24. Начало установки

2. Откроется окно программы Internet Explorer с краткой информацией о «Денвере» (рис. 3.25). Для продолжения установки закройте его.
3. Дальнейшая установка будет вестись в окне консоли (рис. 3.26). Здесь выдаются инструкции по дальнейшей инсталляции и запрашивается необходимая информация.

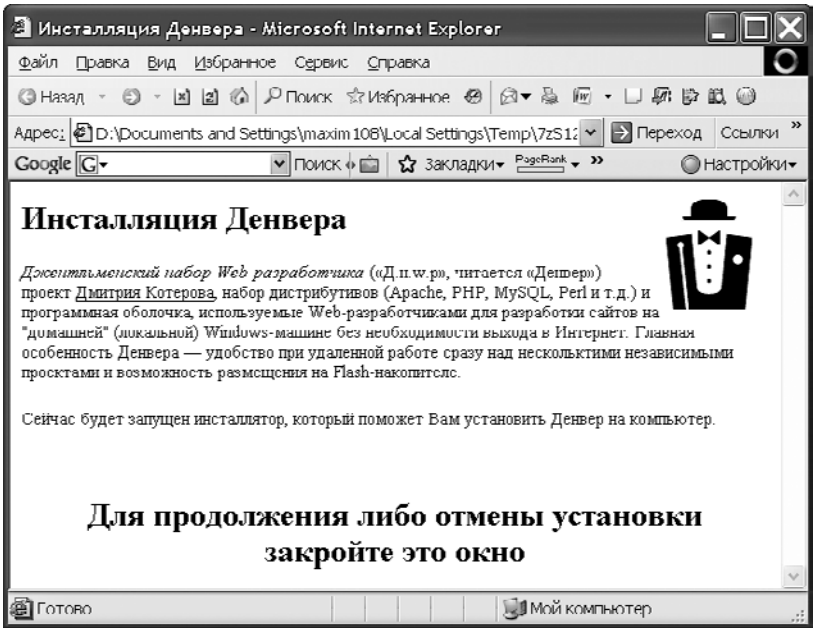


Рис. 3.25. Информация о «Денвере»

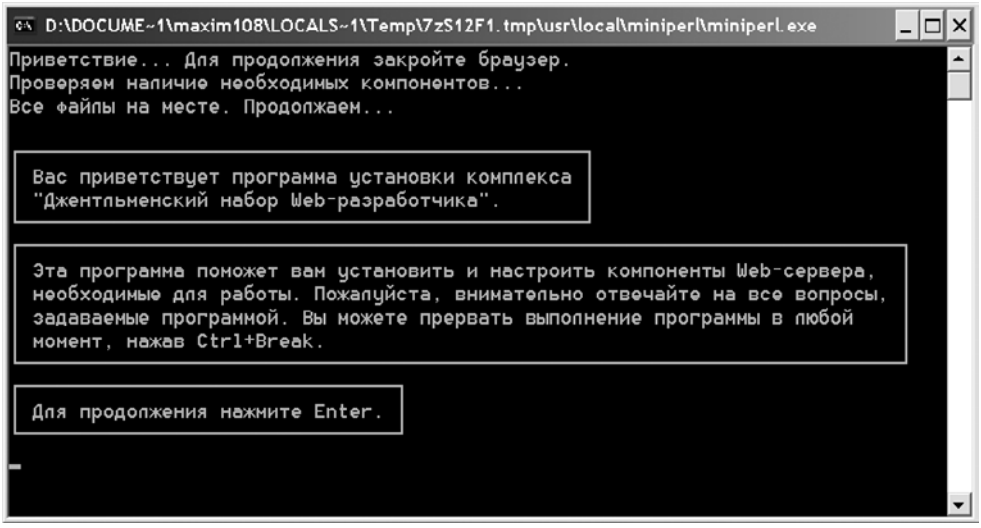


Рис. 3.26. Установка «Денвера»

4. В процессе установки нужно ответить на два вопроса: куда устнавливать пакет (рис. 3.27) и какую букву виртуального диска использовать (рис. 3.28). Использовать виртуальный диск с отдельной буквой очень удобно — это

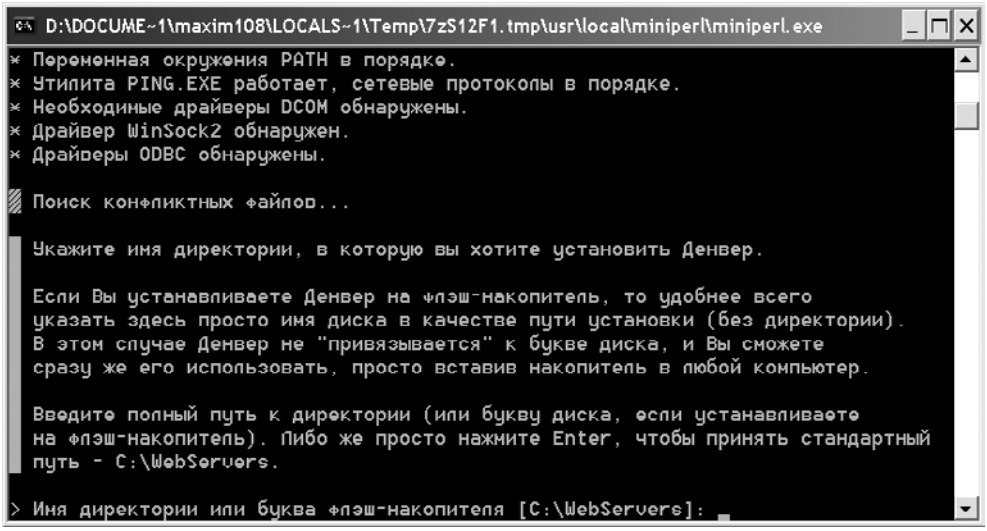


Рис. 3.27. Указание директории для установки

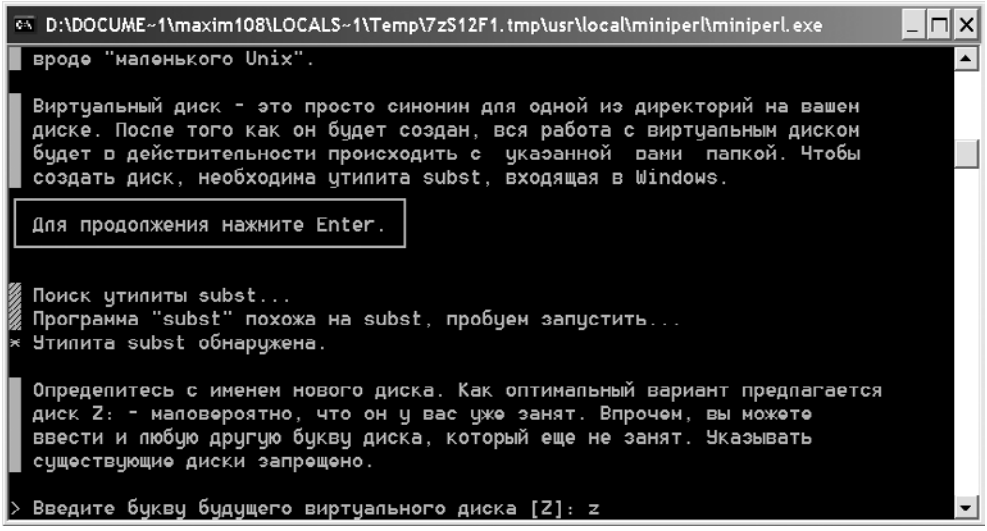


Рис. 3.28. Указание буквы виртуального диска

отдельный диск среди прочих дисков, который целиком находится в распоряжении вашего сайта.

5. Теперь начнется копирование файлов дистрибутива, по завершении которого нужно указать способ запуска «Денвера»: будет ли он запускаться при загрузке операционной системы или же только в ручном режиме (рис. 3.29).

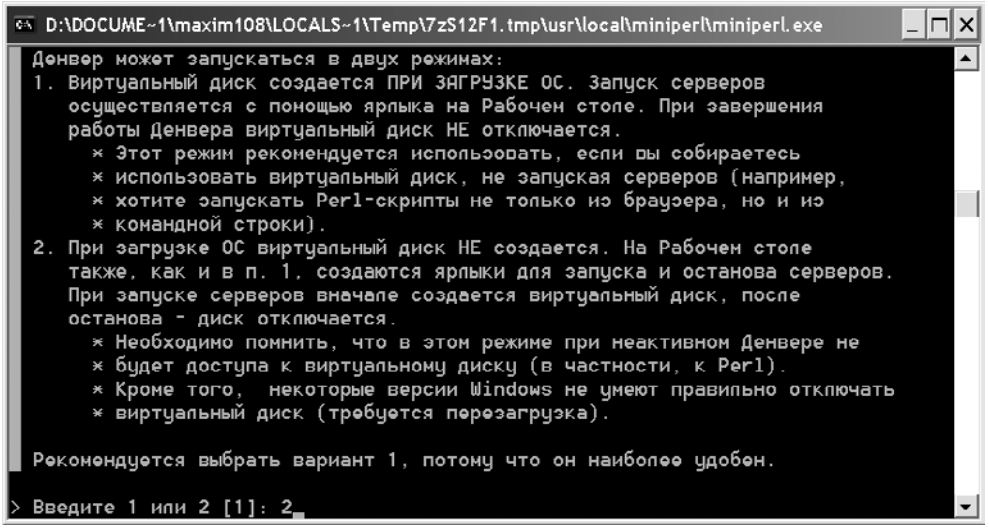


Рис. 3.29. Способ запуска «Денвера»

6. Наконец «Денвер» установлен (рис. 3.30).

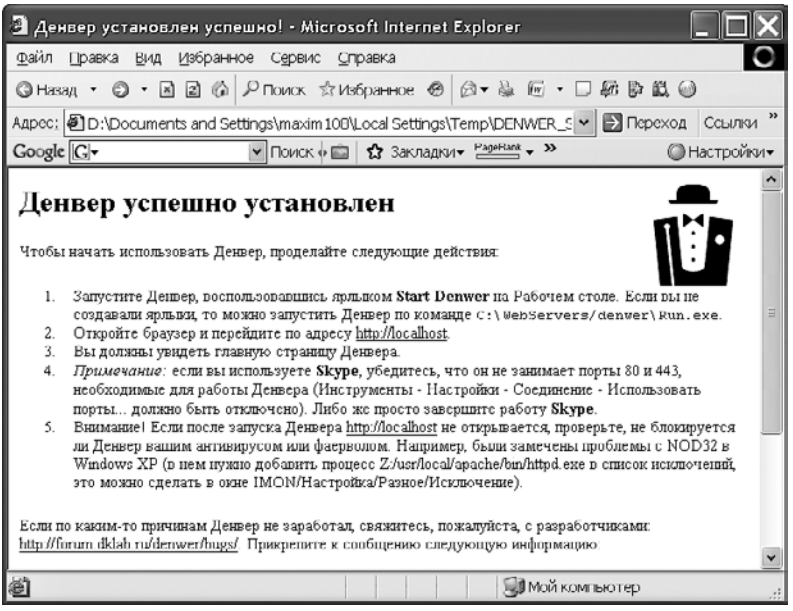


Рис. 3.30. Завершающее окно установки

7. Чтобы проверить, работает ли он, перейдите по адресу http://localhost/ (рис. 3.31).

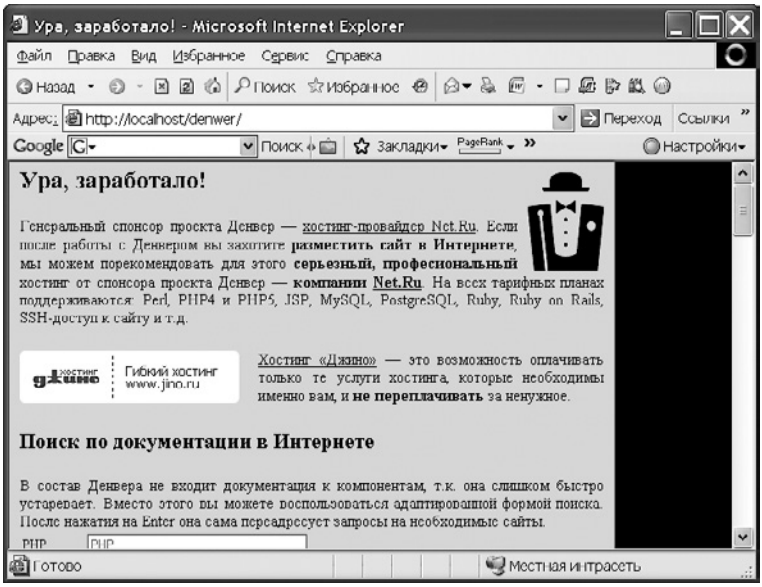


Рис. 3.31. «Денвер» работает

Если окно, показанное на рис. 3.31, не появилось, то «Денвер» работает неправильно. Имейте в виду, что он не функционирует совместно с программой Skype. Чтобы работать с этими программами одновременно, необходимо отключить в Skype использование портов 80 и 443 (см. рис. 3.30). Кроме того, «Денвер» не заработает, если у вас запущен Apache, который вы установили до этого. В этом случае просто остановите Apache (через панель управления или интерфейс Apache) и перезапустите «Денвер».

Если «Денвер» заработал, то у вас появится новый диск, в котором будут созданы папки с названиями серверов (рис. 3.32).

Обращение к базе SQL из программы Perl

Теперь, после того как Perl «познакомился» с MySQL, можно подключать базу данных в сценарии:

```
my $dsn = "DBI:MYSQL:test:localhost";
my $dbh = DBI->connect($dsn, "username", "password") ||
die "[err]: Невозможно подключиться к MySQL: $!";
```

Обратите внимание на следующие моменты: во-первых, имя пользователя и пароль необходимо прописывать в файле в явном виде. Во-вторых, соединение с MySQL

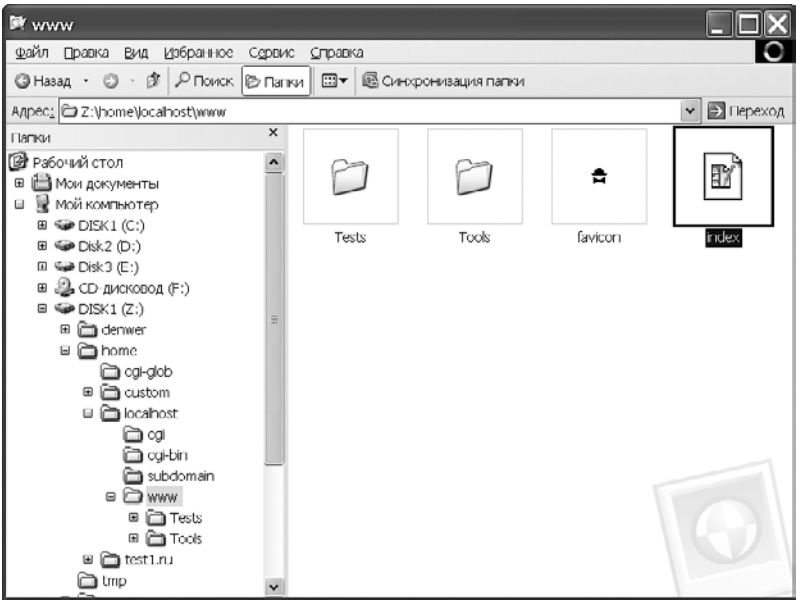


Рис. 3.32. Новый диск Z

нужно создавать каждый раз при обращении к серверу, то есть в каждом файле, который работает с БД.

Основная операция при работе с БД из Perl — выбор данных с помощью функции `Select`:

```
my $sth = $dbh->prepare(qq{select name, phone, id from users});
$sth->execute();
while (my ($name, $phone, $id) =
sth->fetchrow_array())  # делать выборку данных
{
print "$id - $name, $phone <br>";
}
$sth->finish();
```

Здесь мы сначала передаем запрос в базу данных с помощью функций `prepare` и `execute`. Затем для получения ответа от базы данных организуется цикл с функцией `fetch_row` внутри — она возвращает данные из базы данных строку за строкой. Эти данные попадают в переменные `name`, `phone` и `ID`.

Аналогичным образом реализуются и другие команды — `UPDATE`, `INSERT` и пр. В них необязательно использовать цикл.

Если результат действия команды не нужен, то работа организуется еще проще:

```
$dbh->do("insert into users (name, phone) values ('Вася', '111-22-33')");
```

Вот простой пример организации отображения веб-страницы из базы данных:

```
my $dsn = "DBI:MYSQL:mysite:localhost";
my $dbh = DBI->connect($dsn, "username", "password") ||
die "[err]: Невозможно подключиться к MySQL: $!";
my $sth = $dbh->prepare(qq{select title, body from articles where id=$id});
$sth->execute();
(my ($title, $body) = sth->fetchrow_array());
$sth->finish();
Print "<html><body><title>$title</title>";
print <<HTML;
// здесь идет дизайн страницы до основного текста
HTML
Print $body;
print <<HTML;
// здесь идет дизайн страницы после основного текста
HTML
```

Этот код позволяет создать один сценарий, который будет отображать содержимое всех страниц. Изменить их дизайн одновременно очень легко: достаточно исправить в этом сценарии HTML-код.

Скрипт определяет, на какую именно страницу пользователь зашел в данный момент: обращение к базе данных в нем происходит по уникальному номеру (`id`) записи. Обычно уникальный номер передается в ссылке в качестве параметра сценария.

Ссылка на страницу может быть вида http://www.mysite.ru/article.cgi?id=15. Как анализировать параметры, переданные сценарию, мы рассмотрели в подразд. «Примеры использования Perl для взаимодействия с пользователем».

Резюме

Из этой главы вы получили все необходимые знания, которые понадобятся вам при построении собственной базы данных и работы с ней. Вы научились создавать, изменять и удалять базы данных и таблицы, настраивать ключевые столбцы и индексы, а также познакомились с типами данных, используемых в MySQL.

Вы освоили такие операции с данными, как добавление, изменение и удаление строк таблицы. Кроме того, вы узнали, как можно найти в базе данных нужную информацию с помощью запросов.

Конечно, описать в одной главе языки программирования Perl и MySQL целиком невозможно. Однако я уверен, что приведенные здесь сведения помогут разобраться, нужны ли они вам для работы, и принять решение об их дальнейшем углубленном изучении. Проанализировав простые примеры, вы сможете создавать небольшие сценарии, а изучая их работу, вы углубите свои познания в языках Perl и MySQL.

Глава 4

Знакомство с основными системами управления содержимым сайта (CMS) — Joomla! и Drupal

CMS Joomla!

CMS Drupal

Резюме

В начале своей истории сайты в Интернете были несложными, чаще их даже называли не сайтами, а страницами. Их создавали в основном энтузиасты. Оформление сайтов было самым простым, тем более мало кто говорил о том, что их можно сделать интерактивными. Но шло время, и Интернет постепенно привлек к себе внимание крупных коммерческих организаций. Появилась потребность в сайтах, способных вести диалог с пользователями. Одновременно с этим стало ясно, что сайт должен иметь две составляющие: интерфейс пользователя и административную часть, в которой владельцы сайта могут управлять содержимым. Таким образом, назрела необходимость в создании первых систем управления содержимым — CMS (англ. Content Manager Site, система управления содержимым сайта).

Первые CMS были написаны, как правило, на одном из классических языков программирования. При этом интерфейс пользователя создавался с использованием языков HTML и JavaScript, а административная часть представляла собой обычную программу, написанную на одном из языков программирования. Недостатком такой организации сайта является необходимость установки на рабочее место каждого работника, поддерживающего сайт, части CMS. На системных администраторов легла дополнительная большая нагрузка по поддержанию программ на рабочих местах.

Еще больше усложнилась работа сотрудников, которые трудились удаленно. Кроме того, не было единого инструментария, позволявшего относительно быстро и качественно создавать сайты. Поэтому формирование даже средних по насыщенности информацией сайтов по-прежнему оставалось задачей крупных групп разработчиков. Назрела необходимость создания таких CMS, которые, с одной стороны, были бы просты в установке даже для не очень опытных программистов и, с другой стороны, предоставляли гибкие настройки внешнего вида сайта и простое управление содержимым, а также не требовали установки на компьютеры всех сотрудников, поддерживающих сайт. Решение этой задачи и стало причиной развития интерактивных технологий Интернета. Было создано очень много CMS. Одними из представителей таких систем являются Joomla! и Drupal. Рассмотрим каждую из них.

CMS Joomla!

В начале XXI в. под лицензией GPL была выпущена одна из самых первых систем управления сайтом — Mambo. В 2005 г. от этого проекта отделилась группа разработчиков и назвала свой проект Joomla!. Главная особенность этих

систем состоит в том, что с их помощью можно создать и поддерживать сайт даже без знания языков программирования. Это достигнуто за счет использования компонентов системы, из которых можно, как из конструктора, собрать свой сайт.

Рассмотрим возможности проекта Joomla!.

- Система является кросс-платформенной, то есть она одинаково работает под управлением Windows, Linux, Mac OS, Free BSD и других операционных систем. Это достигается за счет того, что при создании Joomla! использовался только язык PHP, который, в свою очередь, функционирует на многих платформах. Поэтому точнее будет сказать, что Joomla! работает на любом веб-сервере, поддерживающем PHP.
- Позволяет создавать практически неограниченное количество страниц сайта. Ограничения больше обусловливаются применяемыми на сервере файловыми системами и ограничениями MySQL.
- Дает возможность легко настраивать и изменять внешний вид сайта с помощью шаблонов. При этом меняется только внешнее оформление сайта, а не его внутренняя структура.
- За счет использования модулей можно существенно расширить функциональность страниц сайта, например поставить счетчик посещений, загружать и показывать ленты новостей.
- Применение компонентов позволяет создать собственные форумы, интернет-магазины, фотоальбомы, опросы и многое другое.
- Можно подготовить для сайта материал и указать определенное время его опубликования.
- Администратор может легко блокировать и разблокировать любые страницы, следя таким образом за содержимым сайта.
- Для каждой динамической страницы можно задать список ключевых слов, что позволит поисковым системам включать ваши страницы в рейтинги.

Это далеко не самый полный перечень возможностей Joomla!.

Система Joomla! состоит из двух больших частей.

- Административная часть — часть сайта, доступная администратору.
- Сайт, или пользовательская часть, — часть сайта, которую видит любой пользователь, вошедший на сайт.

Установка

Для установки системы на сервере должны быть:

- установлен и запущен PHP версии не ниже 4.3.10;
- установлен и запущен MySQL;
- включена поддержка XML;
- прописана поддержка библиотеки сжатия `zlib`;
- доступен для чтения и записи каталог, в который вы будете распаковывать дистрибутив.

Рассмотрим установку системы пошагово.

1. Создайте в корневом каталоге сайта подкаталог joomla.
2. Перепишите в него архив Joomla_1.5.5-Stable-Full_Package.tar.bz2.
3. Распакуйте содержимое файла в каталог joomla. После этого архив `Joomla_1.5.5-Stable-Full_Package.tar.bz2` можно удалить. Для распаковки архива можно воспользоваться архиватором 7-Zip в Windows или tar в UNIX-подобных операционных системах.

   В результате команда `dir` (или `ls` в «Линуксе») должна показать примерно следующее:

```
L:\home\localhost\www\joomla>dir /B
administrator
cache
CHANGELOG.php
components
configuration.php-dist
COPYRIGHT.php
CREDITS.php
htaccess.txt
images
includes
index.php
index2.php
INSTALL.php
installation
```

```
language
libraries
LICENSE.php
LICENSES.php
logs
media
modules
plugins
robots.txt
templates
tmp
xmlrpc
sdsad
```

4. Откройте любой браузер и наберите в нем адрес http://localhost/joomla — в результате откроется страница установки Joomla! (рис. 4.1). В дальнейшем будем считать, что этот адрес является адресом нашего сайта.

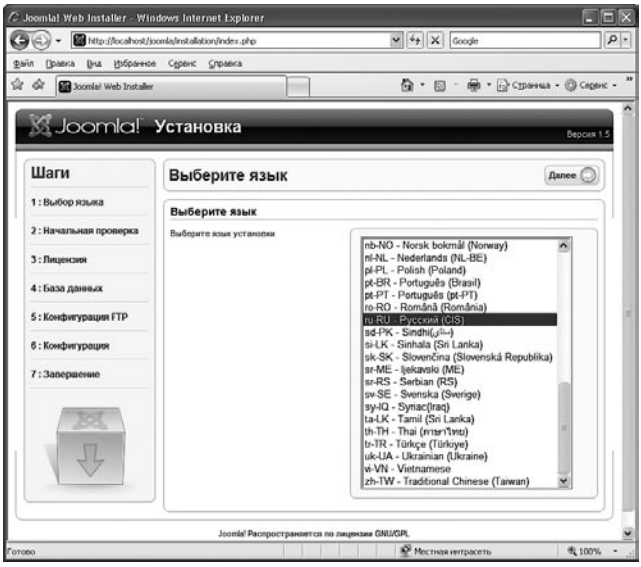


Рис. 4.1. Шаг 1. Выбор языка установки

Первый шаг при установке системы — выбор языка. Укажите русский язык и нажмите кнопку Далее. Обратите внимание, что здесь вы выбрали только язык установки. Русификацию системы Joomla! мы рассмотрим позже.

5. Убедитесь, что конфигурация вашей системы удовлетворяет минимальным требованиям Joomla!.

   В разделе Проверка Joomla! все пункты должны быть помечены Да (рис. 4.2).

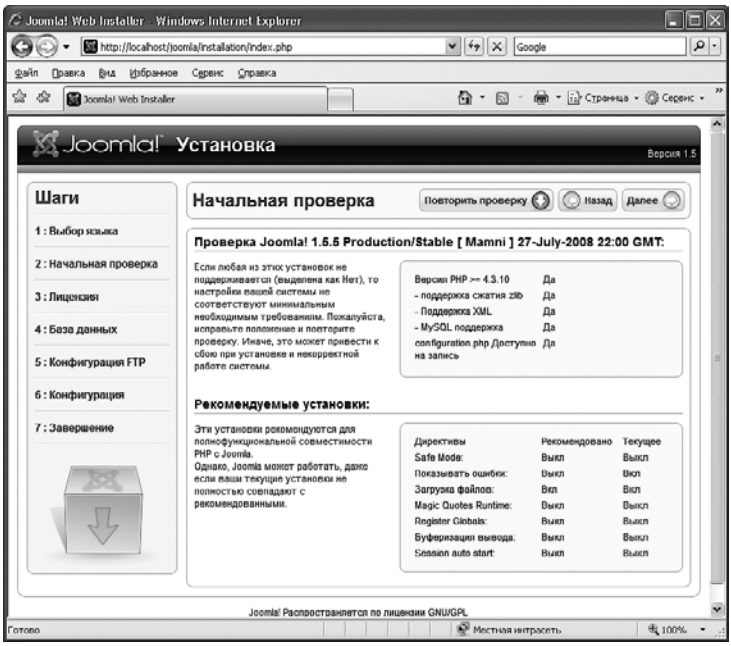


Рис. 4.2. Начальная проверка

- Версия PHP >= 4.3.10 — если версия PHP ниже, то Joomla! будет работать некорректно.
- Поддержка сжатия zlib — эта возможность необходима, так как модули, шаблоны и компоненты системы всегда поставляются в файлах, сжатых с использованием утилиты сжатия и восстановления (декомпрессии) файлов gzip.
- Поддержка XML — эта возможность необходима для действия технологии AJAX, а также для реализации управления настройками модулей.
- MySQL — сервер баз данных MySQL является единственным, который может использовать Joomla! в своей работе; подключение к нему является обязательным, поскольку в базе данных хранятся все данные сайта.
- configuration.php Доступно на запись — эта возможность нужна только для установки Joomla! и для администратора сайта. Для пользователей этот файл доступен только для чтения или вообще не виден.

Раздел Рекомендуемые установки нужен только веб-разработчикам для настройки удобства работы администратора сайта и оптимизации производительности работы сайта. На установку и внешний вид сайта эти параметры не влияют.

Если в разделе Проверка Joomla! все пункты отмечены Да, можно переходить к следующему шагу установки, нажав кнопку Далее.

Если какие-то параметры отмечены Нет, необходимо приостановить установку системы, устранить проблемы и после этого нажать кнопку Повторить проверку.

6. В следующем разделе (рис. 4.3) нужно решить, согласны ли вы с лицензией GNU/GPL. Если не согласны или что-то забыли проверить на предыдущих страницах, нажмите кнопку Назад. В противном случае нажмите кнопку Далее.

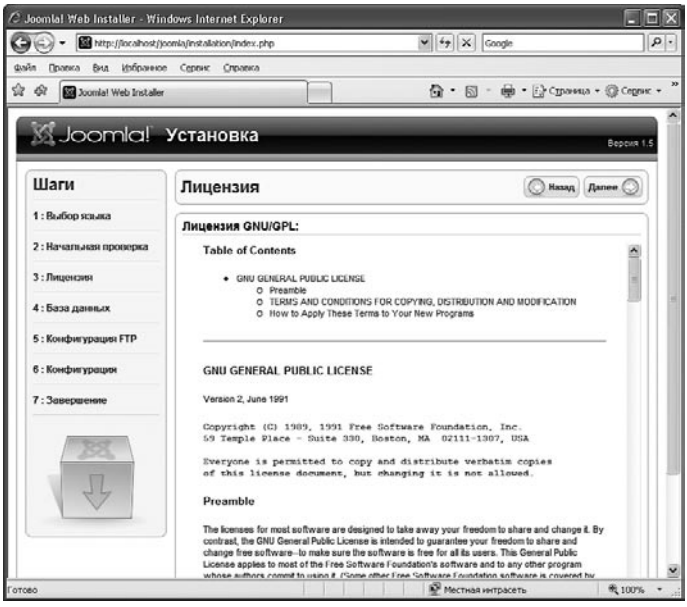


Рис. 4.3. Лицензия GNU/GPL

7. Необходимо настроить сервер баз данных, который будет использовать Joomla! в своей работе (рис. 4.4).

   Как было сказано ранее, Joomla! хранит свои данные на сервере баз данных. В качестве сервера БД используется MySQL.

   На странице настроек доступа к MySQL имеются следующие разделы.

   - Тип базы данных — установите MySQL. Ставить MySQL в качестве сервера баз данных для нашего учебного сайта нет смысла.

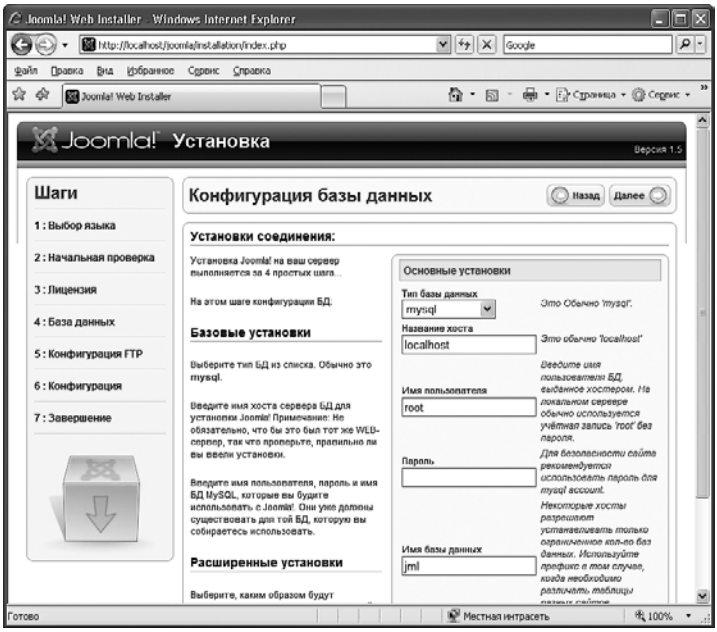


Рис. 4.4. Настройка доступа к MySQL

- Название хоста — укажите IP-адрес либо название сервера, на котором запущен MySQL (тот, к которому вы имеете доступ).
- Имя пользователя и Пароль — укажите имя пользователя и пароль, которые позволят создавать базы, таблицы в них и получать доступ на запись в последние.
- Расширенные установки — этим разделом стоит воспользоваться, если у вас уже есть база данных в MySQL с тем именем, которое вы хотите задать. В этом случае щелкните на названии данного раздела — в результате откроются дополнительные параметры настройки:
  - Удалить существующие таблицы — если при установке новой версии системы имена существующих таблиц в базе данных совпадут с именами вновь создаваемых таблиц, старые таблицы будут удалены;
  - Создать резервную копию старых таблиц — этот параметр аналогичен предыдущему, но в этом случае существующие таблицы не будут удаляться, а переименуются с префиксом `old_`; если же в базе уже есть резервные копии таблицы, резервные копии будут заменяться на новые;
  - Префикс таблиц — префикс системных таблиц Joomla!. Этот префикс нужен для того, чтобы гарантировать несовпадение имен системных

таблиц Joomla! с уже существующими рабочими таблицами в базе данных. Во вновь создаваемой системе это большого значения не имеет, однако во избежание проблем в будущем лучше задать префикс. По умолчанию устанавливается префикс `jos_`.

После выбора и проверки настроек MySQL на этой странице нажмите кнопку Далее.

8. Необходимо настроить параметры доступа по FTP (рис. 4.5).

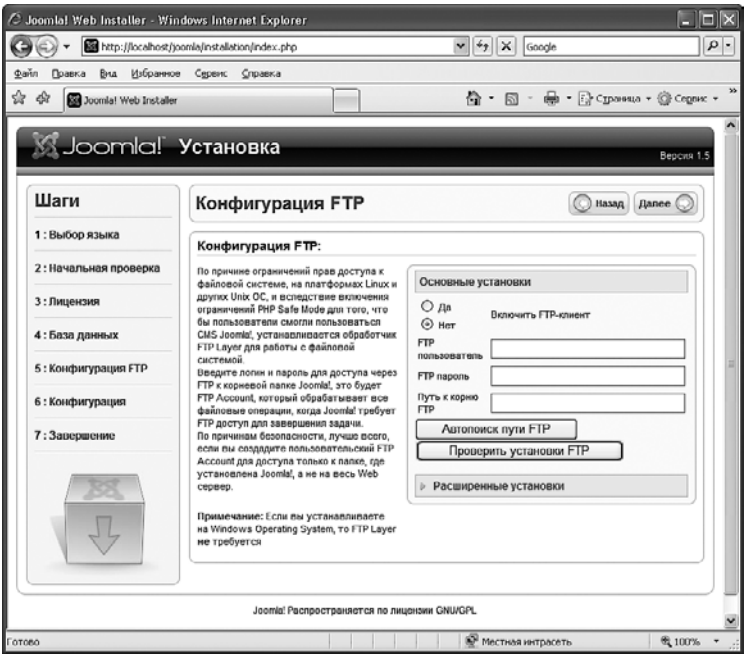


Рис. 4.5. Настройка FTP

На UNIX-подобных системах (Linux, Free BSD и др.) требуется указать имя пользователя, пароль и корневой каталог, в котором находится проект Joomla!. Это необходимо для того, чтобы Joomla! могла получить доступ к своим конфигурационным файлам и папкам. Данная настройка не играет роли, если вы устанавливаете Joomla! на сервер под управлением ОС Windows.

После настройки конфигурации FTP нажмите кнопку Далее.

9. На следующей странице нужно настроить Главную конфигурацию (рис. 4.6).

С этого момента наступает самая интересная часть установки. Вам необходимо задать название своего сайта. Обязательно укажите электронный адрес администратора сайта в строке Ваш e-mail группы E-mail и пароль администратора,

чтобы Joomla! и пользователи сайта могли сообщать обо всех проблемах работы сайта по электронной почте. Наконец, установите пароль доступа администратора к системе управления сайтом.

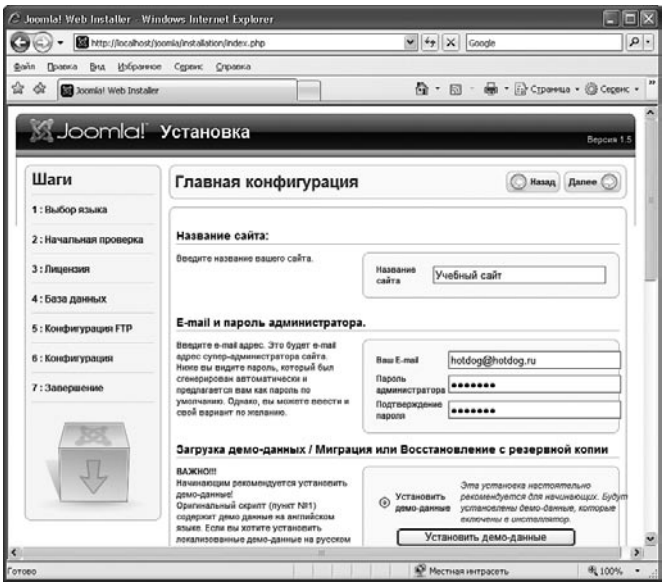


Рис. 4.6. Настройка главной конфигурации

Следующий раздел — Загрузка демо-данных/Миграция и восстановление с резервной копии. Этот раздел предназначается для того, чтобы пользователь смог указать вид установки. Поскольку мы создаем тестовый учебный сайт, нажмите кнопку Установить демо-данные. Обратите внимание, что надпись на кнопке сменилась на текст Демо-данные успешно установлены (рис. 4.7).

При установке системы для реальной работы эту кнопку нажимать не следует.

После того как в этом окне настроены все параметры, нажмите кнопку Далее в верхней части страницы.

10. На странице Завершение установки вы получите сообщение, что установка завершена и можно перейти либо к администрированию сайта (нажав кнопку Админ), либо непосредственно на сайт (нажав кнопку Сайт) (рис. 4.8).

Прежде чем приступить к работе в Joomla!, необходимо удалить каталог installation в каталоге, в который был распакован ее дистрибутив. Это следует сделать обязательно, поскольку в каталоге installation после установки остается информация, которая может быть доступна для потенциальных взломщиков сайта. В дальнейшей работе с сайтом этот каталог не нужен.

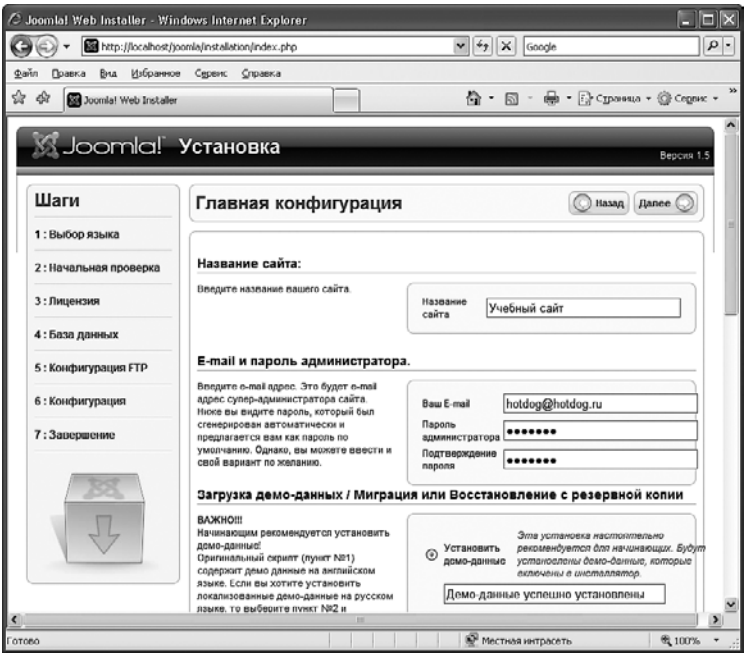


Рис. 4.7. Установка демо-данных

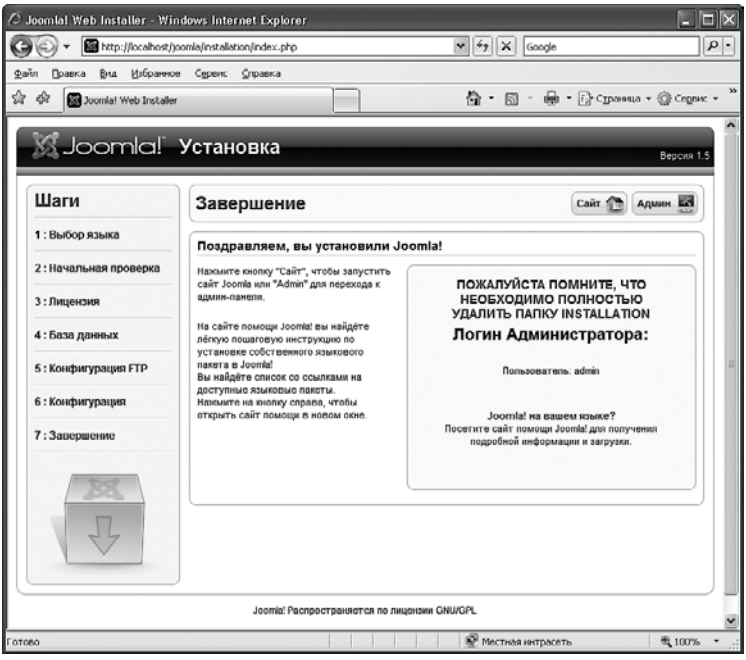


Рис. 4.8. Завершение установки

Администрирование

Чтобы получить доступ к администрированию сайта, нужно ввести в адресную строку браузера названия главного каталога сайта и подкаталога administrator к нему. Например, в нашем случае главным каталогом сайта является http://localhost/joomla, поэтому адресом административной части сайта будет http://localhost/joomla/administrator. После ввода этого адреса откроется страница регистрации пользователя в административной части сайта. Для входа в режим администрирования сайта необходимо ввести имя пользователя и пароль доступа администратора (рис. 4.9).



Рис. 4.9. Вход в систему администрирования

1. В поле Username укажите admin (это имя пользователя для администратора по умолчанию).
2. В поле Password введите пароль, который вы указывали при установке Joomla! на странице Главная конфигурация.
3. В поле Language можно выбрать язык интерфейса административной части сайта. По умолчанию установлен английский язык. О русификации сайта поговорим чуть позже.

Теперь нажмите кнопку Login — в результате появится главная страница администрирования системы (рис. 4.10).



Рис. 4.10. Главная страница администрирования

Главная страница администрирования разделяется на несколько частей: главное меню (в верхней части страницы), поле выбора наиболее часто выполняемых операций и информационная панель (справа). На главную страницу всегда можно попасть, выбрав команду меню Site ▸ Control Panel (Сайт ▸ Панель управления — в русской версии программы).

Русификация

Программа установки русифицирована, однако после установки вы получаете в распоряжение систему, работающую на английском языке. Хотя сегодня последней стабильной версией является версия 1.5.5, русифицирована только версия 1.5.4. Но это не вызывает никаких проблем.

Сайт состоит из двух частей — пользовательской и административной. Следовательно, нужно русифицировать отдельно пользовательскую и административную части.

Для русификации пользовательской части сайта выполните следующие действия.

1. Скачайте файл `ru-RU_Joomla_1.5.4_site.zip`.
2. Распакуйте его в каталог ru-RU.

ВНИМАНИЕ



Распаковывать содержимое архива необходимо именно в каталог ru-RU, в противном случае русский язык появится в меню, но русификации не произойдет.

3. Перепишите каталог ru-RU в каталог установки Joomla!, в подкаталог `language` (в этом каталоге уже есть подкаталог en-GB).

Теперь русифицируйте административную часть сайта.

1. Скачайте файл `ru-RU_Joomla_1.5.4_admin.zip`.
2. Распакуйте его содержимое в подкаталог ru-RU.
3. Перепишите каталог ru-RU в каталог установки Joomla!, в подкаталог administrator/language (в этом каталоге уже есть подкаталог en-GB).
4. Наберите адрес административной части системы, на странице ввода имени пользователя и пароля в поле Language выберите язык `Russian (CIS)`.
5. Зайдите на сайт, нажав кнопку Login.

Пока вы установили только библиотеки русского языка, но сайт по умолчанию работает на английском языке. Теперь установите русский язык в качестве языка по умолчанию для пользовательской и административной частей сайта.

1. Войдите в административную часть сайта.
2. Выберите меню Extensions ▸ Language Manager (Расширения ▸ Менеджер языков — для русской версии программы).
3. На вкладке Site (Сайт) в таблице списка установленных языков поставьте переключатель с зависимой фиксацией в положение Russian (CIS).
4. Нажмите кнопку Default (По умолчанию) в панели инструментов справа вверху, после чего страница обновится и в поле Default (По умолчанию) появится звездочка напротив русского языка.
5. Перейдите на вкладку Administrator (Административная панель).
6. Установите русский язык (точно так же, как и в пп. 3 и 4).
7. Выйдите из режима администрирования сайта, нажав в меню кнопку Logout (Выйти).

Итак, русский язык установлен в качестве языка по умолчанию для пользовательской и административной частей сайта. Когда вы войдете в пользовательскую часть сайта, не пугайтесь, что все тексты остались на английском языке, — вы установили демо-версию, а она работает на английском языке. Русификация пользовательской части распространяется только на стандартные расширения сайта, но никак не на статьи. Например, слева, под главным меню, форма регистрации пользователя работает на русском языке. В административной части также русифицированы только основные элементы управления. Практически все описания к модулям и параметрам, а также справка останутся на английском языке.

Далее в книге предполагается, что вы русифицировали Joomla!.

Пользователи и группы

Как и в любой системе, требующей разграничения доступа, в Joomla! есть возможность назначить пользователям и группам различные права. Для предоставления доступа к странице управления пользователям выполните следующие действия.

1. Войдите в административную часть сайта и зарегистрируйтесь в ней с правами администратора, как описывалось выше.
2. Выберите в меню Сайт ▸ Пользователи либо на панели управления значок Пользователи — в результате откроется страница менеджера пользователей (рис. 4.11).

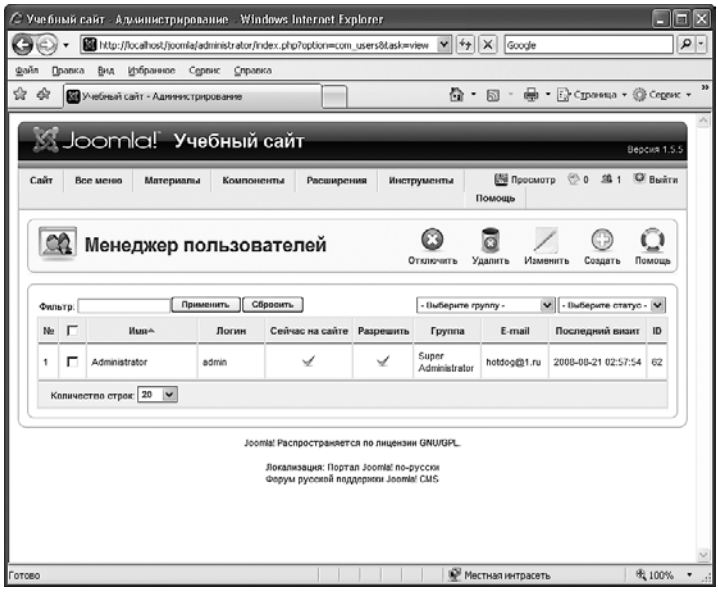


Рис. 4.11. Менеджер пользователей и групп

Управлять группами из административной части в Joomla! нельзя, однако имеющихся групп, как правило, хватает для выполнения большинства задач. Более того, при желании можно управлять и правами группы, но для этого нужно знать как минимум язык PHP.

Рассмотрим группы, которые предоставляет CMS Joomla!.

- Super Administrator — группа, пользователи которой имеют доступ к любой части сайта и ко всем настройкам. Пользователей данной группы нельзя удалить или переместить в другую группу. При установке Joomla! создается пользователь admin, принадлежащий этой группе. Будьте внимательны, помещая в эту группу пользователей. Лучше ограничиться одним пользователем admin.
- Administrator — группа, пользователи которой имеют несколько урезанные права по сравнению с пользователями группы Super Administrator. В частности, они не могут:
  - изменять глобальные настройки сайта в меню Сайт ▸ Общие настройки;
  - проводить массовые рассылки по электронной почте;
  - редактировать и добавлять пользователей группы Super Administrator;
  - добавлять и изменять шаблоны сайта;
  - добавлять и изменять языковые настройки.
- Manager — группа, пользователи которой могут лишь управлять содержимым сайта. Им запрещается:
  - устанавливать модули и компоненты;
  - администрировать пользователей;
  - пользователи группы Super Administrator могут запретить им доступ к некоторым компонентам.
- Registered — группа, пользователи которой могут заходить на сайт с главной страницы сайта, получая доступ к какой-либо дополнительной информации, указанной администратором.
- Author (Автор) — группа, пользователи которой наследуют права группы Registered, но дополнительно к ней они могут изменять информацию, добавленную ими же. К примеру, пользователь может поместить на сайт свою статью и редактировать ее в дальнейшем, при этом другие пользователи данной группы этого сделать не смогут.

- Editor — группа, пользователи которой могут добавлять и изменять информацию, созданную любым пользователем (в эту группу включаются редакторы сайта).
- Publisher — группа, пользователи которой могут делать то же, что и пользователи группы Editor, однако они могут также размещать на сайте любую информацию.

На нем можно разместить форму регистрации новых пользователей. При этом каждый новый пользователь автоматически попадает в группу Registered. Чтобы включить или выключить регистрацию пользователей, выполните следующие действия.

1. Войдите в режим администрирования сайта с правами пользователя группы Super Administrator.
2. Выберите меню Сайт ▸ Общие настройки, по умолчанию должна открыться вкладка Сайт.
3. Перейдите на вкладку Система (рис. 4.12).

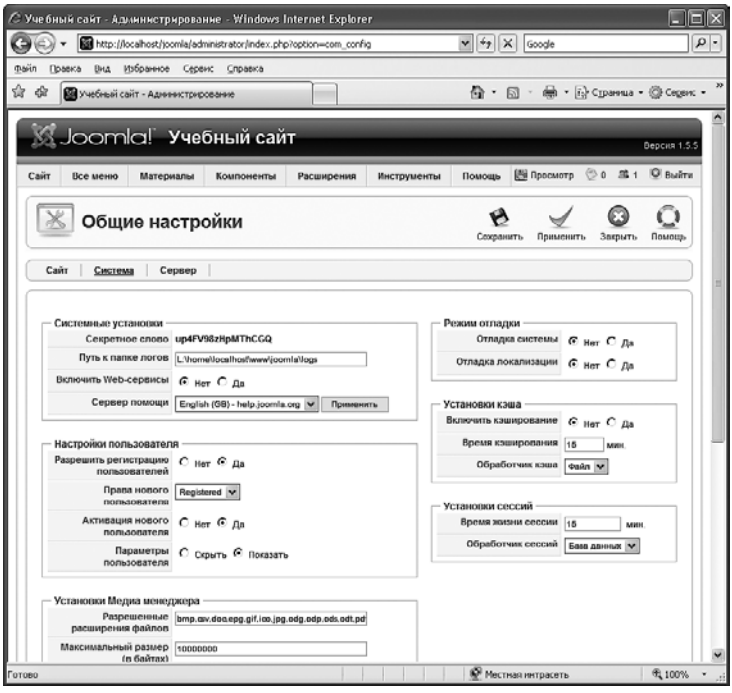


Рис. 4.12. Вкладка Система

4. В области Настройки пользователя установите переключатель напротив поля Разрешить регистрацию пользователей в положение Да. Параметр Права нового

пользователя позволяет установить, к какой группе будут автоматически причисляться новые пользователи.

Часто бывает, что зарегистрированного пользователя нужно активировать. Например, при создании электронной почты необходимо, чтобы каждый новый пользователь активировался. Это делается для того, чтобы предотвратить рассылку спама на ваш сайт. В то же время, например, интернет-магазину может и не потребоваться активация пользователя для доступа к корзине заказов. Поэтому в системе Joomla! предусматривается возможность включать и отключать активацию зарегистрировавшихся пользователей. За это отвечает параметр Активация нового пользователя (см. рис. 4.12). Если переключатель установить в положение Да, то каждый новый пользователь должен активироваться. Эта процедура заключается в том, что после ввода своих параметров (имени пользователя, логина, пароля и электронного адреса) пользователю высылается по электронной почте письмо со ссылкой для активации своей учетной записи. Ему нужно выполнить требуемые действия. Если переключатель установить в положение Нет, то любой зарегистрировавшийся пользователь сразу получает доступ к дополнительной информации сайта без активации.

5. После изменений учетных записей пользователей сохраните их, нажав кнопку Применить или Сохранить. Для отмены изменений и выхода из режима изменения параметров на панели инструментов вкладки Система нажмите кнопку Закрыть.

Компоненты

Компонент — это подпрограмма, которая формирует динамическую страницу сайта. О том, какой компонент надо показать на странице, система Joomla! узнает из адресной строки. Например, строка `http://localhost/joomla/index.php?option=com_sample` сообщает системе, что необходимо вывести на страницу компонент `com_sample`.

В CMS Joomla! компоненты служат для организации основной части сайта. Они могут организовывать форумы, интернет-магазины, новостные порталы, формы связи с администрацией сайта и многое другое. На страницах сайта компоненты располагаются на их центральной части и играют главную роль. Чтобы компонент попал на страницу сайта, его нужно установить, привязать к меню и опубликовать.

Для установки новых компонентов выполните следующие действия.

1. Откройте меню Расширения ▸ Установить/Удалить — в результате появится страница Менеджер расширений (рис. 4.13).

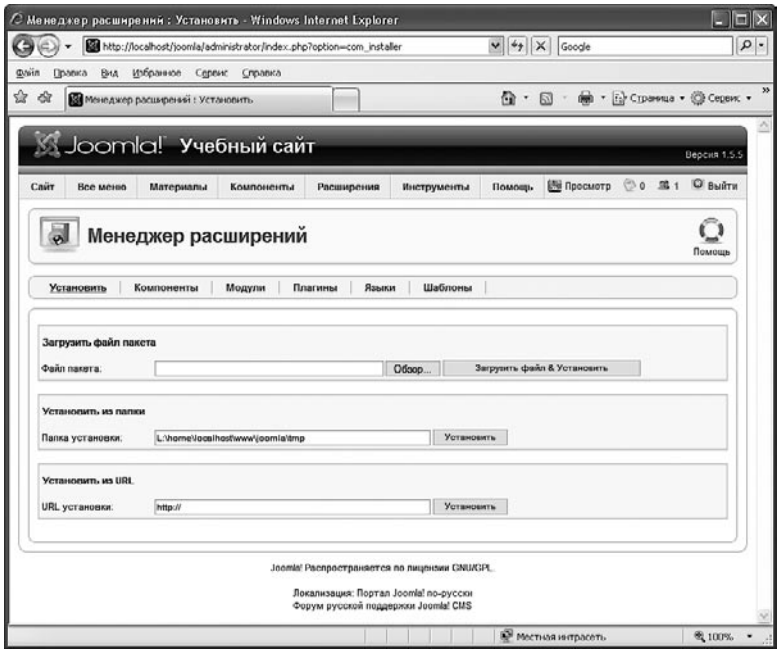


Рис. 4.13. Менеджер расширений

2. Если компонент поставляется в виде zip-архива, нажмите кнопку Обзор в области Загрузить файл пакета и выберите файл, в котором находится компонент.
3. Нажмите кнопку Загрузить файл & Установить — в итоге компонент загрузится на сайт, распакуется и установится в систему.
4. Если компонент поставляется в виде набора файлов, находящихся в каталоге, введите полный путь (локальный на вашем компьютере) к каталогу с компонентом в области Установить из папки и нажмите кнопку Установить.
5. Если вы желаете установить компонент из Интернета, воспользуйтесь областью Установить из URL. В поле URL установки введите полный адрес, откуда система скачает и установит компонент, и нажмите кнопку Установить.

Далее появится уведомление о том, что компонент успешно установился. Теперь получить доступ к администрированию компонента можно через меню Компоненты ▸ Имя компонента. Если установить компонент не удалось, система оповестит вас об этом.

Бывает, что вы удалили компонент, а потом поставили его обратно (например, при обновлении версии компонента или при его разработке и тестировании). Тогда

работа системы останавливается с ошибкой, что данная папка уже занята. В этом случае выполните следующие действия.

1. Войдите в подкаталог administrator в каталоге установки Joomla! на сервере. Затем войдите в каталог components, найдите папку с именем компонента и удалите ее вручную.
2. Вернитесь в каталог установки Joomla!, войдите в подкаталог components и удалите каталог с именем компонента.

В Joomla 1.5 можно также устанавливать компоненты более ранних версий, однако они могут работать некорректно. Чтобы установить компонент, написанный для более ранних версий, войдите в меню Расширения ▸ Менеджер плагинов, найдите плагин System — Legacy и включите его. После этого компонент можно устанавливать.

Для удаления компонента выполните следующие действия.

1. Выберите команду меню Расширения ▸ Установить/Удалить.
2. Перейдите на вкладку Компоненты — в результате откроется страница, представленная на рис. 4.14.

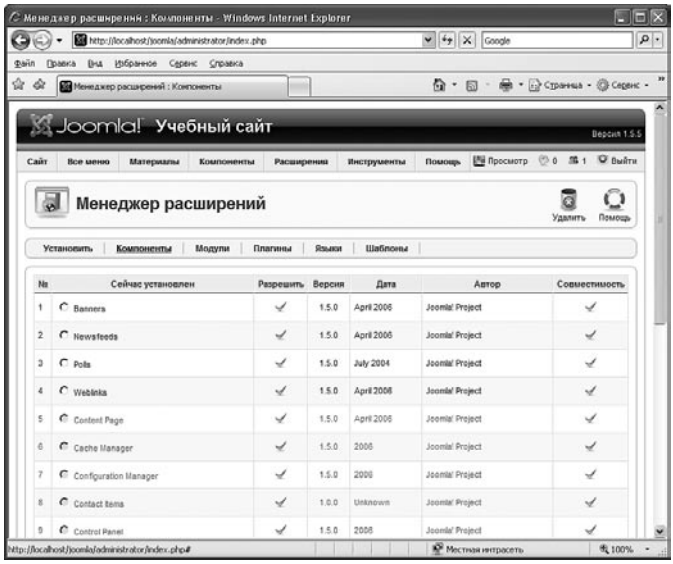


Рис. 4.14. Удаление компонента

3. Выберите переключатель с зависимой фиксацией напротив того компонента, который необходимо удалить.
4. Нажмите кнопку Удалить в верхнем правом углу страницы.

На странице удаления компонентов можно также включать и выключать компоненты. Для этого выполните следующие действия.

1. Если нужно выключить компонент, щелкните на зеленой галочке в поле Разрешить нужного компонента.
2. Если надо включить компонент, щелкните на крестике в поле Разрешить (см. рис. 4.14).

ВНИМАНИЕ

Некоторые компоненты удалять нельзя, поскольку они обеспечивают базовую функциональность Joomla!. Тем не менее при желании их можно удалить вручную. Делается это аналогично удалению «грязи» от старой версии компонента, однако данная операция может привести к неработоспособности всей системы.

Выключенные компоненты нельзя разместить на сайте. Допустим, компонент уже был размещен и к нему предоставлен доступ через меню сайта. Если пользователь будет пытаться получить доступ к соответствующему пункту меню сайта, то он увидит сообщение о том, что запрашиваемая страница не найдена.

Модули

Модуль — это подпрограмма, написанная на PHP или JavaScript. Визуально она представляет собой прямоугольник, который может размещаться практически в любой части сайта — там, где укажет администратор или автор страницы сайта. Модули позволяют показывать баннерную рекламу, анонсы новостей, организуют формы входа пользователей на сайт, в почту и многое другое. Модуль сам по себе существовать не может: он всегда привязывается либо к меню, либо к странице сайта.

Эти подпрограммы бывают двух типов: модули сайта и модули административной части сайта. Модули сайта публикуются на страницах сайта, к ним имеют доступ пользователи сайта. Административные модули предназначаются для расширения функциональности административной части сайта. К модулям можно открывать и закрывать доступ таким же образом, как и для компонентов. Можно разрешить доступ к модулям отдельным группам пользователей. Настройки модулей хранятся, как правило, в xml-файлах, которые являются частями модуля. На одной странице может находиться несколько модулей.

В более ранних версиях Joomla! существовали отдельные страницы для установки компонентов, модулей, шаблонов и фильтров. Теперь все делается на одной странице, поэтому установка модулей идентична установке компонентов. Исходя из этого мы не будем ее рассматривать, а перейдем к изучению администрирования модулей.

Рассмотрим, как осуществляется администрирование модулей.

Войдите в режим администрирования сайта и выберите команду меню Расширения ▸ Менеджер модулей — в результате откроется страница менеджера модулей (рис. 4.15).

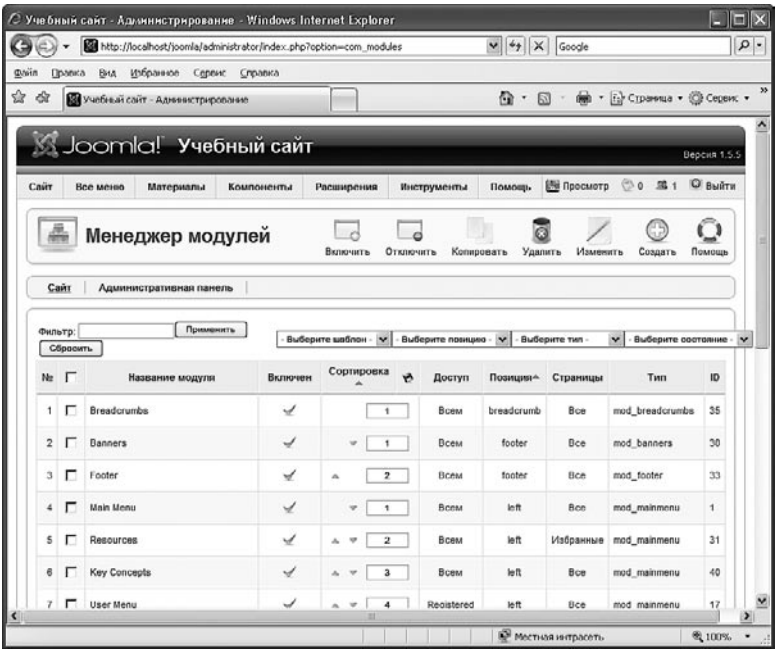


Рис. 4.15. Менеджер модулей

Как было сказано выше, модули делятся на модули сайта и модули административной части сайта. Для переключения между этими двумя большими группами используйте вкладки Сайт и Административная панель (см. рис. 4.15).

В верхней части страницы располагается панель инструментов для управления некоторыми параметрами модулей (рис. 4.16).

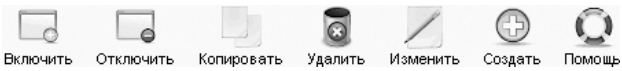


Рис. 4.16. Панель управления

- Включить — разрешить использовать группу модулей (о том, как выделить группу модулей, читайте ниже).
- Отключить — запретить использовать группу модулей.

- Копировать — создать копии выделенных модулей; имена скопированных модулей начинаются с `Copy of`, затем следует имя оригинального модуля.
- Удалить — удалить выделенную группу модулей.
- Изменить — вызвать страницу редактирования выбранного модуля. Если выделено больше одного модуля, появится страница редактирования параметров только для первого модуля в этом выделении.
- Создать — добавить новый экземпляр выделенных модулей.
- Помощь — показать справку.

В центральной части страницы размещается таблица со списком модулей.

- № — порядковый номер модуля.
- Второй столбец — элементы управления с независимой фиксацией для выбора модуля, над которым необходимо будет проводить групповые операции.
- Название модуля — соответственно, название модуля.
- Включен — разрешение использования модуля. Он может появиться на сайте только в том случае, если он разрешен.
- Сортировка — порядок следования модулей на странице или блоке сайта.
- Доступ — уровень доступа пользователей к модулю. Он может принимать следующие значения:
  - Всем (Public) — доступ разрешается всем посетителям сайта;
  - Registered (Зарегистрированные) — доступ к модулю имеют только зарегистрированные пользователи;
  - Special (Специальный) — доступ получают только пользователи с уровнем доступа Author (Автор) и выше.
- Позиция — место расположения модуля на странице. Оно определяется шаблоном, который использует сайт (шаблоны мы рассмотрим ниже).
- Страницы — пункты меню сайта, в которых будет располагаться модуль.
- Тип — системное имя модуля, определяемое разработчиками. Как правило, системное имя модуля совпадает с именем архива, в котором модуль располагался до установки.
- ID — внутренний уникальный номер модуля, генерируемый автоматически при установке нового модуля в систему. Он используется системой Joomla!, например, для внутренних ссылок. Изменять значение этого параметра нельзя.

Для настройки параметров модуля щелкните в столбце Название модуля по имени модуля — в результате откроется страница настройки параметров модуля (рис. 4.17).

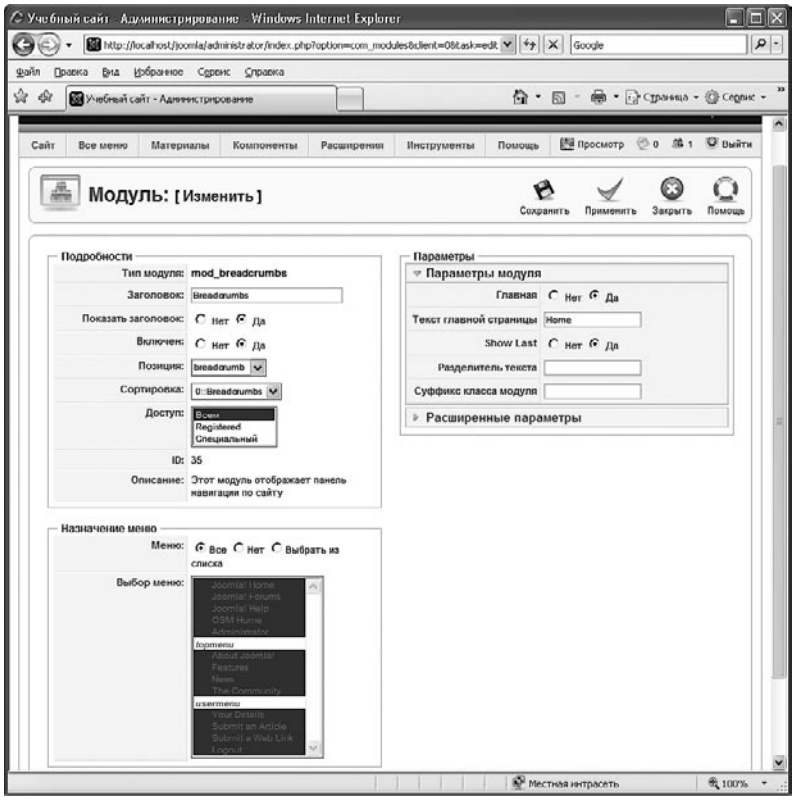


Рис. 4.17. Параметры модуля

В области Подробности размещаются основные параметры модуля.

- Тип модуля — системное имя модуля.
- Заголовок — заголовок модуля, который будут видеть пользователи.
- Показать заголовок — определяет, показывать заголовок модуля или нет.
- Включен — устанавливает, разрешается показывать модуль на сайте или нет.
- Позиция — положение модуля на странице сайта. Оно задается значением из раскрывающегося списка. Перечень значений определяется текущим шаблоном сайта. Можно также указать тег в статье `{loadposition xxx}`, где `xxx` — это уникальное имя, которое будет помещаться в раскрывающийся список. Тогда везде, где встречается тег `{loadposition xxx}`, он будет заменяться в статье на вызов модуля.

СОВЕТ



Чтобы разработчик смог увидеть, в какой части сайта располагается его модуль, необходимо добавить параметр tp=1 в конце адресной строки вызываемой страницы сайта. Например, адрес главной страницы сайта http://localhost/joomla позволяет увидеть, в каких частях страницы располагаются модули. Главная страница вызывается следующим образом: http://localhost/joomla?tp=1. В результате появится главная страница, отображающая названия блоков, в которых содержатся модули.

- Сортировка — порядок показа модулей. Этот параметр будет иметь значение только тогда, когда в одной и той же позиции (поле Позиция) показывается несколько модулей. Другими словами, если несколько модулей в поле Позиция имеют одно и то же значение, то эти модули будут выводиться в соответствии с порядковым номером модуля в поле Сортировка.
- ID — внутренний уникальный номер модуля. Присваивается при добавлении нового модуля в систему. Изменить его нельзя.
- Описание — краткое описание модуля, задаваемое разработчиками модуля.

В области Назначение меню необходимо указать, на каких страницах сайта будет демонстрироваться данный модуль. В этой области имеются следующие элементы.

- Меню — этот переключатель определяет, в каких меню будет показываться модуль. Его можно установить в одно из трех положений.
  - Все — модуль будет демонстрироваться при выборе любого пункта меню. По умолчанию этот параметр устанавливается для модуля `mod_mainmenu`, который отвечает за вид главного меню сайта. Таким образом, главное меню будет показываться при нахождении на любой странице сайта.
  - Нет — модуль не будет показываться в меню сайта. Этот параметр следует указывать для модулей, которые должны показываться только в отдельных статьях сайта.
  - Выбрать из списка — модуль будет показываться только на выбранных страницах сайта. Список страниц указывается в параметре Выбор меню. Если в нем выбрать все строки, то вы получите полный аналог параметра Все.
- Выбор меню — список пунктов меню, при выборе которых будет демонстрироваться модуль. Этот параметр доступен только в том случае, если переключатель Меню установлен в положение Выбрать из списка. Для выбора нескольких пунктов меню, в которых должен демонстрироваться модуль, нажмите кнопку Ctrl и выделите их кнопкой мыши.

Область Параметры индивидуальна для каждого модуля. Список параметров в ней определяют разработчики модулей.

Нажмите кнопку Сохранить или Применить на панели инструментов, чтобы сохранить внесенные изменения в параметры модулей.

Итак, модули являются небольшими вставками в различные точки сайта. Они существенно влияют на функциональность сайта. Именно модули «оживляют» сайт: позволяют организовать опросы, показ анонсов новостей, погоды, установить миниформы для входа зарегистрированных пользователей и многое другое. Таким образом, грамотное расположение модулей на сайте, написанных с помощью Joomla!, является весьма важной задачей любого веб-разработчика.

Шаблоны

Шаблон — это набор стилевых таблиц (CSS), рисунков и html-страниц, в котором определяются вызовы php-функций, изменяющих общий вид сайта. Оформление сайта, а именно: цветовое решение, фоновые рисунки, положение меню, некоторых элементов управления — определяется именно в шаблоне. Главная функция шаблонов — управление внешним видом сайта без изменения исходных кодов компонентов, модулей и плагинов, то есть разделение программной и интерфейсной частей сайта. После инсталляции Joomla! в системе по умолчанию встраиваются три шаблона: `beez`, `JA_Purity` и `rhuk_milkyway`. Установка шаблонов в систему происходит точно так же, как и установка компонентов. Исходя из этого описывать ее здесь нет смысла (при возникновении вопросов обращайтесь к соответствующему подразделу по установке компонентов). Рассмотрим, как следует переключать шаблоны на сайте.

1. Откройте браузер на основной странице сайта. В нашем случае это http://localhost/joomla. Обратите внимание на общее оформление сайта (рис. 4.18).
2. Перейдите в режим администрирования сайта. В нашем случае это адрес http://localhost/joomla/administartor.
3. Выберите меню Расширения ▸ Менеджер шаблонов. В результате вы увидите окно, представленное на рис. 4.19.
4. Обратите внимание, что напротив шаблона `rhuk_milkyway` установлена желтая звездочка (см. рис. 4.19): этот шаблон устанавливается по умолчанию. Установите переключатель с зависимой фиксацией напротив любого другого шаблона.
5. Нажмите кнопку По умолчанию на панели инструментов.
6. После перезагрузки окна убедитесь, что напротив выбранного вами шаблона появилась желтая звездочка.

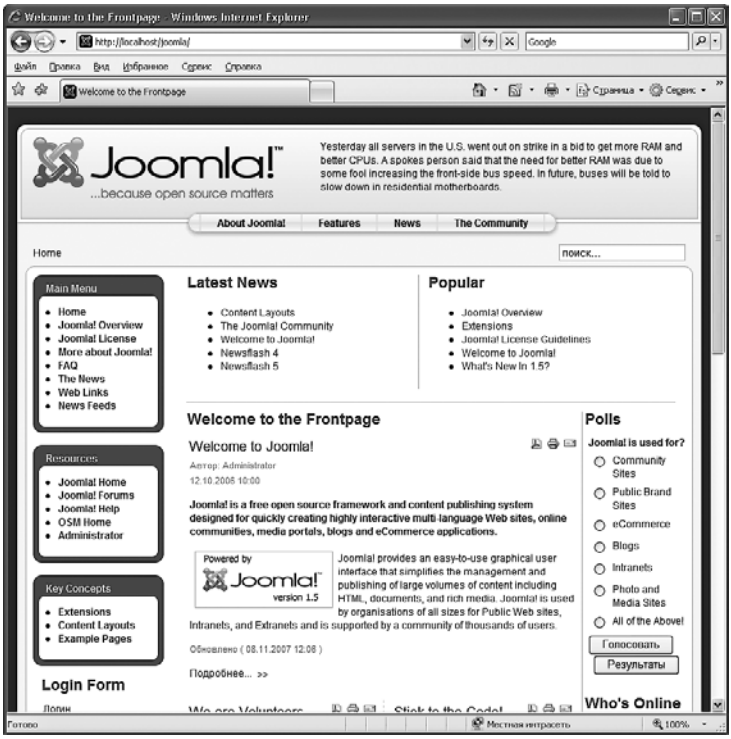


Рис. 4.18. Сайт с шаблоном rhuk_milkyway

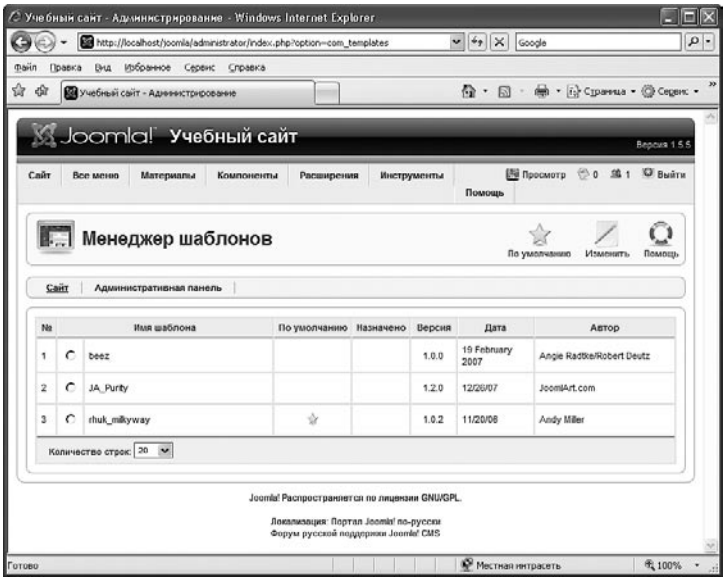


Рис. 4.19. Менеджер шаблонов

7. Перейдите на главную страницу сайта. Обратите внимание, что, хотя общий вид сайта (рис. 4.20) изменился по сравнению с предыдущим (см. рис. 4.18), основные элементы управления (меню, строка поиска и др.) остались на тех же местах.



Рис. 4.20. Сайт с шаблоном beez

Как видите, шаблоны позволяют очень просто менять внешний вид сайта, при этом не вмешиваясь в работу других расширений.

Плагины

Плагин (*мамбот*, или *фильтр*) — это подпрограмма, которая на входе получает код страницы, обрабатывает его и на выходе выдает другой код. Это очень удобный инструмент, с помощью которого можно обработать любую страницу перед показом ее пользователю. Например, можно организовать автоматический перевод сайта на другой язык, отлов и удаление нецензурных выражений на страницах (в частности, на форумах). Вы можете даже добавить свои теги к уже имеющимся в языке HTML. Совместимость Joomla! версии 1.5 с более ранними версиями программы реализуется именно с помощью плагинов.

Установка плагинов ничем не отличается от установки компонентов. Рассмотрим, как осуществляется управление плагинами.

1. Войдите на страницу администрирования сайта.
2. Выберите меню Расширения ▸ Менеджер плагинов.
3. Проверьте, включен ли плагин в столбце Включен.
4. Щелкните на названии плагина для перехода на страницу настройки его параметров.

   Настройка плагина идентична настройкам модуля. В этом легко убедиться, взглянув на рис. 4.21.

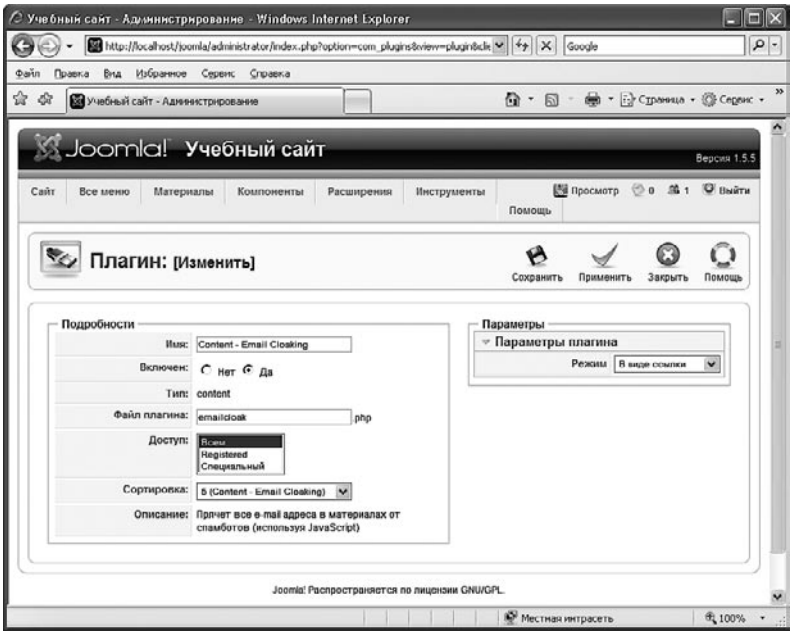


Рис. 4.21. Параметры плагина

В области Подробности находятся общие для всех плагинов параметры.

- Имя — имя плагина, которое показывается в Менеджере плагинов.
- Включен — здесь показывается, включен или выключен плагин.
- Тип — уникальное название плагина в системе. Внутренние обращения к плагину происходят именно по данному параметру. Этот параметр изменить нельзя.
- Файл плагина — имя файла плагина. Каждый плагин состоит из двух файлов: один с расширением php, а другой — с `xml`. В php-файле хранится исполняемый код плагина, а в xml-файле — параметры плагина, настраиваемые в области Параметры.

- Доступ — здесь указывается, какие пользовательские группы имеют доступ к данному плагину.
- Сортировка — порядок следования плагина при показе его в Менеджере плагинов.
- Описание — краткое описание плагина.

Область Параметры содержит настройки, индивидуальные для каждого плагина, поэтому сейчас их рассматривать не имеет смысла. Скажу лишь, что параметры данной области хранятся в xml-файле плагина.

Итак, CMS Joomla! предлагает широкий спектр расширений задач вашего сайта, что позволяет гибко и очень быстро настраивать его практически под любые нужды. Четкая модульная структура дает возможность существенно изменять внешний вид сайта, при этом вы не лишитесь права пользования каким-либо функциями.

CMS Drupal

Drupal, так же как и Joomla!, является системой управления сайтом, выпущенной под лицензией GPL. В отличие от Joomla!, CMS Drupal может работать не только с сервером баз данных MySQL, но и с объектно-реляционной СУБД PostgreSQL. Эту систему создал Дрис Байтаерт (Dries Buytaert) в 2000 г. Со временем к проекту присоединились десятки программистов по всему миру. Начали появляться национальные общества пользователей этой системы. Первым таким обществом стал русский проект Drupal.ru. На сегодня эта CMS существует более чем на трех десятках языков. В отличие от Joomla!, в Drupal нет единой системы оформлений тем, однако взамен можно работать с «движками», использующими шаблоны, и писать собственные темы оформления, обращаясь напрямую к API Drupal. В данном проекте несколько иначе организуется работа сайта, хотя общие принципы остаются теми же. Разработчику в Drupal нужно быть готовым к плотной работе с API системы, так как эта система требует больше программирования сценариев, чем Joomla!.

Для работы системы необходимо, чтобы на сервере были установлены и работали следующие средства:

- PHP версии 4.3.3 или выше;
- веб-сервер;
- СУБД MySQL или PostgreSQL.

Итак, приступим к изучению системы. Для начала нужно установить ее на компьютер.

Установка

Для установки системы выполните следующие действия.

1. Скачайте файл `drupal-6.3.tar.gz` с сайта Drupal.ru.
2. Распакуйте в любой подкаталог (к примеру, drupal) корневого каталога своего сайта.
3. В адресной строке браузера наберите http://localhost/drupal, после чего вы увидите страницу установки системы (рис. 4.22).

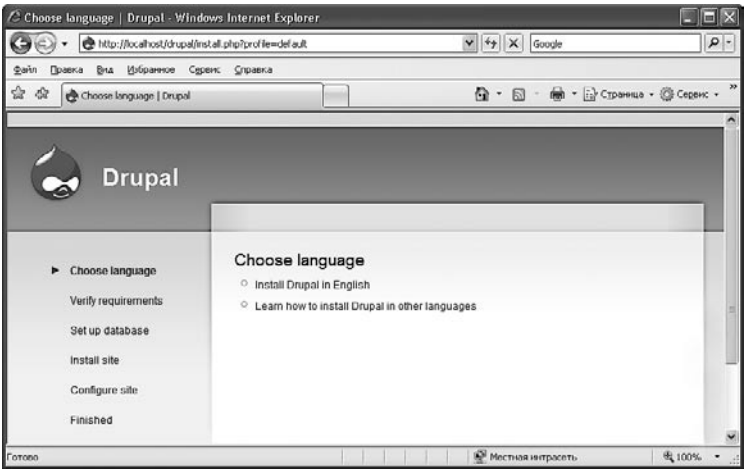


Рис. 4.22. Установка Drupal

4. Выберите пункт Install Drupal in English (Установить Drupal на английском языке).

ВНИМАНИЕ

Иногда при установке инсталлятор показывает сообщение об ошибке, в котором говорит, что нет файла settings.php, необходимого для работы Drupal. Инсталлятор записывает параметры, указанные при установке, в этот файл. Для решения данной проблемы войдите в каталог, в который вы распаковали архив с Drupal (к примеру, в каталог Y:\home\localhost\www\drupal\). Сначала наберите команду
cd Y:\home\localhost\www\drupal\sites\default,
а затем команду
copy nul settings.php.
После этого будет создан файл settings.php. Он обязательно должен быть пустым и иметь нулевую длину.
Для UNIX-подобных ОС файл settings.php необходимо создать соответствующими командами. Параметры доступа к этому файлу должны быть указаны кодом 440, например командой chmod 440. Данный набор параметров означает, что файл доступен только для чтения пользователями и группами.

5. На следующей странице (рис. 4.23) необходимо указать конфигурацию СУБД. Создайте в MySQL базу данных с именем `drp` (создать базу в MySQL можно с помощью онлайнового администратора phpMyApdmin). В поле Database name укажите `drp` — имя нашей учебной базы данных, в поле Database username — `root`, а в Database password ничего указывать не надо. Чтобы указать такие параметры, как имя сервера, на котором установлен сервер БД, порт и префикс таблиц системы Dropal, необходимо щелкнуть кнопкой мыши по надписи Advanced options и настроить параметры в полях Database host, Database port и Table prefix. В нашем случае оставим значения, предложенные по умолчанию. Нажмите кнопку Save and continue.

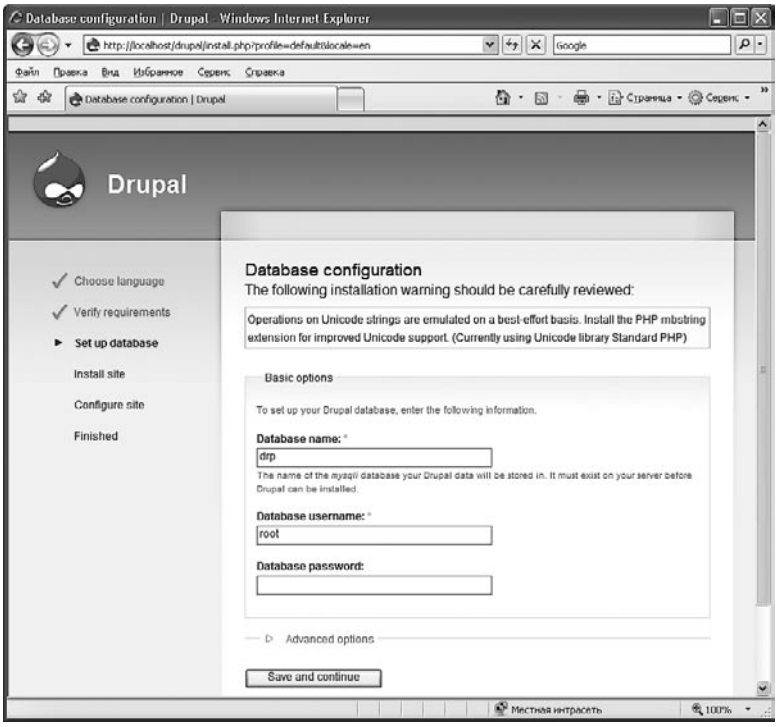


Рис. 4.23. Параметры СУБД

6. В следующем окне (рис. 4.24 и 4.25) необходимо указать параметры самого сайта.

   В эти поля введите следующую информацию:

   - Site name — название вашего сайта;
   - Site e-mail address — электронный адрес, который будет указываться в поле От кого при рассылке электронной почты с сайта.



Рис. 4.24. Настройки сайта

В области Administrator account укажите данные об администраторе сайта:

- Username, Password и Confirm password — имя пользователя, пароль для доступа администратора к сайту и подтверждение этого пароля;
- E-mail address — адрес электронной почты администратора, который будет отображаться на сайте, чтобы пользователи имели возможность связаться с администратором сайта.

В области Server settings настраиваются общие параметры сервера, обслуживающего сайт.

- Default time zone — зона времени, чтобы сервер смог преобразовать данные о времени в свое локальное время. Для выбора временной зоны откройте раскрывающийся список, в котором показываются текущие дата и время.
- Clean URLs — здесь оставьте значение, которое выбрала программа установки. Этот параметр влияет на то, как будут передаваться страницам параметры: через метод GET (в адресной строке) или через метод POST.

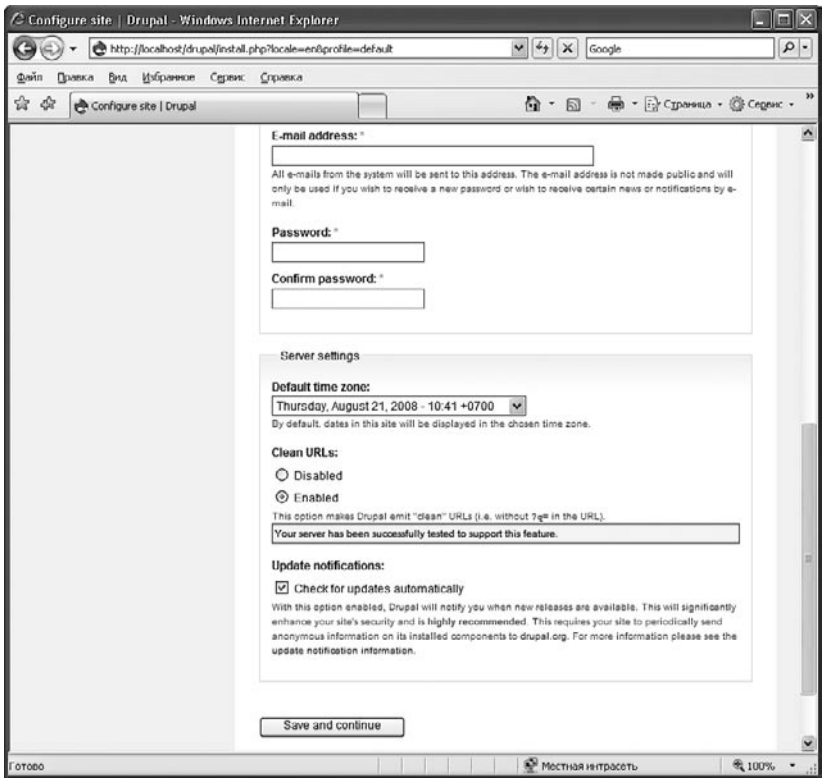


Рис. 4.25. Настройки сайта

- Update notifications — установите флажок, если вы хотите получать обновления к данной версии Drupal. Тут надо быть внимательным: с одной стороны, обновления могут помочь, а с другой — ваша версия сайта может оказаться чувствительной к какой-то конкретной версии Drupal. Поэтому решение о необходимости обновлений нужно принимать обдуманно.

Поля, отмеченные звездочкой, являются обязательными для заполнения.

После установки и проверки всех настроек сайта нажмите кнопку Save and continue.

7. Последняя страница установки (рис. 4.26) информирует вас о том, что система Drupal успешно установилась на сервер. Теперь получить доступ к сайту можно по адресу http://localhost/drupal. Можно также нажать ссылку your new site — она выведет вас на тот же адрес.

Как вы видите, установка Drupal проще, чем установка Joomla!. Однако это не означает, что данная система проще. При создании сайта в каждом конкретном случае

необходимо внимательно изучить, с использованием какой системы лучше его организовать. Например, Joomla! позволяет легко менять темы сайта, в то же время у Drupal существуют аналогичные компоненты, к тому же оформление можно задать непосредственно при написании сайта (в итоге это может оказаться более гибким средством управления сайтом).

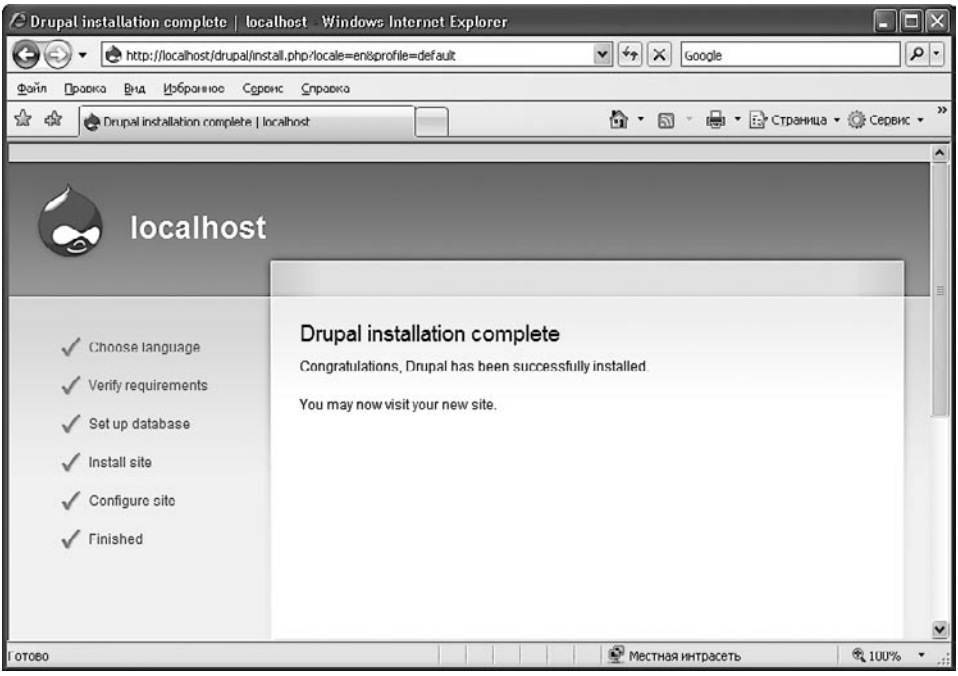


Рис. 4.26. Завершение установки

Возможные проблемы при установке

Рассмотрим две проблемы, с которыми вы можете столкнуться при установке CMS Drupal.

- Если во время установки вы получаете сообщение Drupal already installed (Drupal уже установлен), перейдите в каталог, в который был распакован архив с системой, найдите там файл `.htaccess`, откройте его для редактирования, найдите параметр `ExpiresActive` и поставьте его в положение `Off`.
- Если в начале установки вместо первой страницы инсталлятора появляется пустая страница, то, скорее всего, не хватает памяти на сервере, на который вы устанавливаете Drupal. Откройте файл `.htaccess` и добавьте строку `php value memory limit 32M`. Вместо `32M` можно вписать любое количество

памяти, которое не жалко отдать для работы интерпретатора PHP. Слишком большое значение устанавливать здесь нет смысла.

Администрирование

Для входа в режим администрирования сайта выполните следующие действия.

1. Войдите на главную страницу сайта (в нашем случае это http://localhost/drupal).
2. Слева в поле Username (Пользователь) введите `admin`, в поле Password (Пароль) — пароль администратора, указанный при установке системы.
3. Нажмите кнопку Log in (Войти).

В режиме администрирования сайта можно управлять правами пользователей, включать и выключать модули, редактировать меню и страницы сайта и многое другое. Рассмотрим наиболее важные функции административной части сайта.

Пользователи и группы

Как и в любой системе с ограниченным доступом, в CMS Drupal можно управлять списком пользователей и их правами. Чтобы получить возможность управления списком пользователей, выполните следующие действия.

1. Зарегистрируйтесь на сайте как администратор.
2. Выберите команду меню Administer ▸ User managment ▸ Users — в результате вы увидите страницу управления пользователями и их правами (рис. 4.27).

   Эта страница имеет два режима работы:

   - List — управление учетными записями пользователей;
   - Add user — добавление нового пользователя.

Чтобы увидеть пользователей, которым разрешается или запрещается доступ к определенным модулям, выполните следующие действия.

1. В области Show only users where (Показать пользователей, у которых) в раскрывающемся списке permission is (разрешается) выберите модуль, к которому искомые пользователи либо имеют доступ, либо не имеют.
2. В раскрывающемся списке status (статус) выберите пользователей, которым доступ разрешается (active) или же запрещается (blocked) для данного модуля.
3. Нажмите кнопку Filter (Отсортировать) — она отсортирует и покажет только тех пользователей, которые удовлетворяют заданным условиям выборки.

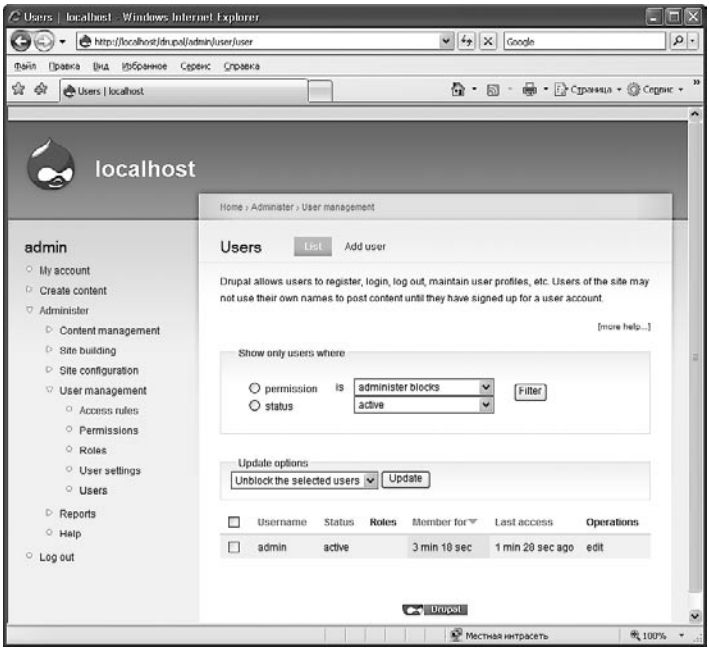


Рис. 4.27. Список пользователей

В нижней части страницы (см. рис. 4.27) находится таблица со списком пользователей. В этой таблице есть следующие поля:

- Username (Логин/Имя пользователя) — имя пользователя в системе;
- Status (Состояние) — статус пользователя (активен или заблокирован);
- Roles (Правила/Группа) — группа, которой принадлежит пользователь;
- Member for (Срок существования) — время, прошедшее с момента регистрации пользователя в системе;
- Last access (Последний доступ) — время, прошедшее с момента последней отправки пользователю страниц сайта;
- Operations (Действия) — список операций над учетной записью пользователя, который вам доступен.

Если нужно произвести какую-либо операцию над несколькими пользователями, то это удобно делать с помощью групповых операций. Для этого выполните следующие действия.

1. Отфильтруйте список пользователей по какому-либо признаку в области Show only users where.

2. Включите выключатель с независимой фиксацией напротив имени каждого пользователя, над которым необходимо произвести групповую операцию. Если надо выбрать или снять выделение со всех пользователей, щелкните по переключателю с независимой фиксацией в заголовке таблицы.
3. В области Update options (Групповые операции) в раскрывающемся списке выберите операцию, которую необходимо произвести над отфильтрованным списком пользователей:
   - Unblock the selected users (Разблокировать выделенных пользователей) — разблокировать выделенных пользователей, то есть установить для каждого пользователя статус `active`;
   - Block the selected users (Заблокировать выделенных пользователей) — заблокировать выделенных пользователей, то есть установить для каждого пользователя статус `blocked`;
   - Delete the selected users (Удалить выбранных пользователей) — удалить выделенных пользователей.
4. Нажмите кнопку Update (Обновить) — в результате над выделенными пользователями выполнится требуемая операция.

Если щелкнуть по имени пользователя в столбце Username (Пользователь), то вы увидите страницу с историей данного пользователя (когда он входил в систему, статус и другие данные).

Для управления группами пользователей выполните следующие действия.

1. Войдите на сайт с правами администратора.
2. Выберите меню Administer ▸ User managment ▸ Roles (Администрирование ▸ Пользователи ▸ Группы) — в итоге откроется страница управления группами пользователей. При установке Drupal по умолчанию создаются только две группы пользователей:
   - anonymous user (анонимный пользователь) — незарегистрированные пользователи;
   - authenticated user (зарегистрированный пользователь) — зарегистрированные пользователи.
3. В таблице списка групп выберите ссылку edit permissions (редактировать права) для группы, права которой вы желаете изменить. Откроется страница управления правами группы (рис. 4.28).

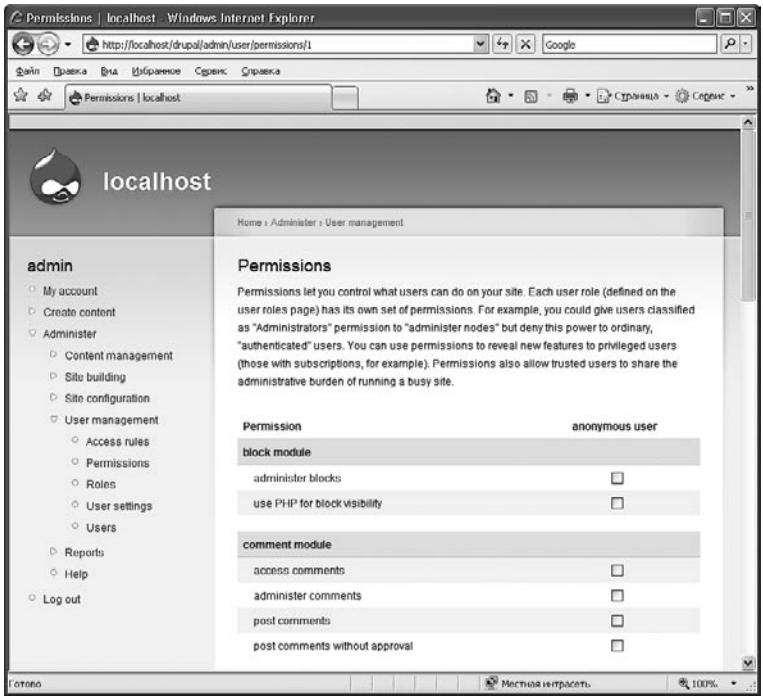


Рис. 4.28. Настройка прав группы

Для изменения прав доступа к модулям для группы выполните следующие действия.

1. Зайдите на страницу управления правами группы.
2. Напротив каждого модуля (см. рис. 4.28), к которому нужно открыть доступ для пользователей данной группы, установите флажок в переключателе с независимой фиксацией. Для модулей, доступ к которым нужно запретить, снимите этот флажок.
3. В самом низу страницы нажмите кнопку Save permissions (Сохранить права) — в результате все внесенные изменения сохранятся и вы вернетесь в начало страницы управления правами группы.

Модули

Функциональность CMS Drupal определяется ее модулями. Понятие *модуля* здесь включает в себя понятия *компонента*, *модуля* и *плагина* в Joomla!. Подключая и настраивая модули, можно добиться необходимой функциональности своего сайта. Все модули находятся в подкаталоге modules каталога установки Drupal.

После установки Drupal к системе подключаются только самые необходимые модули, поэтому для расширения функциональности сайта нужно выполнить следующие действия.

1. Перейти в режим администрирования сайта путем набора адреса http://localhost/drupal и регистрации на сайте с правами администратора.
2. Подключить и при необходимости настроить нужные модули.

Поскольку каждый модуль имеет свои настройки, рассматривать, как настраивать конкретный модуль, здесь мы не будем. Разберем установку модулей в систему.

На главной странице администрирования (рис. 4.29) модули могут быть рассортированы либо по задачам — By task (По назначению), либо по названиям модулей — By modules (По именам модулей). Какой вид удобнее — решать вам как администратору сайта.

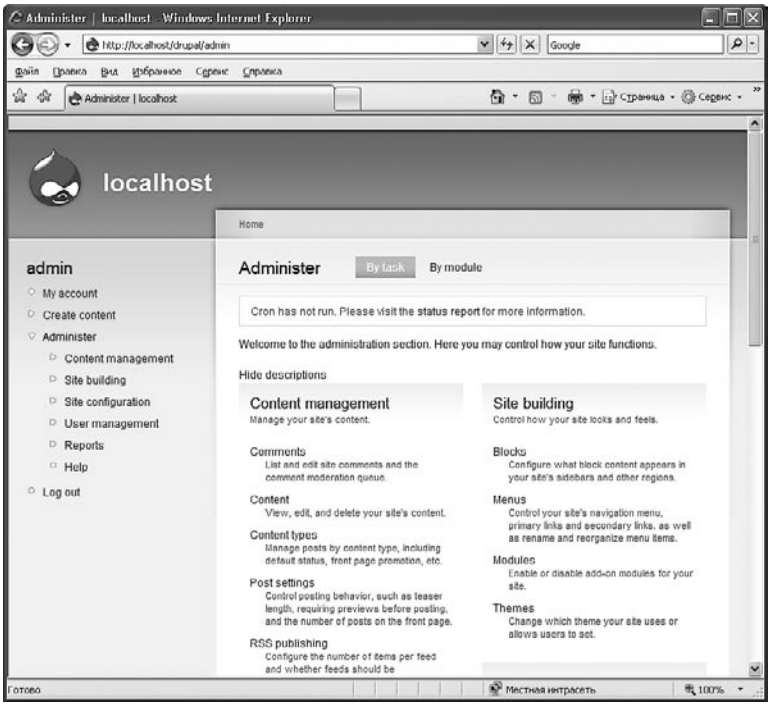


Рис. 4.29. Главная страница администратора

Установка модулей в Drupal сложнее, чем в Joomla!, поэтому трудно свести ее к некой общей процедуре установки. Необходимо изучить документацию по установке каждого модуля, которая находится либо в архиве модуля, либо на

сайте разработчика этого модуля. Тем не менее опишу типичные шаги установки модулей.

1. Скачайте архив модуля, например с официального сайта http://drupal.org.



ВНИМАНИЕ

Если имя файла модуля заканчивается на –dev, то этот модуль находится в стадии разработки и его безошибочная работа не гарантируется.

2. Распакуйте архив в отдельный каталог.
3. Обязательно ознакомьтесь с файлами `INSTALL.TXT` и `README.TXT`, в которых описываются действия, необходимые для установки модуля.
4. Скопируйте папку в системный каталог модулей `sites/all/modules`. Если каталог не существует, создайте его.
5. Перейдите на свой сайт в Drupal и выберите в главном меню Administer ▸ Site building ▸ Modules (Администрирование ▸ Построение сайта ▸ Модули). Найдите в списке модулей тот модуль, который вы устанавливаете (они разбиты по разделам, поэтому при поиске открывайте соответствующий раздел — в нем находится устанавливаемый модуль), и включите переключатель с независимой фиксацией напротив имени модуля (рис. 4.30).
6. Выберите команду меню Administer ▸ User managment ▸ Permissions (Администрирование ▸ Пользователи ▸ Доступ) и выставьте права доступа к модулю.
7. Наконец, выставьте необходимые настройки для модуля. Местоположение настроек модуля можно узнать из документации (файлы `README.TXT`, `INSTALL.TXT`) к модулю или на сайте производителя. Можно также выбрать в главном меню Administer ▸ Site configuration ▸ <имя модуля> (Администрирование ▸ Параметры сайта ▸ <имя модуля>).



СОВЕТ

Некоторые модули могут установиться только при наличии других модулей. Если требуемые модули не установлены, тогда, во-первых, снят переключатель, включающий модуль; во-вторых, в самом конце описания модуля будет написано Depends on: ‹имя требуемого модуля› (missing) (отсутствует). В этом случае необходимо скачать, установить и включить требуемый модуль — и надпись сменится на Depends on: ‹имя требуемого модуля› (enabled) (включен).

После подключения модуля к системе он, скорее всего, не появится на сайте сразу. Чтобы включить модуль, необходимо, как правило, вписать в код страницы тег, вызывающий его. В этом отношении подключение модуля Drupal аналогично подключению модуля или плагина в Joomla!. Чтобы выяснить, как вставить модуль в страницу, обращайтесь к документации по этому модулю.



Рис. 4.30. Установка модуля

Сайт. Любой сайт представляет собой информационный ресурс, поэтому центральное место в любом сайте занимают его страницы. В Drupal они бывают двух типов: *простая статическая страница* и *динамическая страница*. Простые страницы изменяются редко: как правило, они представляют собой статические страницы типа «О нас». Динамические страницы — это страницы, содержимое которых постоянно меняется. Этот тип страниц хорошо подходит для создания блогов, новостей, форумов, страниц, содержимое которых могут изменять посетители вашего сайта.

Рассмотрим, как создаются страницы обоих типов. Для перехода к созданию своих страниц сайта выполните следующие действия.

1. Зайдите на главную страницу своего сайта на Drupal.
2. Зарегистрируйтесь с правами администратора.
3. Выберите меню `Create content` (Создать материал) — в результате откроется страница выбора типа страницы, которую вы будете создавать (рис. 4.31).

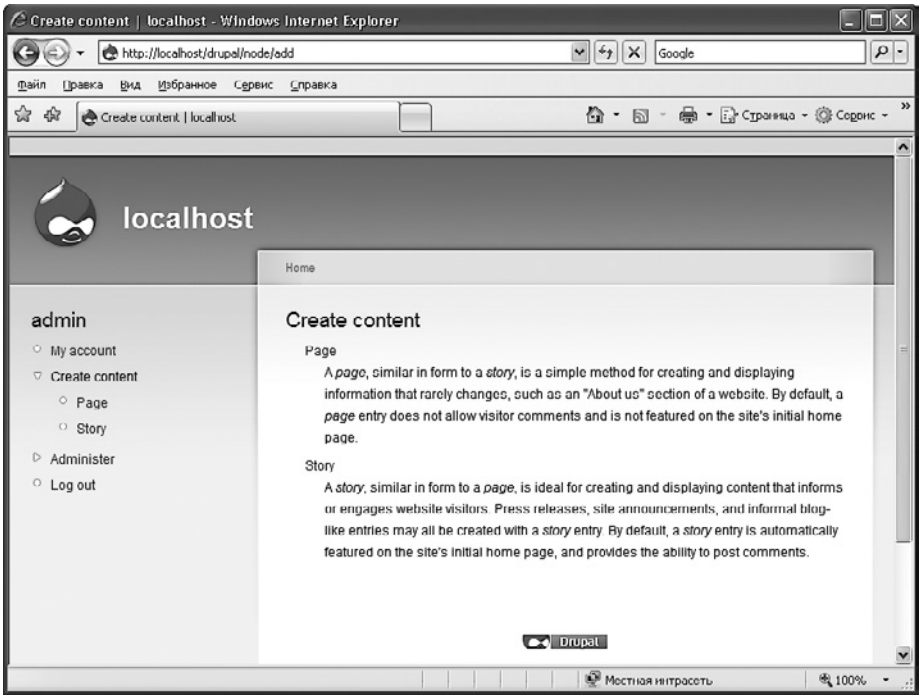


Рис. 4.31. Создание страницы

4. Статические и динамические страницы создаются одинаково, поэтому выберите любой пункт в меню Page (Статическая страница) или Story (Динамическая страница) в зависимости от того, какого типа страницу вы хотите создать. Откроется окно, в котором необходимо ввести код для страницы (рис. 4.32).

Рассмотрим, какие существуют параметры страницы и на что они влияют.

- Поле Title (Заголовок) — заголовок страницы, который будет отображаться при переходе на данную страницу.
- Поле Body (Содержимое) — содержимое страницы. Здесь может находиться как простой текст, так и HTML-код страницы. Кнопка Split summary at cursor позволяет разделить код на две части в месте нахождения курсора. Это может оказаться удобным, когда необходимо редактировать длинную страницу, при этом какую-то ее часть нужно постоянно видеть (например, таблицы).
- Область Menu settings (Параметры меню) управляет тем, в каких пунктах меню будет находиться страница. Щелкните по названию этой области — в результате развернется список ее параметров (рис. 4.33).

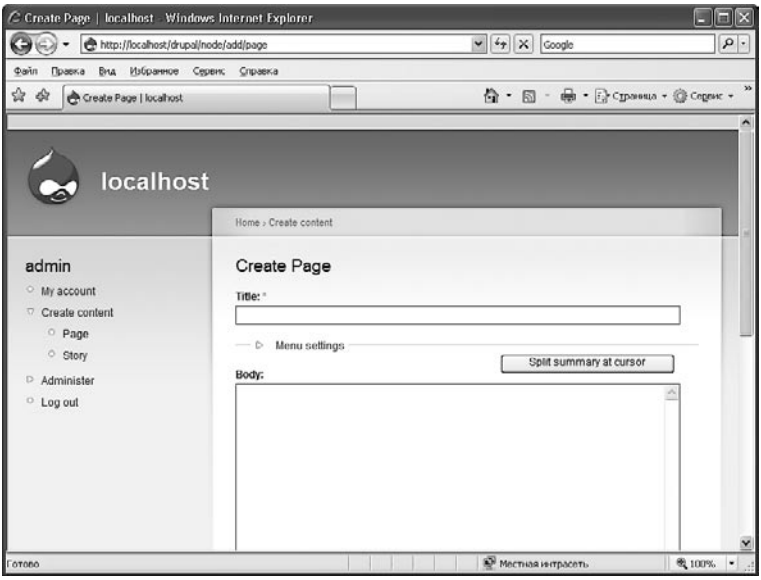


Рис. 4.32. Содержимое страницы

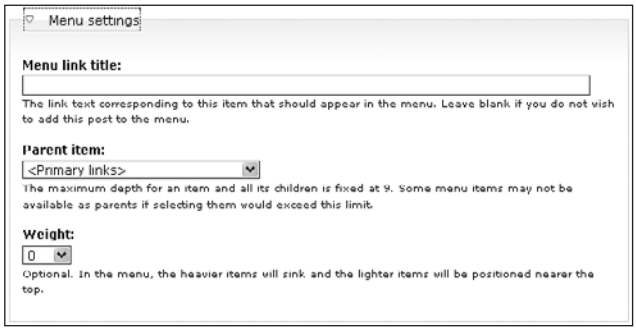


Рис. 4.33. Область с параметрами меню

Рассмотрим кратко эти параметры.

- Menu link title (Имя пункта меню) — название пункта меню, вызывающего данную страницу.
- Parent item (Родительский пункт меню) — раскрывающийся список пунктов меню, которые будут родительскими пунктами меню данной страницы. Таким образом можно выстраивать иерархию меню.
- Weight (Вес) — «вес» страницы. По сути, это порядок сортировки страниц в меню. Чем меньше значение этого параметра, тем выше располагается страница в иерархии меню.

- Область Input format (Форматировать на входе) управляет обработкой страницы перед отображением (рис. 4.34).

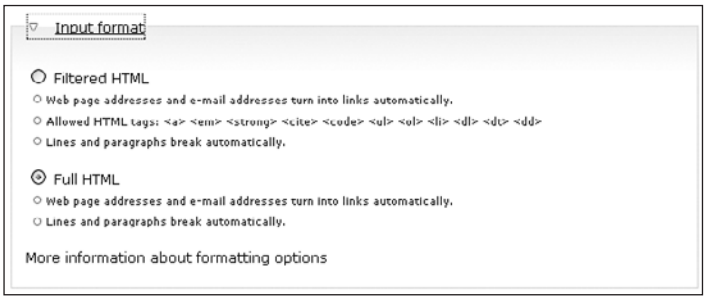


Рис. 4.34. Область настройки отображения страницы

Эта область имеет два параметра.

- Filtered HTML (Обрабатывать HTML) — если этот параметр установлен и в коде страницы (набирается в поле Body) встречается тег, подключающий модуль, то система в это место кода вставит модуль. Если вы хотите, чтобы модуль отображался на странице, подключите его к системе, вставьте вызывающий его тег в код страницы и включите этот параметр.
- Full HTML (Весь HTML) — просто отображает код страницы. Теги, вызывающие модули, не обрабатываются. Они воспринимаются как обычный HTML-код или текст.

- Область Comment settings (Комментарии) управляет комментариями к странице (рис. 4.35).



Рис. 4.35. Область настройки комментариев к странице

Рассмотрим кратко параметры этой области.

- Disabled (Выключить) — отключить комментарии к странице.
- Read only (Только чтение) — посетители сайта могут только читать комментарии к странице.
- Read/Write (Чтение/Запись) — посетители сайта могут добавлять и читать комментарии к странице.

- Область Authoring information (Автор) — данные об авторе страницы.

- Область Publishing options (Публикация) — параметры публикации страницы.

  Рассмотрим эти параметры.

  - Published (Опубликовать) — если включен данный параметр, страница публикуется на сайте.
  - Promoted to front page (Помещать на главную) — если он включен, страница публикуется на главной странице сайта.
  - Sticky at top of lists (Закреплять вверху списков) — если включен этот параметр, страница будет помещаться вверху сообщений на главной странице.

Меню. После того как вы создали на сайте страницы, они не сразу появятся на сайте. Пользователь должен иметь доступ не только к страницам, находящимся на главной странице сайта, но и к вашим страницам. Для этого необходимо разработать и создать систему меню. Чтобы понять, как организуется меню в Drupal, рассмотрим понятие *таксономии*. На сайте http://www.wikipedia.org/ дается следующее определение этого понятия.

Таксономия (гр. «расположение по порядку + закон») — иерархически выстроенная система целей и результатов от простой к сложной системе.

Таксономия в Drupal используется для классификации материалов сайта. Их можно группировать и разделять по разным признакам.

Для редактирования меню выполните следующие действия.

1. Зайдите в режим администрирования сайта.
2. Выберите меню Administer ▸ Site building ▸ Menus ▸ Navigation (Администрирование ▸ Построение сайта ▸ Меню ▸ Навигация) — в результате вы увидите страницу, представленную на рис. 4.36.

   Пункты меню представляются в иерархической таблице, которая имеет следующие поля:

   - Menu item (Пункт меню) — название пункта меню;
   - Enabled (Включен) — определяет, доступен ли данный пункт меню;
   - Expanded (Разворачивать) — устанавливает, может ли этот пункт меню иметь дочерние элементы;
   - Operations (Действия) — операции, которые можно проводить с данным элементом:
     - Edit (Редактировать) — редактировать;
     - Reset (Сброс) — сбросить настройки в значения по умолчанию;
     - Delete (Удалить) — удалить данный пункт меню.

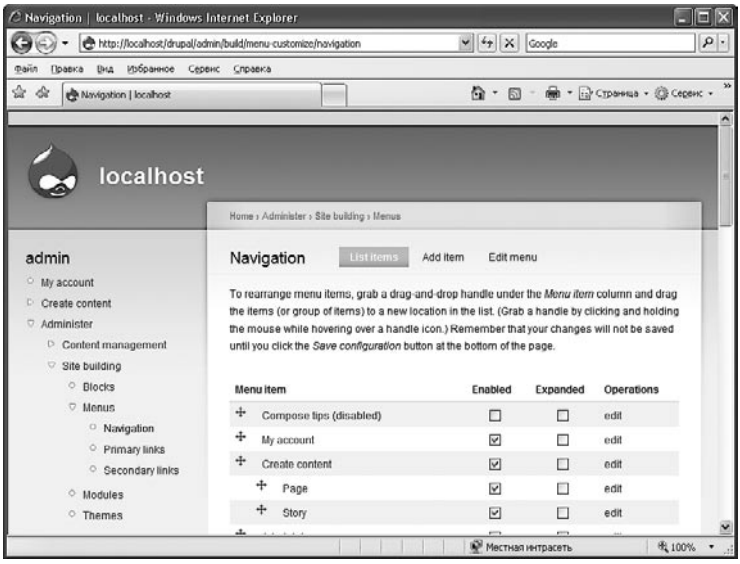


Рис. 4.36. Меню

3. Если вы хотите отредактировать параметры существующего пункта меню, щелкните по ссылке Edit (Изменить) в строке пункта меню, если же хотите добавить новый пункт — по ссылке Add Item (Вставить пункт меню) в верхней части страницы.

При добавлении и редактировании параметров пункта меню вы попадаете на страницы с одинаковыми элементами, поэтому рассмотрим только один из режимов — добавление нового пункта меню (рис. 4.37).

Область Menu settings этого окна имеет следующие параметры.

- Path (Путь) — путь, куда следует переходить при выборе данного пункта меню. Возможны такие значения:
  - внутренний путь Drupal, например `node/add`;
  - ссылка на внешний URL;
  - `<front>` — ссылка на главную страницу.

  Если пункт меню имеет дочерние элементы, то вы все равно должны указать путь перехода. Поэтому, если вы не знаете, куда делать переход, чтобы открылись дочерние элементы, поставьте в это поле ссылку `<front>`.
- Menu link title (Заголовок пункта меню) — название пункта меню.
- Description (Описание) — описание пункта меню.

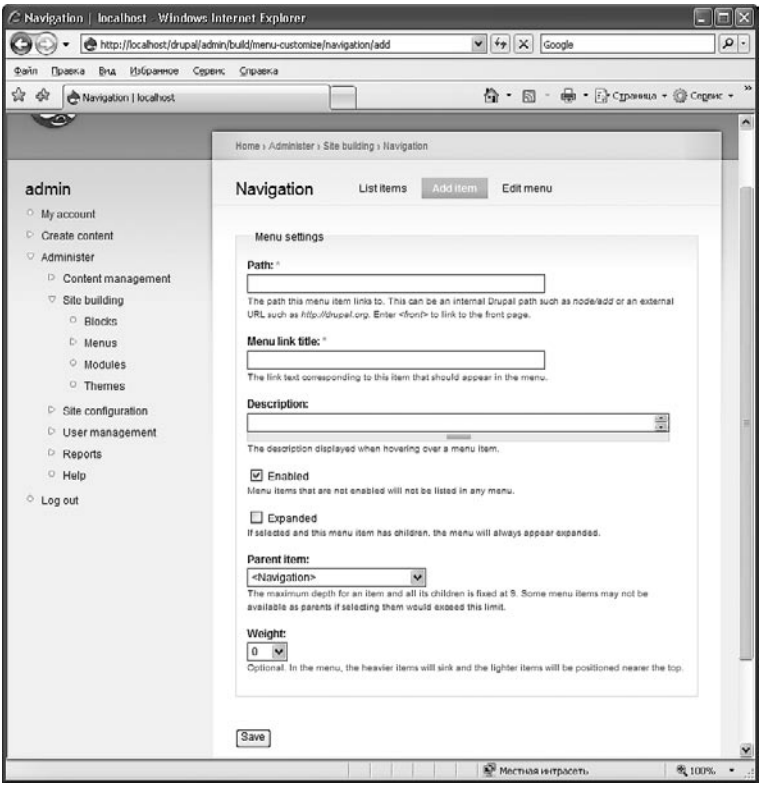


Рис. 4.37. Добавление пункта меню

- Enabled (Включить) — если включен этот параметр, то пункт меню включается в меню.
- Expanded (Разворачивать) — указывает, может ли данный пункт иметь дочерние элементы.
- Parent item (Родительский пункт меню) — родительский элемент, которому принадлежит текущий пункт меню.
- Weight (Вес) — «вес» пункта меню. Чем меньше значение, тем ближе пункт меню находится к верхней части меню (внутри родительского элемента).

Резюме

В этой главе мы вкратце рассмотрели две системы управления содержимым сайтов — Joomla! и Drupal. Нельзя сказать, что одна система лучше, а другая хуже. Каждая из них имеет свои особенности, сильные и слабые стороны. Если вы начи-

нающий «сайтостроитель» и не хотите глубоко вникать в программирование сайтов, то обратите более пристальное внимание на Joomla! с ее развитой модульной структурой. Эта система имеет также неплохое средство отладки положения модулей на странице. Drupal же лучше использовать программистам, которые хорошо знают PHP, JavaScript и хотят иметь более гибкую систему управления сайтом, хотя и относительно сложную. Пользователям следует выбирать конкретную CMS исходя из того, какая система предоставляет больше возможностей, необходимых конкретному сайту, а какая система ближе по идеологии и конфигурации веб-серверу.

Глава 5

Создание серверных сценариев на языке PHP 6

Создание и запуск первого сценария PHP

Комментарии внутри кода PHP

Переменные

Типы данных PHP

Константы в PHP

Операторы PHP

Работа с функциями в PHP

Резюме

Аббревиатура PHP расшифровывается как Hypertext Preprocessor, что переводится как «препроцессор гипертекста». Синтаксис языка PHP достаточно прост для изучения и берет свое начало в языках Perl, Java и C. Однако, в отличие от перечисленных языков, PHP изначально разрабатывался как язык программирования для написания сценариев (веб-приложений), исполняющихся на веб-сервере. В этом состоит его первое достоинство — средства PHP позволяют быстро и эффективно создавать полнофункциональные веб-приложения.

Другим достоинством PHP является предоставляемая им возможность внедрять свои сценарии в HTML-код веб-страниц, что значительно упрощает задачу создания динамических сайтов. С помощью PHP разработчики могут динамически изменять HTML-код страниц в зависимости от действий посетителя сайта. Например, при вводе посетителем данных в поля формы, или установке переключателей, флажков, или выборе определенного элемента списка сервер учтет все эти данные и динамически перестроит всю информацию, передаваемую интернет-браузеру.

Вместе с тем язык PHP весьма прост для освоения. Даже начинающие программисты смогут быстро научиться создавать с помощью PHP достаточно сложные сценарии, реализующие функции и интерфейс на уровне профессиональных веб-сайтов.

Итак, приступим к изучению языка PHP. Надеюсь, что вы прочли предыдущую главу этой книги и у вас на компьютере установлены серверы Apache и MySQL. Тогда вы сможете повторить изложенные в данной главе конструкции языка PHP. Если вы это не сделали, настоятельно рекомендую вернуться назад и выполнить установку этих средств.

Вначале займемся изучением синтаксиса языка PHP. Вы познакомитесь с типами данных, используемых в PHP, базовыми конструкциями по управлению ходом работы программы, функциями, способами передачи данных в сценарии, научитесь работать с базами данных MySQL, открывать сеансы работы со сценарием и записывать cookie-файлы. По мере изложения материала я буду демонстрировать применение указанных средств на примерах сценариев, исполняемых на локальном сервере пользовательского компьютера. Поэтому так важно, чтобы вы установили эти средства.

Создание и запуск первого сценария PHP

Для начала создадим простой сценарий, чтобы продемонстрировать общую структуру PHP-кода и возможность его использования вместе с HTML-кодом. Откроем

какой-нибудь текстовый редактор, например Блокнот системы Windows, и запишем в него следующий код PHP:

```
<?php
   phpinfo();
?>
```

Далее откроем корневую папку htdocs сервера Apache, в которой сохраняются исполняемые сценарии, и создадим подпапку с названием examples, в которой мы будем сохранять файлы PHP-сценариев. Назовем наш файл `start.php` и сохраним его в созданной папке.



ПРИМЕЧАНИЕ

Если вы используете пакет XAMPP, то папка htdocs находится внутри папки, в которую вы установили пакет. Обычно эта папка называется имя_диска:\XAMPP. При работе с сервером Apache напрямую данный каталог необходимо указать в директиве DocumentRoot в файле httpf.conf. Однако если вы предпочитаете работать с сервером IIS, то помещать свои сценарии нужно в папку inetpub//wwwroot.

Теперь все готово для исполнения первого сценария. Откройте браузер и в его адресной строке наберите `http://localhost/examples/start.php`. В окне браузера отобразятся сведения о версии и конфигурации установленного языка PHP.

Запись `phpinfo()` внутри сценария означает вызов встроенной функции PHP — одной из многих, которые этот язык предлагает разработчикам сценариев. Запомните, что в конце любого оператора языка PHP должна находиться точка с запятой.

Обратите внимание на остальные записи внутри файла. Строки `<?php` и `?>` соответствуют начальному и конечному тегу кода PHP, внутри которых помещаются сценарии, внедряемые в HTML-код веб-страницы. Давайте посмотрим, как это делается на практике. Откроем новый документ текстового процессора и запишем в него следующий код (листинг 5.1).

Листинг 5.1. Документ HTML с кодом PHP

```
<html>
  <head>
    <title>
        Эта страничка содержит код PHP
    </title>
    <body>
```

```
        <h1>
          Внедрение сценария PHP в код HTML
        </h1>
    <?php
        echo "Привет Мир! Это PHP-сценарий внутри странички Web!";
    ?>
    </body>
</html>
```

Назовем файл с этим кодом `hello.php` и сохраним его в папке examples. Введем в адресную строку браузера вызов сохраненного кода `http://localhost/examples/hello.php`, после чего в окне браузера отобразится результат его исполнения (рис. 5.1).

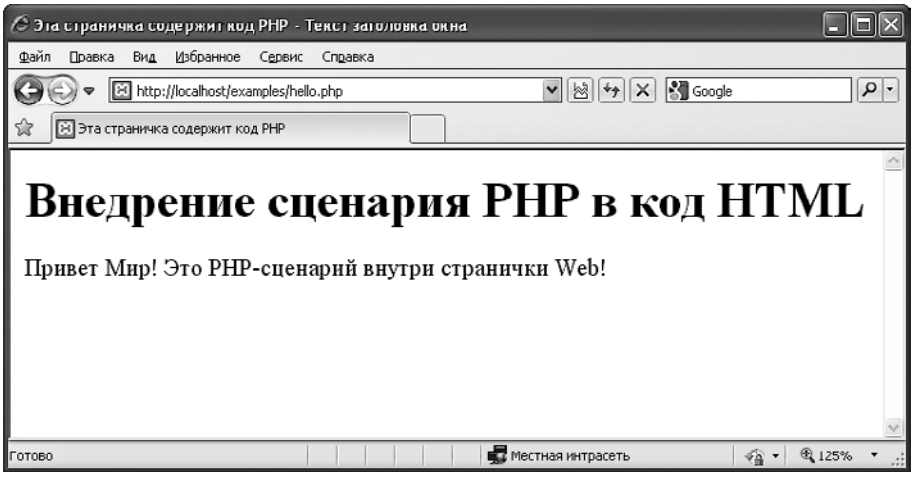


Рис. 5.1. Приветствие сценария PHP внутри кода HTML

Внимательно рассмотрим содержимое листинга 5.1. Вы видите, что внутри тела документа HTML в содержимом тега `<h1>` присутствует заголовок, после которого внутри тегов `<?php` и `?>` помещен оператор `echo` с текстом приветствия. Оператор `echo` является одним из основных в языке PHP, он позволяет отображать любой текст внутри веб-страницы. В листинге 5.1 представлено простейшее использование этого оператора:

```
echo "Здесь помещается выводимый текст";
```

Выводимый текст следует заключить в одинарные (апострофы) или двойные кавычки. Числа могут отображаться без кавычек. Например, корректно писать:

```
echo "872976";
```

Вы можете включать в выводимую строку теги HTML, которые будут восприниматься браузером как обычный HTML-код. Например, код

```
echo "<b>Этот текст</b> выделен полужирным начертанием";
```

приведет к выделению строки «**Этот текст**» полужирным начертанием.

В операторе `echo` можно выводить несколько строк по отдельности. Например, запись

```
echo "Привет", "это", "PHP";
```

приведет к выводу текста «ПриветэтоPHP». Чтобы отделить слова друг от друга, помещайте внутри кавычек пробелы:

```
echo "  Привет  ", "  это  ", "  PHP  ";
```

Если вы хотите отобразить текст в кавычках, пишите так:

```
echo " Петя сказал: \" Я пошел в школу ";
```

Иными словами, перед кавычками следует поставить обратный слеш.

Комментарии внутри кода PHP

Всякий программист при создании сценария оставляет в нем свои замечания или комментарии, которые должны информировать пользователя программы о функции и назначении определенного фрагмента кода. Не следует пренебрегать этой возможностью, без комментариев вам придется все время вспоминать, что и как вы реализовали в своем сценарии, тратя на это силы и время (а программные коды забываются очень быстро). Поэтому используйте комментарии, они вам очень пригодятся, если вы захотите вернуться к ранее написанным программам.

Чтобы поместить комментарий в сценарий PHP, вы можете использовать несколько возможностей. Во-первых, заключить комментарий в знаки `/* Текст комментария */`, например:

```
<?php
/* Этот сценарий просто отображает сведения
 о версии и конфигурации языка PHP
 в окне браузера */
```

```
   phpinfo();
?>
```

Приведенный комментарий допустимо располагать в нескольких строках, но учтите, что вложенные сценарии недопустимы. Следующая запись приведет к ошибке:

```
<?php
/* Этот сценарий просто отображает сведения
/* о версии и конфигурации языка PHP */
   в окне браузера */
   phpinfo();
?>
```

Здесь строка

```
в окне браузера */»
```

не будет распознана как комментарий, поскольку начальная метка комментария `/*` завершится в конце второй строки меткой `*/`.

Существует еще два способа размещения комментариев, но только размещенных в одной строке. Их следует помещать после символов `//` или #, например:

```
<?php
// Этот сценарий просто отображает сведения
// о версии и конфигурации языка PHP
// в окне браузера
   phpinfo(); # Это встроенная функция PHP вывода информации о PHP
?>
```

Опять-таки учтите, что много комментариев не бывает. Поэтому настоятельно рекомендую вам не пренебрегать ими: в идеале каждая функция и значимый оператор сценария должны снабжаться комментарием.

Переменные

Все данные, с которыми работает сценарий, хранятся в переменных. *Переменные* — это контейнеры данных, обозначенные определенным идентификатором. В языке PHP **идентификатор каждой переменной начинается со знака доллара ($).** Переменная создается после того, как ей присваивается какое-либо значение:

```
$message= "Здравствуй мир!";
$counter=10;
$amount=3.62;
```

Переменные могут хранить данные разных типов — целочисленные, вещественные, логические, текстовые и др., что мы и видим в вышеприведенном примере. В отличие от других языков программирования в PHP **не требуется явно определять тип** данных, поскольку он определяется автоматически. Однако если вы используете в одном выражении переменные разных типов, могут возникнуть трудности. Например, в коде:

```
$temperature="0";
$temperature=$temperature+2;
```

к текстовой переменной `$temperature`, хранящей символ `"0"` (ноль), добавляется число `2`. В этом случае выполнятся неявное преобразование типов. Чтобы явно определить преобразование типов, следует писать:

```
$int_variable=(integer)$variable;,
```

то есть указывать в скобках нужный тип данных. Ниже приведен список допустимых типов данных PHP:

- `boolean` — логический тип со значениями `true` или `false`;
- `string` — текстовые данные;
- `integer` — целые числа;
- `float` — вещественные числа;
- `object` — объект;
- `array` — массив;
- `resourse` — ресурс;
- `NULL` — пустое значение.

Рассмотрим указанные типы данных подробнее.

Типы данных PHP

В настоящем разделе мы перечислим все типы данных, кроме типа `object`, и приведем примеры их использования на простых примерах. Данные типа `object` используются для создания сценариев методами объектно-ориентированного программирования, а рассмотрение этой темы выходит за рамки нашей книги.

Boolean

Логический тип данных позволяет переменной иметь два значения: `true` (истина) и `false` (ложь). Для определения переменной этого типа данных ей следует присвоить одно из указанных значений:

```
<?php
$variable = false; // присвоить $variable значение false
?>
```

Обычно логические типы данных используются для создания конструкций, в которых операторы или функции проверяют исполнение какого-либо условия и возвращают результат в виде булевой переменной. Например, в конструкции

```
<?php
       if (MyFunction() == true) {
          echo "Функция MyFunction выполнена успешно";
       }
?>
```

оператор `if` проверяет значение, возвращаемое функцией `MyFuction()`, и, если оно равно `true`, сообщает об успешности выполнения функции.

String

Строковые переменные хранят текстовые данные любой длины. Длина строки ограничена только размером оперативной памяти.

Строки обрабатываются либо с помощью набора функций, предоставляемых PHP, либо к любому символу строки можно обратиться непосредственно операторами языка PHP. Ниже представлен пример строковой переменной:

```
<?php
$string = "Пример текста, хранимого строковой переменной";
echo $string;
?>
```

Функция `echo` отображает содержимое строки `string` в окне браузера.

Integer

Указанный тип данных предназначен для хранения целочисленных данных с разрядностью 32 бита. Значения этих чисел лежат в пределах от –2 147 483 648

до 2 147 483 647. Кроме того, они могут иметь знак «+» или «-» и записываться с помощью десятичной, восьмеричной и шестнадцатеричной системы счисления.

- При использовании восьмеричной системы счисления перед значением пишется символ `0` (ноль), например: `$variable=0739`.
- Если вы используете шестнадцатеричную систему, вы должны предварять число символами `0x`, например: `$variable=0x837`.

Float

Тип данных `float` предназначен для хранения вещественных чисел. Максимальное значение вещественного числа зависит от платформы, на которой исполняется сценарий, и обычно ограничено числом с 14 **десятичными знаками. Вещественные** числа записываются в одном из представленных ниже форматов:

```
<?php
$variable1 = 1.264;
$variable2 = 1.3e3;
$variable3 = 3E-10;
?>
```

Array

В любом языке программирования массивами называются упорядоченные наборы данных, в которых каждому хранимому значению присваивается соответствующий индекс, или ключ. Индекс используется для идентификации элемента, и между значениями индекса и элементом массива установлено взаимно-однозначное соответствие.

Массивы можно организовать по-разному, в зависимости от того, как организовано соответствие между ключом и индексом. Например, в простейшем массиве каждый индекс является целым числом от 0 и выше:

```
<?php
$array[0]="Диван";
$array[1]="Кресло";
$array[2]="Стол";
$array[3]="Стул";
?>
```

Здесь `$array` обозначает имя массива, а 0, 1, 2 и 3 — индексы массива.

Массивы, в которых индексами служат строки, называются ассоциативными, например:

```
<?php
$array["Имя"]="Иван";
$array["Отчество"]="Николаевич";
$array["Фамилия"]="Семенов";
?>
```

Здесь ключами массива служат названия пунктов анкетных данных, а значениями — соответствующие сведения.

Массивы могут быть многомерными. Синтаксис элементов многомерного простого массива таков:

```
$имя_массива[индекс_1][индекс_2]..[индекс_N];
```

Например, вы можете определить многомерный массив следующим образом:

```
<?php
$array[0][0]="Диван";
$array[0][1]="Кровать";
$array[1][0]="Кресло";
$array[1][1]="Тумбочка";
$array[1][2]="Стол";
$array[2][0]="Стул";
$array[2][1]="Шкаф";
$array[2][2]="Комод";
?>
```

Чтобы создать массив, можно воспользоваться функцией PHP под названием `array()`. Ниже приведен пример ее использования:

```
<?php
$array["Семенов"] = array("Возраст"=>"34", "Имя"=>"Игорь", "Отчество"=>
"Онуфриевич");
$array["Козлов"] = array("Возраст"=>"44", "Имя"=>"Егор", "Отчество"=>
"Тимофеевич");
$array["Федотов"] = array("Возраст"=>"32", "Имя"=>"Степан", "Отчество"=>
"Лукич");
?>
```

Здесь каждой фамилии сопоставлены данные анкеты по трем пунктам: `Возраст`, `Имя` и `Отчество`. Метка => как раз и задает соответствие между индексом массива и значением его элемента.

Resource

Тип данных `resource` содержит ссылку на внешний ресурс, который будет далее использован в сценарии. Например, это может быть ссылка на файл или запись в базе данных. Для создания данных такого типа используются специальные функции, которые мы обсудим.

NULL

Указанный тип данных еще называется пустым типом. Это значение говорит о том, что переменной еще не установлено значение. Она будет иметь значение `NULL` в следующих случаях:

- если ей было присвоено значение `NULL`;
- при исполнении сценария ей еще не было присвоено какое-либо значение;
- значение переменой было удалено с помощью специальной функции `unset()`.

Константы в PHP

Часто для работы сценария необходимо использовать какой-либо параметр, с помощью которого вы собираетесь производить расчеты или отображать параметр в каких-то информационных сообщениях и т. д. Классический пример — число π, с помощью которого вычисляется длина окружности. Можно, конечно, везде использовать его значение в виде вещественного числа, но это крайне неудобно, особенно если требуется обеспечить точность и нужно все время писать много знаков после десятичной точки: 3.1415926535. Но лучше просто определить константу `pi` с помощью оператора

```
define ("pi", 3.1415926535);
```

Теперь во всех выражениях можно использовать константу `pi`, значение которой будет автоматически подставляться в код программы. Константы могут хранить данные только логического, вещественного, целого и строкового типа.

Общий формат функции `define()` имеет следующий вид:

```
define (имя_константы, значение константы, $case_sen);
```

Здесь `$case_sen` — переменная логического типа, определяющая чувствительность имени константы к регистру: значение `true` делает ее чувствительной, `false` — нет.

В листинге 5.2 представлен сценарий подсчета длины окружности и площади круга с помощью определенной константы `pi`.

Листинг 5.2. Подсчет длины окружности и площади круга

```
<?php
define("pi", 3.1415926535, true);
$r=10;
$lenght=2*pi*$r;
echo " Длина окружности радиуса ", $r, "  равна   ", $lenght;
$square=pi*pow($r, 2);
echo "<br>Площадь круга радиуса   ", $r, "  равна  ", $square;
?>
```

Результат исполнения сценария представлен на рис. 5.2.

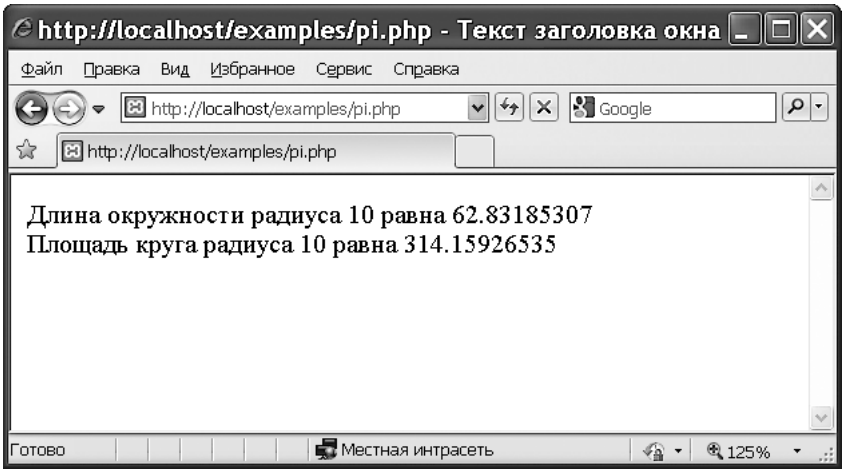


Рис. 5.2. Вычисление длины окружности и площади круга

При работе с константами следует учесть, что константы определяются только функцией `define()`, они могут быть определены и будут доступны в любом месте сценария.

Язык PHP предоставляет множество предопределенных констант, которые можно использовать в своих сценариях. Набор этих констант определяется конфигурацией

той среды, в которой вы работаете. Для проверки наличия константы существует функция `defined()`, возвращающая `true`, если константа объявлена.

Операторы PHP

Язык PHP содержит множество операторов, то есть средств, с помощью которых производится обработка данных в программе. Ниже мы обсудим основные операторы языка, выполняющие разные функции при построении программного кода: присвоение значений различным переменным, математические операции с числами, обработку текстовых и битовых данных, а также операторы, управляющие ходом выполнения программы.

Операторы присвоения

Основной оператор присвоения данных любой переменной представляет собой знак равенства «=» и используется так:

```
Левый_операнд=Правый_операнд;
```

Здесь оператор присвоения получает значение правого операнда и устанавливает его в левый операнд. Таким образом, в результате исполнения кода

```
<php?
  $variable1=5;
  $variable2=variable1;
?>
```

переменным `$variable1` и `$variable2` присваивается значение 5.

Операция присвоения позволяет присвоить значение сразу нескольким переменным:

```
$var1=$var2=$var3=$var4=2;
```

Это очень удобно и значительно упрощает код программы.

Математические операторы

С помощью математических операторов можно выполнять различные арифметические вычисления с переменными и числами. К их числу относятся следующие операторы:

- `+` — суммы двух чисел;
- `-` — разности двух чисел;
- `*` — произведение двух чисел;
- `/` — частное от деления двух чисел;
- `%` — остаток от деления двух чисел.

Примеры применения этих операторов представлены в листинге 5.3.

Листинг 5.3. Применение математических операторов

```
<?php
echo "7 + 3 = " , 7 + 3, "<br>";
echo "7 - 3 = ", 7 - 3, "<br>";
echo "7 * 3 = ", 7 * 3, "<br>";
echo "7 / 3 = ", 7 / 3, "<br>";
echo "7 / 3 = ", 7 % 3, "<br>";
?>
```

Результат исполнения кода представлен на рис. 5.3.

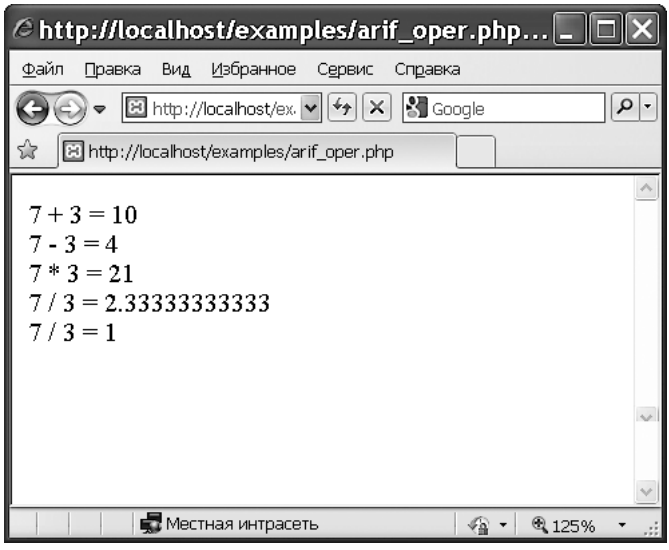


Рис. 5.3. Пример исполнения арифметических операторов PHP

Обратите внимание на операцию деления. Ее результатом всегда будет значение вещественного типа, даже если оба операнда были целочисленными. При использовании арифметических операций возможно использование скобок, как в обычных

арифметических выражениях. Приоритет выполнения арифметических операций над другими соответствует обычным математическим правилам.

Комбинированные операции присваивания

При работе с арифметическими операторами в PHP можно использовать комбинированные операторы присваивания, в которых значение левого операнда в операции присваивания применяется в арифметической операции, а затем результат сохраняется в той же переменной. Например, операция

```
$variable += 21;
```

означает добавление к переменной `$variable` числа `21` и сохранение результата в самой переменной `$variable`, то есть вышеприведенная операция эквивалентна следующей:

```
$variable = $variable + 21;
```

Ниже приведены примеры оператора присвоения, скомбинированные с различными арифметическими операторами:

```
$variable-=24;  // Вычитание из $variable числа 24
$variable*=5;   // Умножение $variable на 5
$variable/=3;   // Деление $variable на 3
```

Строковые операторы

Для работы со строковыми типами данных язык PHP предоставляет оператор конкатенации («.»), объединяющий в одну строку свои операнды, и оператор присвоения с конкатенацией, присоединяющий правый операнд к левому. Рассмотрим пример их использования:

```
<?php
$variable1 = "Здравствуй ";
$variable2 = $variable1 . "мир!";
?>
```

В приведенном примере переменной `$variable2` присваивается строка `"Здравствуй мир!"`. Это же можно сделать с использованием комбинированной операции присвоения и конкатенации:

```
<?php
$variable1 = "Здравствуй ";
```

```
$variable1 .= "мир!";
?>
```

Здесь к переменной `$variable1` присоединяется строка `"мир!"`. В обоих случаях мы получим одинаковый результат.

Операторы инкремента и декремента

Указанные операции увеличивают или уменьшают значение переменной на 1. Так, выражение `++$variable` выполняет увеличение значения `$variable` на 1 с последующим присвоением результата переменной `$variable`, а выражение `--$variable` уменьшает значение `$variable` на 1. Это очень удобно при организации счетчиков, применяемых, например, для создания циклических операций (см. ниже).

Операции инкремента и декремента бывают *префиксными* и *постфиксными*. Разница между ними проявляется в случае применения инкрементируемой или декрементируемой переменной. При префиксной операции вначале выполняется инкрементация или декрементация переменной, а затем ее использование в соответствующем выражении. В постфиксной операции в выражении используется первоначальное значение переменной, а ее инкрементация или декрементация выполняется после.

В листинге 5.4 показаны способы применения операций инкрементации и декрементации в обоих случаях.

Листинг 5.4. Постфиксное и префиксное выполнение инкремента и декремента

```
<?php
$var1=$var2=$var3=var4=2;
echo "\$var1++ = ", $var1++, "<br>";
echo "++\$var2 = ", ++$var2, "<br>";
echo "\$var3--  = ", $var3--, "<br>";
echo "--\$var4 = " , --$var4, "<br>";
?>
```

Результат выполнения сценария листинга 5.4 представлен на рис. 5.4.

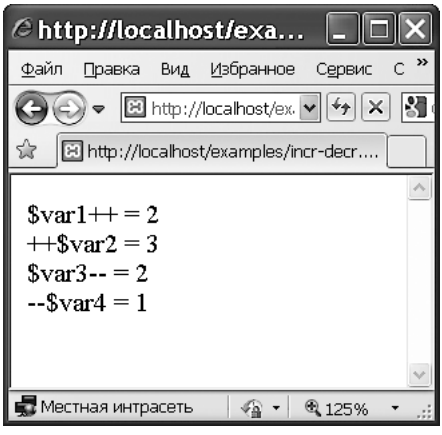


Рис. 5.4. Префиксная и постфиксная инкрементация и декрементация

Учтите, что булевские типы данных не могут быть инкрементированными или декрементированными.

Битовые операторы

Битовые операторы позволяют устанавливать значения в двоичном представлении целочисленных переменных — 1 и 0. Целые числа в PHP представлены 32-разрядными переменными. Ниже приведен пример такого числа:

```
0000 0000 0000 0000 0000 0000 0000 0101
```

В десятичном представлении это число 5. Иногда возникает необходимость установить 1 или сбросить 0 бит в двоичном разряде или группе разрядов. Для этого и служат двоичные переменные, которые позволяют сравнивать биты в двоичных разрядах двух переменных, скажем, `$var1` и `$var2`, и по результатам сравнения устанавливать соответствующий бит в возвращаемом значении. Перечислим битовые операторы:

- `&` (побитовое «и») — устанавливает в результате `var1 & var2` только те биты, которые установлены и в `$var1`, и в `$var2`;
- `|` (побитовое «или») — устанавливает в результате `var1 | var2` биты, которые установлены или в `$var1`, или в `$var2`;
- `^` (исключающее «или») — устанавливает в результате `var1 ^ var2` биты, которые установлены или только в `$var1`, или только в `$var2`;
- `~` (отрицание) — устанавливает в результате `~$var1` биты, которые в `$var1` не установлены, и наоборот;
- `<<` — выражение `$var1 << $var2` сдвигает влево все биты переменной `$var1` на `$var2` позиций влево;
- `>>` — выражение `$var1 >> $var2` сдвигает вправо все биты переменной `$var1` на `$var2` позиций вправо.

На практике приведенные операции применяются редко.

Операторы сравнения

Операторы сравнения представляют собой множество логических операторов, которые используются в других операторах для управления их работой. Самым популярным оператором такого рода является условный оператор `if`, используемый для разветвления процесса исполнения программы в зависимости от различных условий. Его общий вид приведен ниже:

```
if (логическое_выражение)
   оператор
```

Здесь логическое выражение может принимать значение `true` или `false`, в зависимости текущих значений входящих в него переменных. Если логическое выражение внутри `if` истинно, исполняется оператор, стоящий сразу после `if`.

В логическое выражение могут входить многие операторы, включая функции, возвращающие логические значения. Но чаще всего в нем используются операторы сравнения, например, выражение `$x<$y` истинно, если значение `$x` меньше `$y`.

В PHP имеются следующие операторы сравнения:

- == (равенство) — выражение `$x==$y` истинно, если значение `$x` равно `$y`;
- === (идентичность) — выражение `$x===$y` истинно, если значение `$x` равно `$y` и они одного типа;
- != (неравенство) — выражение `$x!=$y` истинно, если `$x` не равно `$y`;
- <> (неравенство) — выражение `$x<>$y` истинно, если `$x` не равно `$y`;
- !== (не идентичность) — выражение `$x!==$y` истинно, если или `$x` не равно `$y` или они не одного типа;
- < (меньше) — выражение `$x<$y` истинно, если `$x` меньше `$y`;
- > (больше) — выражение `$x<$y` истинно, если `$x` больше `$y`;
- <= (меньше или равно) — выражение `$x<=$y` истинно, если `$x` меньше или равно `$y`;
- >= (больше или равно) — выражение `$x>=$y` истинно, если `$x` больше или равно `$y`.

Рассмотрим пример использования операторов сравнения:

```
<?php
  $temperature=26;
  if($temperature>=24)
     echo "На улице тепло"
?>
```

Операторы сравнения разрешается применять только к скалярным переменным, но не к массивам и объектам.

Логические операторы

Управлять ходом вычислений в программе, делая его максимально гибким и интуитивно понятным, позволяют также логические операторы. Например, с их помощью можно объединять два условных оператора:

```
if(логическое_выражение_1&&логическое_выражение_2)
   оператор
```

Здесь оператор после `if` выполняется, если истинно и `логическое выражение 1`, и `логическое выражение 2`. Таким образом, комбинируя логические выражения и операторы, можно строить весьма сложные конструкции, реализующие любую, самую изощренную, логику работы программы.

В PHP имеются следующие логические операторы:

- AND (логическое «и») — выражение `$x AND $y` истинно, если и `$x`, и `$y` истинны;
- `OR` (логическое «или») — выражение `$x OR $y` истинно, если истинно или `$x`, или `$y`;
- `XOR` (исключающее «или») — выражение `$x XOR $y` истинно, если истинно или `$x`, или `$y`, но не оба;
- `!` (отрицание) — выражение `!$x` истинно, если `$x` ложно;
- `&&` (логическое «и») — выражение `$x && $y` истинно, если и `$x`, и `$y` истинны;
- `||` (логическое «или») — выражение `$x || $y` истинно, если истинно или `$x`, или `$y`.

Возникает вопрос, зачем нужны совпадающие по функциям операторы AND и `&&` или `OR` и `||`. Дело в том, что операторы `&&` и `||` имеют более высокий приоритет операций. Обсудим этот вопрос подробнее, так как он очень важен при создании в программе сложных управляющих конструкций.

Приоритеты операторов

Приоритет операторов языка PHP представлен в табл. 5.1. При знакомстве с ней помните, что операторы с более высоким приоритетом выполняются в первую очередь.

Таблица 5.1. Приоритет операторов

Уровень приоритета	Оператор
1	= += -= *= /= %= >>= <<== &= ^= |=
2	||
3	&&

Уровень приоритета	Оператор
4	|
5	^
6	&
7	!=
8	<<= >>=
9	<< >>
10	+-
11	* / %
12	++ --

В любом случае использование скобок позволяет решить все проблемы — в PHP правила приоритета операторов вполне аналогичны правилам, применяемым в школьной арифметике.

Условные операторы

Условные операторы в языке PHP применяются для управления ходом исполнения программы. Выше при описании операторов сравнения мы рассмотрели наиболее популярный оператор `if`, но кроме него PHP предлагает и другие условные операторы.

Оператор if

Синтаксис оператора `if` в общем случае выглядит следующим образом:

```
if (логическое_выражение)
   {
       оператор1;
          …
        опратор2;
   }
```

Если логическое выражение в операторе `if` истинно, то выполняются операторы, следующие за конструкцией `if`. Если таких операторов должно быть несколько, то их помещают в фигурные скобки. Приведем пример использования оператора:

```
<?php
    if ($x < $y) {
```

```
      echo "x меньше y";
      $z=$y;
      $y = $x;
      $x=$z;
   }
?>
```

Здесь выполняется сравнение двух переменных `$x` и `$y` и, если `$x` меньше `$y`, переменной `$x` присваивается значение `$y`, а `$y` — значение `$x`, то есть они обмениваются значениями.

Конструкция if...else

Если в программе необходимо совершить некоторые действия, независимо от того, выполняются или не выполняются определенные условия, используется конструкция `if...else`. Общий вид этой конструкции приведен ниже:

```
if (логическое_выражение)
    оператор_1;
else
    оператор_2;
```

Здесь в случае истинности логического выражения в операторе `if` выполняется `оператор_1`, а иначе — `оператор_2`. Если исполняемых операторов должно быть несколько, то их заключают в фигурные скобки:

```
<?php
    if ($x > $y) {
      echo "x меньше y";
      $z=$y;
      $y=$x;
      $x=$z;
     } else
      echo "y больше или равно x";
?>
```

Кроме того, конструкцию `if...else` можно написать как:

```
if (логическое_выражение_1):
   набор_операторов_1;
elseif(логическое_выражение_2):
   набор_операторов_2;
```

```
else:
   набор_операторов_3;
endif
```

В приведенной конструкции после каждого блока `if...else` вместо фигурных скобок ставится двоеточие, после которого располагаются наборы операторов. Обратите внимание на новую конструкцию `elseif`, расширяющую возможности оператора `if`. Общий синтаксис конструкции `elseif` выглядит следующим образом:

```
if (логическое_выражение_1) {
     набор_операторов_1;
} elseif (логическое_выражение_2) {
     набор_операторов_2;
} elseif (логическое_выражение_3) {
     набор_операторов_3;
}
```

Описанные выше конструкции позволяют эффективно разветвлять процесс исполнения кода программы в зависимости от условий, но ими возможности PHP не ограничиваются.

Оператор switch

Если программе следует выбрать варианты действий в зависимости от значений какого-либо выражения, не обязательно логического, вместо оператора `if` удобнее использовать конструкцию `switch-case`. Синтаксис конструкции `switch-case` приведен ниже:

```
switch(выражение) {
case значение_выражения_1:
    набор_операторов_1;
    [break;]
case значение_выражения_2:
    набор_операторов_2;
    [break;]
. . .
case значение_выражения_N:
    набор_операторов_N;
    [break;]
```

```
[default: операторы_выполняемые_по_умолчанию;
[break]]
}
```

В этой конструкции вычисляется выражение в операторе `switch`, и полученный результат сравнивается со `значениями_выражения` каждого блока `case`. Если значения совпадают, исполняется набор операторов данного блока `case`. Если ни одно `значение_выражения` в блоках `case` не совпадет с результатом подсчета выражения, выполняются операторы блока `default`.

Вместе с тем нужно учесть одну тонкость: после исполнения операторов блока `case` выполнение сценария переходит в следующий блок, потом в следующий и так до конца оператора `switch`, либо пока не встретится оператор `break`.

Приведем пример конструкции `switch-case`:

```
<?php
switch ($x) {
case 0:
case 1:
case 2:
     echo "x меньше 3<br>";
     break;
case 3:
    echo "x=3<br>";
    break;
case 4:
    echo "x=4<br>";
    break;
case 5:
    echo "x=5<br>";
    break;
case 6:
    echo "x Больше 5<br>";
}
?>
```

Как видим, здесь набор операторов `case` для значений `0` и `1` пуст, но если значение `$x` равно `0` или `1`, управление перейдет соответственно на блок `switch 0:` или `switch 1:`. После чего выполнится оператор третьего блока `switch 3:` и отобразится сообщение "`x меньше 3`".

Операторы циклов

Циклы являются следующей по значимости категорией операторов PHP после условных операторов. Они предназначены для повторения исполнения определенного набора операторов по заданному условию. Повторяемые операторы образуют тело оператора цикла, а повторяемые проходы по этим операторам — циклы.

В PHP имеются три вида операторов циклов:

- `FOR` — цикл со счетчиком;
- `WHILE` — цикл с предусловием;
- `DO...WHILE` — цикл с постусловием;
- `FOREACH` — цикл перебора массивов.

Рассмотрим эти операторы по порядку.

Оператор цикла FOR

Оператор `for` позволяет выполнить операторы тела цикла заданное число раз. Синтаксис оператора цикла `for` имеет следующий вид:

```
for (операторы_инициации_цикла; условие_цикла; операторы_после_выполнения_
цикла) {
   тело_цикла;
}
```

Приведем пример такого оператора:

```
<?php
   for(x=2, $i=1; $i<12; i++) {
       x +=2;
       echo "<br>  i = ", $i, "  x = ", $x;
    }
?>
```

В этом примере в начале цикла инициализируется значение счетчика `$i` и переменной `$x`, а затем начинается выполнение цикла (обратите внимание, что операторы инициации цикла разделяются запятыми). После инициализации выполняются следующие действия.

1. Проверяется условие цикла, то есть недостижение переменной цикла значения 12.
2. Если условие цикла истинно, выполняется тело цикла, иначе итерации завершаются.
3. Далее выполняются операторы после выполнения цикла.
4. Переходим на шаг 1.

Если при задании цикла требуется ввести несколько операторов инициации цикла, проверки условий цикла или операторов после цикла, их можно разделить запятыми, как в приведенном выше примере.

Оператор цикла do...while

Оператор цикла `while` также проверяет значение выражения после каждого прохода, поэтому, в отличие от `for`, тело цикла `while` выполняется хотя бы один раз. Синтаксис цикла `while` приведен ниже:

```
do {
    тело_цикла;
}
while (логическое_выражение);
```

После очередной итерации проверяется, истинно ли `логическое_выражение`, и, если это так, управление передается вновь в начало цикла, в противном случае цикл обрывается. Приведем пример применения цикла с постусловием `do...while`:

```
<?php
$i = 1;
do {
     echo "<br> i =   ", $i;
} while ($i++<10);
?>
```

Здесь перед выполнением цикла счетчику цикла `$i` присваивается начальное значение, после чего цикл выполняется 10 раз.

Оператор цикла while

Синтаксис цикла с предусловием имеет следующий вид:

```
while (логическое_выражение) {
    набор_операторов
}
```

Обычно тело цикла состоит из большого числа операторов.

Оператор цикла `while` работает следующим образом.

1. Вычисляется значение логического выражения.
2. Если значение истинно, выполняется тело цикла, в противном случае переходим на следующий за циклом оператор.

Приведем пример цикла с предусловием `while`:

```
<?php
   $x=0;
   while ($x++<12) {
       echo "<br> i = ", $i;
   }
?>
```

Важно учитывать последовательность выполнения операций условия `$i++<12` с постфиксной инкрементацией. Сначала проверяется условие, а только потом увеличивается значение переменной. Если операция инкремента префиксная, то есть записывается как `++$i<12`, то сначала будет выполнено увеличение переменной, а только затем — сравнение.

Оператор цикла foreach

Оператор `foreach` специально предназначен для перебора массивов. Он имеет следующий синтаксис:

```
foreach (массив as $ключ=>$значение) {
   набор_операторов
}
```

Приведем пример работы цикла `foreach`:

```
<?php
$fruits["<бананы"] = "3 USD";
$fruits["ананасы"] = "5 USD";
```

```
$fruits["яблоки"] = "2 USD";
$fruits["апельсины"] = "3 USD";
foreach ($fruits as $key => $value) {
  echo "<b>  Фрукт: ", $key , "    Цена:   ", $key;
}
?>
```

В примере циклически выполняется просмотр массива `$fruits` и для каждого элемента выводится пара `$key` и `$value`, которая содержит название фрукта и его цену.

Оператор break

Чтобы прервать выполнение цикла в ходе любой итерации, не дожидаясь ее завершения, необходимо использовать оператор `break`. Синтаксис оператора `break` очень прост — внутрь тела цикла в нужное место помещается оператор `break`. Например:

```
<?php
 $i=1;
 while ($i++<9) {
  if ($5==3) break;
     echo "<b> Итерация", $i";
  }
?>
```

Здесь при достижении `$i` значения 3 цикл прерывается.

Если используются вложенные циклы, можно указать номер прерываемого цикла. Нумерация циклов выглядит следующим образом:

```
for (...) // Третий цикл
{
  for (...) // Второй цикл
  {
     for (...) // Первый цикл
     {
     }
  }
}
```

Чтобы прервать выполнение какого-то вложенного цикла, необходимо использовать такую конструкцию:

```
break(номер_цикла);
```

По умолчанию для номера цикла используется 1, то есть выполняется выход из текущего цикла, но иногда применяются и другие значения.

Оператор continue

Оператор `continue` немедленно завершает текущую итерацию цикла и в случае выполнения условий цикла переходит к новой итерации. Для `continue` можно указать уровень вложенности цикла, который будет продолжен после возврата управления.

В основном `continue` позволяет сократить количество фигурных скобок в коде и увеличить его удобочитаемость. Чаще всего это нужно в циклах-фильтрах, когда требуется перебрать какое-то количество объектов и выбрать из них только те, которые удовлетворяют определенным условиям. Приведем пример использования конструкции `continue`:

```
<?php
   $i=0;
   while ($i++<15) {
      if ($i==6) continue;
        echo "<b>Итерация  $i";
}
?>
```

Рассмотренный цикл прервется на шестой итерации.

Работа с функциями в PHP

В PHP, как и в любом языке программирования, существуют подпрограммы, то есть фрагменты программы, оформленные так, что к ним можно обратиться из любого места программы. Это позволяет значительно упростить программу за счет исключения повторяющихся кодов, улучшить ее читабельность, а также сократить объем памяти, необходимый для ее хранения и исполнения.

В PHP подпрограммы называются пользовательскими функциями. Ниже приведены некоторые особенности пользовательских функций, отличающие их от подобного рода средств в других языках программирования.

- Функции PHP возвращают любой тип данных.
- Дают возможность изменять переданные функции параметры.
- Параметры поддерживаются по умолчанию, можно вызвать одну и ту же функцию с переменным числом параметров.

К перечисленным можно добавить отличия в видимости, то есть доступности, переменных в различных частях программного кода. На этом вопросе мы остановимся ниже, а сейчас посмотрим, как в PHP **задаются пользовательские функции.**

Синтаксис функций PHP

Синтаксис объявления функций имеет следующий вид:

```
function имя_функции (аргумент1[=значение1],...,аргумент1[=значение1])
{
   тело_функции
}
```

В начале объявления функции стоит служебное слово `function`, потом — имя функции, и далее в скобках через запятую перечислены аргументы функции. Каждому аргументу можно присвоить значение по умолчанию.

Затем приводится тело функции, содержащее список операторов, заключенный в фигурные скобки. Функцию можно объявить в любом месте программы, но обязательно до момента ее первого применения. В общем случае имена функций могут содержать русские буквы, но лучше их не использовать — могут возникнуть проблемы совместимости. Имена функций должны быть уникальными и не должны содержать пробелов. Регистр символов в данном случае не учитывается.

Пользовательские функции могут возвращать значения любого типа. Для этого используется оператор `return`, который может вернуть что угодно, включая массивы. Если указанный оператор опустить, функция не вернет никакого значения.

Приведем пример функции `return`:

```
<?php
 function user_function() {
   $return_value = 666;
   return $return_value;
  }
```

```
$x = user_function();
echo "<br>x=   ", $x;
?>
```

Здесь функция `user_function` с помощью оператора `return` возвратит число `666`.

Передача аргументов

Если при объявлении функции был указан список аргументов, этой функции при вызове можно передавать параметры. Например, пусть функция объявлена так:

```
<?php
 function user_function ($x, $y, $z)  {
    …
 }
?>
```

Тогда при вызове функции `user_function()` обязательно нужно указать все передаваемые аргументы.

Аргументы можно передавать по ссылке. Для этого перед именем аргумента в определении функции следует поместить амперсанд («&»).

```
<?php
    function user_function(&$string_argument) {
     $string_argument .= ‘а эта внутри.’;
}
$string = ‘Эта строка вне функции, ‘;
user_function($string);
echo "<br>  string =   ", $string;
?>
```

В результате будет выведена строка «**Эта строка вне функции, а эта внутри.**»

Параметры по умолчанию

Если при задании функции указать для аргументов значения по умолчанию, можно опустить передачу параметров. В этом случае будут использоваться значения по умолчанию. Приведем пример:

```
<?php
     function sale_fruit($argument = "банан") {
     return "Дайте мне $argument";
}
echo "<br>", sale_fruit();
echo "<br>", sale_fruit("яблоко");
?>
```

При исполнении программы появятся две строки:

```
Дайте мне банан

Дайте мне яблоко
```

Первая строка отображает значение аргумента по умолчанию, а вторая — переданное значение. Учтите, значение по умолчанию должно быть константой.

Видимость переменных

С точки зрения видимости переменные подразделяются на глобальные и локальные.

- *Глобальные переменные* — доступны из любого места программы, в том числе из функции.
- *Локальные переменные* — определены внутри функции и доступны только изнутри функции.

По умолчанию все используемые внутри функции переменные локальны и вы не можете изменить их вне тела функции. Даже если ее имя совпадает с именем глобальной переменной, объявленной в другом месте программы, вне функции, никакие изменения локальной переменной не повлияют на глобальную.

Например, рассмотрим такую программу:

```
<?php
$x = 69;
function user_function() {
  $x = 96;
  echo "<br> внутри функции x =   ", $x;
}
```

```
user_function();
echo "<br> извне функции x =  ", $x;
?>
```

Программа отобразит следующее:

```
внутри функции x = 96

извне функции x = 69
```

Чтобы переменная была доступна из любого места программы, ее следует объявить глобальной с помощью следующей конструкции:

```
global $variable1, $variable 2…. ;
```

Рассмотрим пример использования глобальной переменной:

```
<?php
$x = 5;
$y = 4;
function addition() {
     global $x, $y, $z;
     $z = $x + $y;
}
addition();
echo "<br> z =   ", $z;
?>
```

Программа приведенного примера отобразит `"z = 9"`. После определения внутри функции `addition()` переменных `$x`, `$y` и `$z` глобальными все обращения к этим переменным будут указывать на глобальную версию. Не существует ограничений на количество глобальных переменных, которые могут обрабатываться пользовательскими функциями.

Кроме локальных и глобальных переменных, в программах PHP можно использовать статические переменные. Если переменную внутри функции объявить статической, ее значения не будут изменяться после вызова функции. Ниже приведен пример работы пользовательской функции со статистической переменной, реализующей счетчик:

```
<?php
   function function_counter()    {
```

```
    static $counter;
    $counter++;
    echo $counter;
  }
  for ($i = 0; $i++<8;) function_counter();
?>
```

Программа в данном примере выведет строку

```
12345678
```

Если удалить объявление статической переменной, отобразится строка

```
11111111
```

Причина в том, что при каждом новом вызове функции `function_counter()` переменные внутри функции будут инициализироваться, вместе с тем переменным присваивается значение `0`. Далее в нашем примере это значение инкрементируется и выводится оператором `echo`.

Резюме

Итак, мы познакомились с основными синтаксическими конструкциями языка PHP. Конечно, многое осталось вне нашего обсуждения, но теперь вы сможете реализовать практически все возможности, предоставляемые PHP, по обработке данных любого типа. Кроме того, вы знаете, как объявляются переменные, организуются циклы и ветвления при выполнении сценария, создаются и используются функции.

В следующей главе мы продолжим знакомиться со средствами языка PHP и научимся создавать настоящие веб-приложения, реализующие самые различные интернет-сервисы. Мы рассмотрим следующие процессы: организацию сеансов работы со сценарием, создание cookie-файлов, передачу данных между сценариями и получение данных из форм, размещенных на страницах веб-сайтов.

Глава 6

Веб-приложения на языке PHP

Работа с формами

Загрузка и обработка файлов

Подключение и работа с базами данных MySQL

Сеансы

Работа с cookie

Отправка почты

Резюме

В предыдущей главе мы познакомились с синтаксисом языка PHP и освоили простейшие конструкции из операторов, позволяющие выполнять обработку данных в сценариях. Теперь настала пора, используя полученные знания, построить собственное веб-приложение.

В этой главе мы научимся передавать в сценарии PHP данные, вводимые пользователем в формы на веб-страницах, и обрабатывать их, отображать результаты в окне браузера. Иными словами, мы научимся превращать статические веб-сайты в динамические, интерактивные веб-приложения, наиболее широко используемые при создании профессиональных веб-сайтов.

Чтобы обеспечить работу такого приложения, на практике часто приходится сохранять введенные пользователем данные в специальном хранилище — базе данных, — чтобы потом можно было воспользоваться ими при повторных посещениях сайта, работе с различными ресурсами и сервисами Интернета, требующими регистрации посетителя, и т. п. Для этого существует специальная технология, называемая «базы данных MySQL» (в просторечии — «мускул»), которая представляет собой специальный сервер MySQL, обеспечивающий работу с базами данных, в которые сохраняются все сведения, используемые веб-приложениями в процессе работы.

Для работы с MySQL язык PHP предоставляет множество функций, позволяющих прямо из сценария пополнять базу данных, обновлять ее, извлекать из нее данные и, наоборот, удалять их при необходимости. Мы познакомимся с некоторыми из этих функций, реализующих основные операции с базами данных. Далее вы сможете сами пополнить свой арсенал работы с MySQL, обратившись к каким-либо справочным материалам. Например, на сайте `http://php.su` существуют специальные средства поиска сведений по функциям PHP, и вы без труда найдете там нужную информацию.

Наконец, мы освоим работу с cookie-файлами и научимся организовывать сеансы работы пользователя с веб-сайтом. Это позволит нам организовывать авторизованный доступ к его ресурсам с момента регистрации на сайте и открытия сеанса до момента завершения работы, то есть выхода из сеанса.

Таким образом, мы пройдем все этапы построения динамического веб-сайта. Возможно, он будет слишком простым для настоящего профессионального веб-приложения, но научит вас всем принципам программирования соответствующих сценариев PHP.

Начнем с обсуждения работы с формами на веб-страницах.

Работа с формами

Рассмотрим простейшую веб-страницу, содержащую форму из двух элементов: поля ввода и кнопки отправки данных (листинг 6.1).

Листинг 6.1. Простая форма веб-страницы

```
<html>
  <head>
     <title>
       Простая форма
     </title>
  </head>
  <body>
     <center>
         <h1> Пример формы </h1>
         <form method="post"  action="simple_form.php">
            Укажите свое имя
            <input name="name" type="text">
            <br>
            <input type=submit value="OK">
         </form>
     </center>
   </body>
</html>
```

Сохраните код этого листинга в файл формата `simple_form.html` в папке examples и отобразите в окне браузера, введя `http://localhost/examples/simple_form.html` в его адресную строку (рис. 6.1).

Как вы помните из первых глав книги, в которых мы обсуждали язык HTML, при нажатии кнопки `OK` выполнится запрос сценария `simple_form.php`, указанного атрибутом `ACTION` формы, и данные, введенные пользователем, передадутся сценарию для обработки. Как сценарий получит указанные данные? Для этого в языке PHP предусмотрены так называемые суперглобальные массивы, к которым вы можете обратиться из любого места сценария. В таких случаях используются массивы `$_GET` и `$_POST`, содержащие передаваемые данные при применении соответственно методов `GET` и `POST` (этот метод, как вы помните, указывается атрибутом `METHOD`). Вы также можете использовать суперглобальный массив

$_REQUEST, включающий данные из обоих массивов. Приведем пример его использования в сценарии обработки нашего запроса (листинг 6.2).

Листинг 6.2. Сценарий обработки простой формы

```
<html>
  <head>
     <title>
       Сценарий обработки простой формы
     </title>
  </head>
  <body>
     <center>
         <h1> Чтение данных, введенных пользователем </h1>
         Вы ввели имя:
         <?php
           echo $_POST["name"];
         ?>
     </center>
   </body>
</html>
```

Сохраните код сценария в файле simple_form.php в папке examples, откройте в браузере файл simple_form.html, введите имя в поле формы и запустите сценарий, нажав кнопку OK (см. рис. 6.1). Результат представлен на рис. 6.2.

Как видите, введенное имя благополучно передалось в сценарий обработки.

Таким же образом можно передать в сценарий обработки данные формы любой сложности. Сейчас мы займемся этим, но вначале сделаю одно замечание.

Как вы могли заметить в приведенном выше примере, мы построили веб-приложение из двух файлов: один содержал форму, другой — сценарий ее обработки. Однако это не единственный и не самый распространенный способ написания веб-приложений.

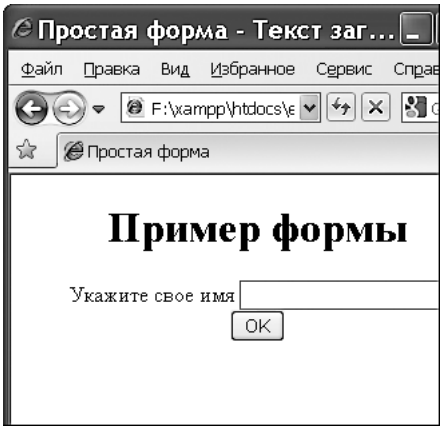


Рис. 6.1. Простая форма ввода имени

Общепринятый способ состоит в объединении таких файлов, то есть во включении в единый файл кода HTML для ввода данных и сценария для их последующей обработки. В листинге 6.3 представлен сценарий, объединяющий код из листингов 6.1 и 6.2.

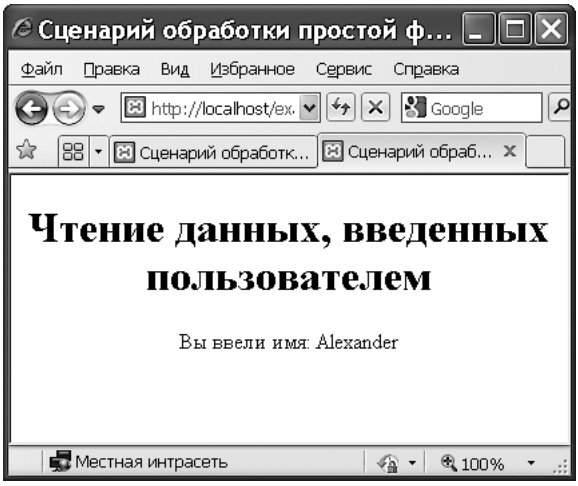


Рис. 6.2. Результат обработки формы

Листинг 6.3. Веб-приложение в едином файле

```
<html>
  <head>
     <title>
       Сценарий обработки простой формы
     </title>
  </head>
  <body>
     <center>
         <?php
// Этот оператор if управляет выполнением нашего веб-приложения
              if(isset($_REQUEST["name"]))
                {
         ?>
                   <h1> Чтение данных, введенных пользователем </h1>
                   Вы ввели имя:
         <?php
```

```
// Это оператор из листинга 6.2 для отображения введенных в форму данных
                  echo $_POST["name"];                    }
               else
                 {
         ?>
// Это форма из листинга 6.1 для ввода данных
         <h1> Пример формы </h1>
         <form method="post"  action="simple_form_script.php">
            Укажите свое имя
            <input name="name" type="text">
            <br>
            <input type=submit value="OK">
         </form>
      <?php
            }
       ?>
     </center>
   </body>
</html>
```

Несмотря на устрашающий вид, сценарий весьма простой и содержит все тот же самый HTML-код из листинга 6.1, в который с помощью тегов `<?php` и `?>` включены операторы PHP, обрабатывающие введенные данные. Чтобы браузер знал, когда ему следует отображать форму ввода данных, а когда — результаты обработки введенных данных, в сценарий включен условный оператор

```
if(isset($_REQUEST["name"]))
```

Функция `isset()` в условном выражении оператора `if` определяет существование переданной веб-приложению переменной: если переменная отсутствует, она возвращает значение `false`, иначе — `true`.

На первом шаге работы веб-приложения, до ввода данных пользователем своего имени, элемент массива `$_POST["name"]` не определен и функция возвращает `false`. При этом исполнение сценария переходит на блок `else`, содержащий форму ввода. После того как вы введете данные и нажмете кнопку OK, вызывается повторное исполнение веб-приложения, но теперь элемент `$_POST["name"]` определен, и исполняются операторы, стоящие после `if`, которые просто отображают введенное имя.

Как видите, все просто и эффективно, и мы избежали повторного ввода HTML-кода в обоих файлах — в документе HTML и сценарии PHP. Вы можете сохранить этот код в файле `simple_form_script.php` в папке examples, а затем открыть его, введя `http://localhost/examplex/simple_form_script.php` в адресную строку браузера. Результат будет аналогичным представленному на рис. 6.1 и 6.2.

Теперь приведем пример более сложной формы, включающей в себя все определенные в языке HTML элементы (листинг 6.4).

Листинг 6.4. Веб-приложение в едином файле

```
<html>
  <head>
     <title>
       Первое веб-приложение
     </title>
  </head>
  <body>
     <center>
         <?php
// Этот оператор if управляет выполнением нашего веб-приложения
              if(isset($_POST["login"]))
                {
         ?>
                  <h1> Вы ввели: </h1>
         <?php
// Это операторы для отображения введенных в форму данных
                 echo "<br> Логин: ", $_POST["login"],
                      "<br> Пароль: ", $_POST["password"],
                      "<br> Ваши увлечения: ", $_POST["preferences"],
                      "<br> Ваш любимый вид спорта: ", $_POST["sport"],
                      "<br> Вы занимаетесь спортом: ", $_POST["sportsmen"],
                     "<br> Вы хотите получать рассылку: ", $_POST["message"];
                 }
                else
```

```
                {
        ?>
<! Это форма для ввода данных !>
        <h1> Пример формы </h1>

        <form method="post"  action="complex_form.php">
           Логин
           <input name="login" type="text">
           <br><br>Пароль
           <input name="password" type="password">
           <br><br>Перечислите свои увлечения <br>
           <textarea name="preferences" rows="3" cols="30">
1.
2.
3.
           </textarea>
           <br><br>Выберите любимый вид спорта
           <Select name="sport">
              <option>Футбол</option>
              <option>Баскетбол</option>
              <option>Волейбол</option>
           </select>
           <br><br><br><br>Вы занимаетесь спортом?
           <input name="sportsmen" type="radio" value="Да"> Да
           <input name="sportsmen" type="radio" value="Нет"> Нет
           <br> Я хочу получать уведомления о новых событиях
           <input name="message" type="checkbox" value="Да">
           <br><br>
           <input type=submit value="OK">
        </form>
     <?php
        }
      ?>
    </center>
```

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

Как и ранее, сохраним этот сценарий в файл `complex_form.php` в папке examples и откроем в браузере, введя `http://localhost/examples/complex_script.php` в его адресную строку. Результат представлен на рис. 6.3.

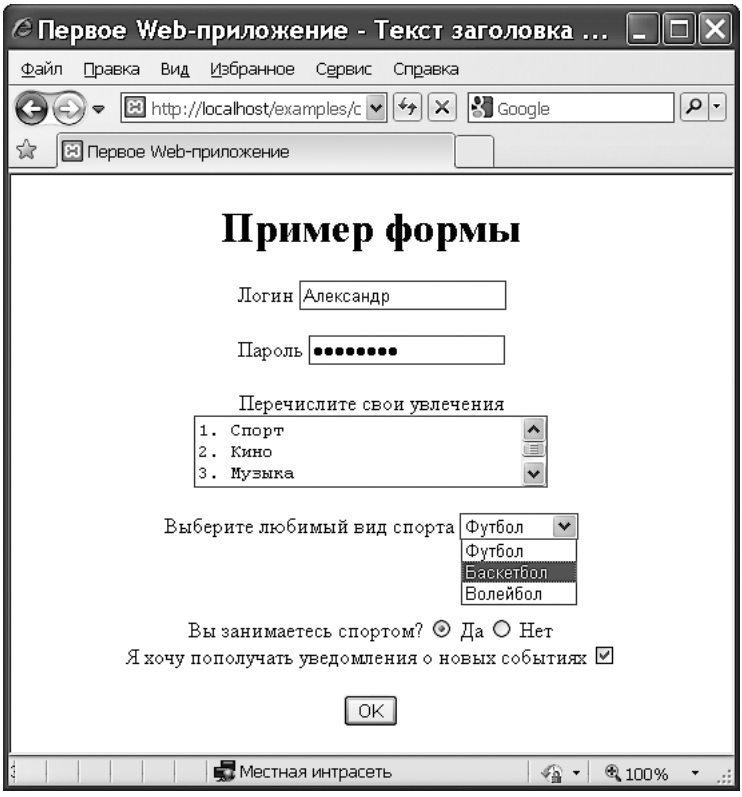


Рис. 6.3. Форма ввода данных в первое веб-приложение

Введя нужные данные, нажмите кнопку OK и запустите веб-приложение. Сценарий исполнится, и в окне браузера отобразятся введенные данные (рис. 6.4).

Вы сами сможете разобраться в работе сценария. По сути, он делает одну и ту же операцию со всеми данными, полученными из формы: извлекает из массива $_POST значения для элементов формы и отображает их оператором `echo` наподобие кода в листинге 6.2.

Немного сложнее в веб-приложения загружаются файлы. Остановимся на этом вопросе отдельно.

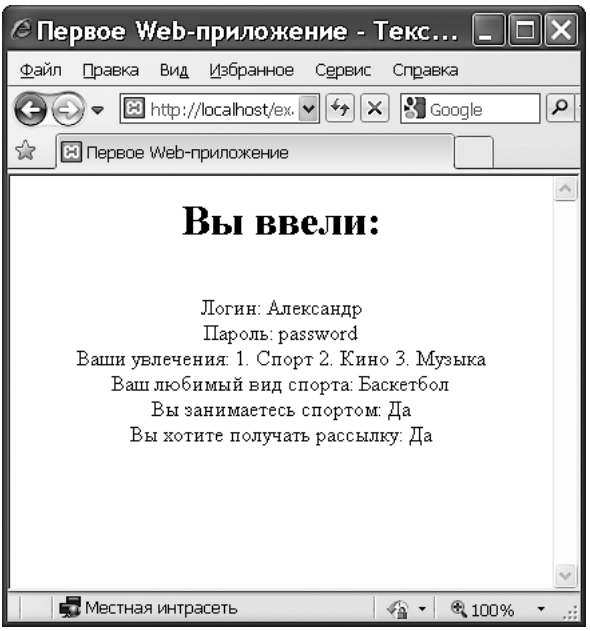


Рис. 6.4. Сценарий отобразил полученные из формы данные

Загрузка и обработка файлов

Для загрузки файлов в веб-приложение используется следующий тег HTML:

```
<input type="file">
```

Этот тег помещается в форму для особого типа данных, определяемого атрибутом `ENCTYPE="multipart/form-data"` (листинг 6.5).

Листинг 6.5. Форма для загрузки файлов

```
<html>
  <head>
     <title>
       Форма загрузки файлов
     </title>
  </head>
  <body>
     <center>
         <h1> Загрузка файла </h1>
```

```
        <form
                enctype="multipart/form-data"
                method="post"
                action="form_file.php">
           Укажите загружаемый файл
            <input name="myfile" type="file">
            <br>
           <input type=submit value="Загрузить">
        </form>
     </center>
   </body>
</html>
```

Сохраним его в файле `form_file.html` в папке examples и откроем в браузере (рис. 6.5).

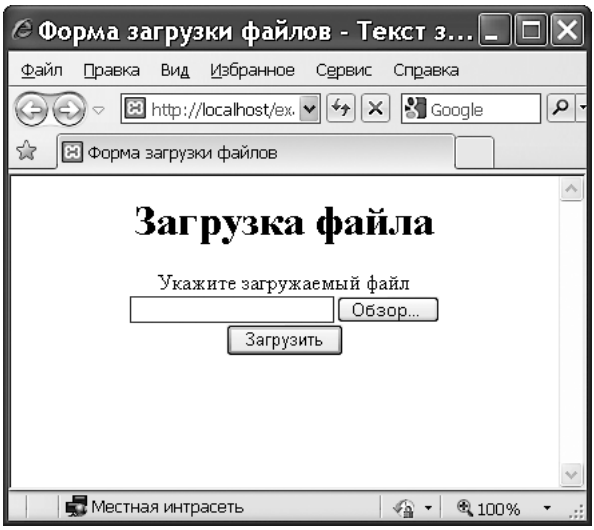


Рис. 6.5. Форма загрузки файла

Как видим, форма принимает имя загружаемого файла и при нажатии кнопки подтверждения Загрузить обращается к сценарию `form_file.php` (листинг 6.6).

Листинг 6.6. Сценарий обработки загруженного файла

```
<html>
  <head>
```

```
    <title>
      Сценарий обработки загруженного файла
    </title>
  </head>
  <body>
    <center>
        <h1> Отображение данных файла  </h1>
        Загруженный файл содержит такие данные <br>
        <?php
           $handle=fopen($_FILES["myfile"]["tmp_name"], "r");
           while (feof($handle)==0)
             {
                $file_content=fgets($handle);
                echo $file_content, "<br>";
             }
           fclose($handle);
        ?>
    </center>
   </body>
</html>
```

Рассмотрим, как работает этот сценарий. После успешной загрузки файла он сохраняется в каталог `tmp_name` сервера, а имя файла сохраняется в суперглобальном массиве

`$_FILE["имя_поля_формы_с именем_файла"]["tmp_name"].`

Поле формы для ввода имени файла в листинге 6.5 имеет следующий вид:

`<input name="myfile" type="file">.`

Таким образом, имя нашего файла хранится в ячейке массива

`$_FILES["myfile"]["tmp_name"].`

Чтобы получить содержимое файла, сначала его нужно открыть, то есть определить переменную для доступа к указанному ресурсу. Эта операция выполняется с помощью функции `fopen()` **со следующими обязательными параметрами:**

```
resource fopen ( string filename, string mode)
```

Здесь `filename` — это имя файла, а `mode` — режим работы с файлом.

Заданный в вышеприведенном примере режим "r" разрешает доступ к файлу, если же этому параметру установить "w", будет разрешена запись в файл. Функция fopen() возвращает так называемый дескриптор файла, имеющий тип данных ресурса и позволяющий получить доступ к данным в файле.

Итак, создадим с помощью Блокнота текстовый файл MyFile.txt, сохраним в папке examples, выберем его в форме загрузки файла и нажмем кнопку Загрузить. В ответ наше веб-приложение загрузит файл и приступит к его обработке. Для этого в листинге 6.5 используется следующий код:

```
$handle=fopen($_FILES["myfile"]["tmp_name"], "r");
 while (feof($handle)==0)
   {
     $file_content=fgets($handle);
     echo $file_content, "<br>";
   }
 fclose($handle);
```

Первый оператор вызывает функцию fopen(), которая открывает файл, а в переменную $handle сохраняется дескриптор файла.

Далее с помощью цикла while мы читаем содержимое файла порцию за порцией, вызывая на каждой итерации функцию fgets(). Общий синтаксис этой функции следующий:

```
string fgets ( resource handle [, int length])
```

Здесь handle — дескриптор файла, а length равно числу байтов, читаемых при вызове функции fgets(). Таким образом, функция fgets() возвращает строку длиной length-1 (считая первый разряд нулевым); по умолчанию length равно 1024. Чтобы завершить цикл по прочтении файла, необходимо использовать функцию feof(), которая равна true при чтении конца файла и false в противном случае.

В результате наше веб-приложение отобразит результат, приведенный на рис. 6.6.

Таким образом, наше приложение заработало: мы прочли файл и сумели отобразить его содержимое в окне браузера. С помощью других функций PHP мы могли бы сохранить полученные данные на сервере, чтобы впоследствии обращаться к ним по мере надобности. Но сохранение данных в файле — не лучший, хотя и приемлемый, выбор. Гораздо эффективнее использовать для этого базы данных, и сейчас мы их рассмотрим.

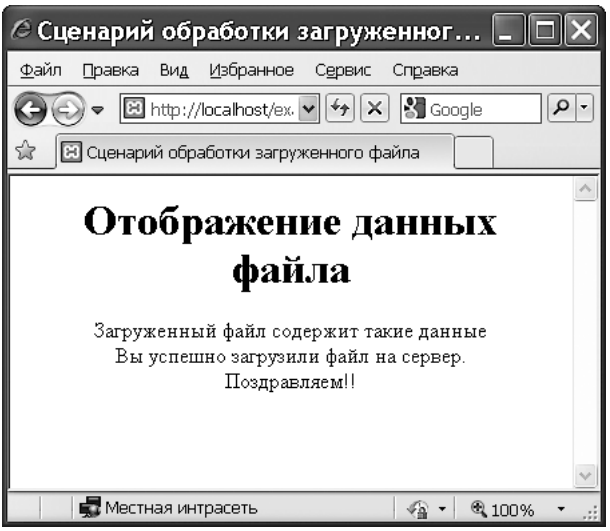


Рис. 6.6. Результат чтения файла **MyFile.txt**

Подключение и работа с базами данных MySQL

В гл. 3 этой книги мы познакомились с основными принципами работы баз данных MySQL и освоили технику запросов к базе данных средствами языка Perl. Теперь сделаем то же самое для языка PHP: опишем, как в сценарии PHP выполняется подключение к MySQL и каким образом из базы данных извлекаются и записываются данные.

Подключение к серверу MySQL и выбор базы данных

Чтобы начать работу с базами данных MySQL, предварительно следует удостовериться, что на вашем компьютере (или на удаленном сервере) запущена служба сервера MySQL. На локальном компьютере это можно сделать из панели управления пакета, который вы используете для работы серверов Apache и MySQL, — например, Денвер, описанный в гл. 3, или XAMMP. Мы полагаем, что вы уже освоили процедуру запуска, описанную в гл. 3, и сразу приступим к изучению средств PHP.

Подключение к серверу баз данных MySQL выполняется следующей командой:

```
resource mysql_connect ([string server [,string username [,string password
[,bool new_link [,int client_flags]]]]] )
```

Как видим, функция `mysql_connect()` возвращает значение ресурсного типа данных, содержащее указатель на соединение с сервером MySQL. В случае неудачи возвращается значение `false`.

Параметр `server` содержит имя узла, на котором запущен сервер. По умолчанию его значение равно `localhost`, но в случае необходимости можно указать путь и номер порта для подключения к серверу, например: `/path_to_MySQL_socket` или `database:3306`. В любом случае значение данного параметра вам должен указать интернет-провайдер, который предоставляет вам эту услугу. Для локального сервера MySQL используется значение по умолчанию.

Параметры `username` и `password` задают, соответственно, логин и пароль для подключения к серверу MySQL. Их вы также должны либо получить от провайдера этой услуги, либо установить в учетной записи пользователя локального сервера MySQL с помощью панели администрирования.

Остальные параметры менее существенны. Задание значения `true` параметру `new_link` приводит к открытию нового соединения с MySQL в случае повторного обращения к функции `mysql_connect()`. По умолчанию установлен режим, при котором функция просто возвратит указатель на уже открытое соединение. Параметр `client_flags` задает особые режимы работы с сервером, нам он не понадобится.

Если поместить перед вызовом функции `mysql_connect()` символ @, то в случае возникновения ошибки при открытии соединения пользователь не получит соответствующее сообщение. Это удобное средство, поскольку такие сообщения, как правило, недружественны к пользователю. Вместо этого следует при каждом открытии соединения проверять полученное значение функции и выдавать собственные сообщения, более понятные потребителям вашего сценария.

Соединение с сервером MySQL закрывается либо по завершении сценария, либо путем вызова функции `mysql_close()`.

Рассмотрим пример корректного открытия и закрытия соединения с MySQL:

```
<?php
     $connection = mysql_connect("localhost", "user_MySQL", "password_
MySQL");
    If($connection==false)
        die("Невозможно подключиться к MySQL:  " . mysql_error());
    echo "Соединение с MySQL выполнено успешно";
    mysql_close($connection);
?>
```

Здесь в случае неуспешного выполнения функции MySQL вызывается функция `die()`, которая выводит сообщение об ошибке и завершает работу сценария. Данное сообщение содержит ваш текст, к которому операцией конкатенации текстовых строк («.») добавляется текст сообщения об ошибке, возвращаемый функцией `mysql_error()`.

После соединения с сервером MySQL нужно выбрать базу данных, которая вам нужна для последующих действий. Это делается с помощью оператора

```
bool mysql_select_db ( string имя_базы_данных [, resource указатель_соединения_с_MySQL] )
```

В случае успешного выбора базы данных функция возвращает значение `true`, неуспешного — `false`. Для выбора базы данных функции передаются имя базы данных и указатель на соединение с сервером MySQL, который мы получили от функции `myssql_connect()` в предыдущем примере. Если опустить этот указатель, будет использовано последнее открытое соединение с MySQL, а в случае его отсутствия будет сделана попытка открыть соединение с MySQL, причем с помощью параметров, которые использует по умолчанию функция `mysql_connect()`.

Если вызов функции `mysql_select_db()` завершится успешно, то все последующие запросы к базе данных, выполняемые из сценария, будут обращены к выбранной базе данных. Для выполнения таких запросов PHP предоставляет множество функций. Опишем некоторые из них, наиболее часто применяемые в сценариях. Если вам потребуются дополнительные возможности, вы можете обратиться к ресурсам сайта `www.php.su` или другим справочным материалам.

Работа с базой данных MySQL

Мы начнем наше обсуждение с метода создания новой базы данных средствами PHP. Для этого используется функция `mysql_query()`, выполняющая запросы к серверу MySQL. Она имеет следующий синтаксис:

```
resource mysql_query ( string mysql_query [, resource mysql_connection])
```

Ошибка при выполнении функции возвращает значение `false`, иначе, в случае запроса на создание базы данных, возвращается `true` (для других запросов возвращаемое значение может содержать указатель на результат запроса).

Параметр `mysql_query` содержит строку запроса к базе данных, а `mysql_connection` — указатель на соединение с сервером MySQL. Если соединение

отсутствует, делается попытка соединиться с MySQL, аналогично тому, как это делает вышеописанная функция `mysql_connect()`. Строка запроса формируется по правилам, которые мы обсуждали в гл. 3. При создании новой базы данных запрос записывается следующим образом:

```
CREATE DATABASE IF NOT EXIST db_name
```

Здесь параметр `db_name` содержит имя создаваемой базы данных. Согласно запросу она будет создана, если такой базы данных еще не существует.

Приведем пример кода, выполняющего создание базы данных MySQL:

```
$mysql_query="CREATE DATABASE IF NOT EXIST my_db"
$result=mysql_query($mysql_query);
If((!$result)
     die("Ошибка создания базы данных");
```

Суть этого кода достаточно проста: мы формируем в переменной `$mysql_query` строку запроса к MySQL на создание базы данных с именем `my_db`, после чего обращаемся к функции `mysql_query()`. Если запрос на создание базы данных выполнится с ошибкой, выполнение сценария прекращается.

После создания базы данных мы должны заполнить ее информацией. Для этого в базах данных используются таблицы, как вы помните из обсуждения в гл. 3. Таблица базы данных создается тем же самым запросом, который мы использовали в гл. 3 при описании баз данных:

```
CREATE TABLE table_name (column_1, column_2, … column_N)
```

В запросе задаются имя таблицы (переменная `table_name`) и имена столбцов (переменные `column_...`).

Таким же образом формируется запрос на добавление в таблицу данных:

```
INSERT INTO table_name (column_1, column_2, …, column_N) VALUE (value_1,
value_2, …, value_N)
```

Здесь переменные `value_1, ..., value_N` содержат значения, добавляемые в столбцы таблицы, перечисленные в операторе после ее имени. При добавлении значений ко всем столбцам их список можно опустить.

Более сложные варианты запросов содержатся в гл. 3, но нам достаточно тех, которые приведены выше. Составим сценарий, выполняющий чтение данных из формы на рис. 6.3 и запись ее в базу данных (листинг 6.7).

Листинг 6.7. Веб-приложение записи данных в базу данных

```
<html>
  <head>
     <title>
       веб-приложение записи введенных данных в БД
     </title>
  </head>
  <body>
     <center>
         <?php
// Этот оператор if управляет выполнением нашего веб-приложения
              if(isset($_POST["login"]))
                {
         ?>
                   <h1> Запись в базу данных </h1>
         <?php
// Открываем соединение с MySQL
              $connection=mysql_connect("localhost", "user", "password");
              if($connection==false)
                     die("Соединение с MySQL невозможно");
// Создаем базу данных my_db
             $query="CREATE DATABASE IF NOT EXISTS my_db";
              $result=mysql_query($query);
              if($result==false)
                     die("Ошибка создания базы данных");
// Выбираем созданную базу данных my_db
              $my_db=mysql_select_db("my_db", $connection);
              if($my_db==false)
                     die("Ошибка выбора базы данных");
// Создаем таблицу
              $query="CREATE TABLE account (login text, password text,
preferences text, sport text, sportsmen text, message text)";
              $result=mysql_query($query);
              if($result==false)
```

```
                    die("Ошибка создания таблицы");
// Читаем данные из запроса
              $login=$_POST["login"];
              $password=$_POST["password"];
              $preferences=$_POST["preferences"];
              $sport=$_POST["sport"];
              $sportsmen=$_POST["sportsmen"];
              $message=$_POST["message"];
// Добавляем данные в БД
        $query="INSERT INTO account  (login, password, preferences, sport,
sportsmen, message) VALUES ('$login', '$password', '$preferences', '$sport',
'$sportsmen', '$message')";
              $result=mysql_query($query);
              if($result==false)
                      die("Ошибка добавления данных");
// Сообщаем об успешном завершении сценария
               echo "Данные формы занесены в БД";
               mysql_close($connection);
                }
               else
                {
         ?>
<! Это форма для ввода данных !>
…
```

В этом листинге мы опустили часть, относящуюся к выводу формы. Остальная его часть включает все операции по созданию, выбору и пополнению базы данных, которые мы обсудили в этом разделе. Вы сами сможете найти соответствующие части кода, руководствуясь комментариями.

Сохраним данные в файле `forma_db.php` в папке examples и запустим веб-приложение, введя `http://localhost/examples/forma_db.php` в адресную строку браузера. После ввода данных в форму и нажатия кнопки OK мы получим сообщение об успешном занесении в созданную базу данных информации, внесенной в форму. Но как определить, что именно в ней теперь содержится? Для этого служит следующий запрос выборки данных:

```
SELECT name_1, name_2, … name_N FROM name_table
```

Здесь `name_...` — это имена столбцов таблицы с названием `name_table`. Вы уже знаете, как отправлять запросы к базе данных средствами PHP, так что уже готовы написать сценарий выборки данных и их отображения (листинг 6.8).

Листинг 6.8. Веб-приложение чтения и отображения информации в базе данных

```
<html>
  <head>
     <title>
       веб-приложение чтения данных в БД
     </title>
  </head>
  <body>
     <center>
      <h1> Чтение базы данных </h1>
         <?php
// Открываем соединение с MySQL
              $connection=mysql_connect("localhost", "user", "password");
              if($connection==false)
                     die("Соединение с MySQL невозможно");
// Создаем базу данных my_db
             $query="CREATE DATABASE IF NOT EXISTS my_db";
              $result=mysql_query($query);
              if($result==false)
                     die("Ошибка создания базы данных");
// Выбираем созданную базу данных my_db
              $my_db=mysql_select_db("my_db", $connection);
              if($my_db==false)
                     die("Ошибка выбора базы данных");
// Читаем таблицу
              $query="SELECT * FROM account";
              $result=mysql_query($query);
              if($result==false)
                     die("Ошибка выбора таблицы");
// Извлекаем и отображаем данные из таблицы
              echo "База данных содержит следующие сведения";
              $account=mysql_fetch_array($result);
               echo "<br> Логин: ", $account['login'],
```

```
                    "<br> Пароль: ", $account['password'],
                    "<br> Предпочтения: ", $account['preferences'],
                    "<br> Любимый вид спорта: ", $account['sport'],
                    "<br> Занятие спортом: ", $account['sportsmen'],
                    "<br> Получение уведомлений: ", $account['message'];

            mysql_close($connection);
        ?>
      </center>
  </body>
</html>
```

В этом сценарии мы создали запрос на извлечение данных:

```
$query="SELECT * FROM account";
```

Далее мы выполнили его средствами PHP:

```
$result=mysql_query($query);
```

Из полученного результата с помощью функции `mysql_fetch_array()` мы извлекли указанные данные в виде массива, содержащего все сведения, которые мы ввели в форму из предыдущего примера. Это и отображается в окне браузера после исполнения сценария (рис. 6.7).

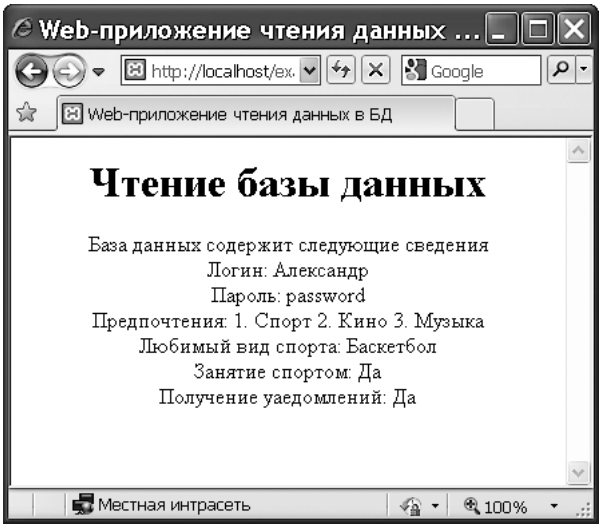


Рис. 6.7. Отображение информации в базе данных

Таким же образом, пользуясь запросами MySQL и функциями PHP, мы можем обновлять, удалять и редактировать информацию, хранимую в базе данных. Все эти операции вы без труда освоите самостоятельно. А мы двинемся дальше и освоим другое важнейшее средство PHP — создание сеансов, то есть среды, в которой сохраняются все сведения о работе данного пользователя с веб-приложением.

Сеансы

Каждое веб-приложение, как правило, состоит из нескольких страниц, содержащих сценарии. При загрузке каждой страницы переменные в этих сценариях очищаются, и для обмена данных между страницами можно использовать механизм передачи параметров, или базу данных, или запись информации в cookie-файлах. Однако наиболее удобным методом является использование сеансов.

При открытии сеанса для каждого пользователя на сервере формируется определенная среда, заданная совокупностью параметров, относящихся к данному пользователю. Например, там могут сохраняться какие-то настройки интерфейса, конфигурации отдельных сервисов и т. п. Соответствующие данные хранятся в суперглобальном массиве `$_SESSION`. Так, в нем можно сохранить выбранный цвет фона страницы:

```
$_SESSION['background']="зеленый";
```

теперь при переходе от одной страницы к другой вы сможете использовать этот параметр для установки нужного цвета фона.

Каждому сеансу присваивается уникальный идентификатор, который может использоваться при повторном открытии сеанса. Этот идентификатор можно сохранить, например, в cookie-файле, после чего при повторном посещении сайта вам не потребуется заново настраивать работу с его интерфейсом и сервисами.

Для открытия сеанса используется функция `session_start()`, не имеющая параметров. После этого становится доступным массив `$_SESSION`, и в нем вы можете сохранять свои данные, которые будут доступны из любой части веб-приложения до завершения сеанса. В листинге 6.9 приведен пример использования сеансов для создания счетчиков посещений страницы. Вы не сможете сделать это с помощью переменных сценария, поскольку они при каждом новом обращении обнуляются, но решить указанную проблему позволяет сохранение значения счетчика в массиве `$_SESSION`.

Листинг 6.9. Счетчик показов

```
<?php
    session_start();
```

```
?>
<html>
  <head>
     <title> Использование сеансов  </title>
  </head>
  <body>
     <center>
     <h1> Счетчик показов страницы </h1>
   <?php
           if(isset($_SESSION['counter'])==false)
                $_SESSION['counter']=0;
           $_SESSION['counter']++;
           echo  "Эту страницу открывали  ", $_SESSION['counter'],  "  раз";
       ?>
     </center>
  </body>
</html>
```

Сценарий в листинге достаточно прост. При каждом обращении к странице (например, нажатием клавиши F5) выполняется функция инициализации сеанса `session_start(0)`, после чего происходит либо инициализация счетчика, если это первое открытие страницы, либо инкрементация. На рис. 6.8 показан результат исполнения этого сценария.

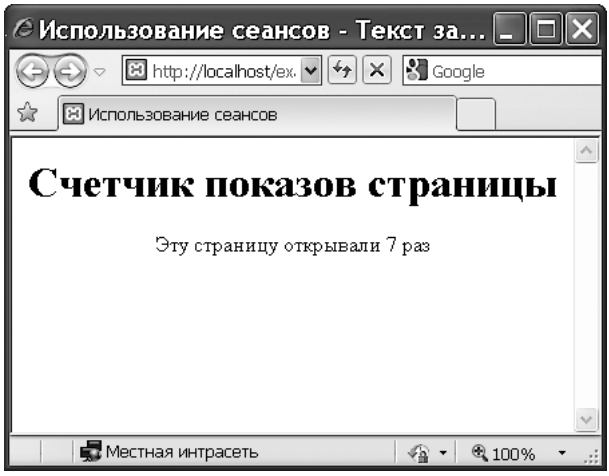


Рис. 6.8. При каждом показе страницы срабатывает счетчик

Иногда возникает потребность удалить данные, сохраненные в сеансе: не просто обнулить или очистить данные, а убрать соответствующий параметр из массива `$_SESSION`. Для этого применяется функция `unset()`. Например, использованный в предыдущем примере счетчик мы можем удалить следующим образом:

```
session_start();
unset($_SESSION['counter'];
```

Мы уже говорили, что каждому сеансу сопоставляется идентификатор сеанса, позволяющий обратиться к данным, сохраненным во время сеанса работы. Чтобы его получить или установить, можно использовать функцию `session_id()`, имеющую такой синтаксис:

```
string session_id ([string id])
```

Эта функция возвращает идентификатор текущей сессии, а если задан параметр `id`, замещает его значением идентификатора текущего сеанса. В последнем случае функцию `session_id()` следует вызывать до инициализации сеанса, то есть вызова функции `session_start()`. В строке идентификатора сеанса можно использовать только символы a–z, A–Z и 0–9.

Идентификатор сеанса обычно хранится в cookie-файле. Однако можно сконфигурировать PHP на передачу его в строке вызова сценария или использовать передачу через скрытое поле формы, имеющее имя `PHPSESSID`. Последний способ полезен в случаях, если в клиентском браузере отключено сохранение cookie, а PHP сконфигурирован без поддержки передачи идентификатора через строку вызова. Но мы не будем отвлекаться на эти методы, а сконцентрируемся на использовании cookie-файлов как наиболее удобного и широко используемого метода.

Работа с cookie

Файлы cookie содержат текстовые строки, которые сохраняются на клиентском компьютере и которые можно передать на сервер во время обращения. Тем самым вы можете сохранить, а потом передать информацию о текущем сеансе работы с веб-сервером из одного сценария в другой, не прибегая к упомянутым выше средствам передачи параметров через формы в адресной строке вызова сценария. Имейте в виду, что данные в cookie отправляются на сервер в виде заголовков HTTP-запросов, посылаемых сервером на клиентский компьютер, поэтому вызовы этой функции вы должны помещать *до* тегов HTML, поскольку заголовки HTTP отправляются клиенту до отправки HTML-кода страницы.

Для установления данных в cookie-файле используется функция `setcookie()`, имеющая следующий синтаксис:

```
int setcookie (string name [, string value [, int expire [, string path [,
string domain [, int secure]]]]])
```

Аргумент `name` задает имя cookie, а `value` — его значение. Необязательные аргументы можно пропустить с помощью их замены пустой строки "". Они определяют следующие параметры:

- `expire` — срок действия cookie;
- `path` — путь на сервере, в пределах которого cookie доступен;
- `domain` — домен, на котором доступен cookie;
- `secure` — указывает на использование cookie только при безопасном соединении.

Чтобы показать на примере, как работает механизм cookie, создадим страницу с кодом записи cookie (листинг 6.10).

Листинг 6.10. Сценарий установки cookie

```
<?php
    setcookie("color", "green");
?>
<html>
  <head>
     <title> Использование cookie </title>
  </head>
  <body>
     <center>
     <h1> Задание  cookie</h1>
   cookie установлен. Содержимое можно посмотреть <a href="read_cookie.php">
здесь </a>
     </center>
  </body>
</html>
```

В сценарии устанавливается cookie с названием `"color"` и значением `"green"`, а значения читаются в сценарии `read_cookie.php`, к которому можно обратиться щелчком на ссылке в странице. Сохраним сценарий в файле под именем `cookie.php` в папке examples. Теперь создадим сценарий чтения cookie (листинг 6.11).

Листинг 6.11. Сценарий чтения cookie

```
<html>
  <head>
     <title> Чтение cookie </title>
  </head>
  <body>
     <center>
     <h1> Чтение cookie </h1>
   <?php
          if(isset($_COOKIE['color'])!=false)
               echo "cookie имеет значение ". $_COOKIE['color'];
          else
               echo  "cookie не установлено";
       ?>
     </center>
  </body>
</html>
```

Здесь значения cookie извлекаются из суперглобального массива `$_COOKIE` и показываются пользователю. Сохраним этот сценарий в файле под именем `read_cookie.php` в папке `examples` и запустим первый из них, введя `localhost/examples/cookie.php` в адресную строку браузера. В отобразившейся странице (рис. 6.9, *слева*) щелкнем на ссылке и исполним сценарий в листинге 6.11. Результат представлен на рис. 6.9, *справа*.

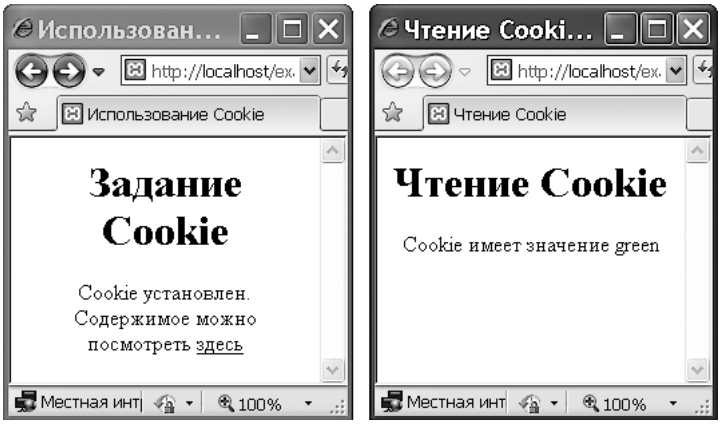


Рис. 6.9. Страницы установки *(слева)* и чтения *(справа)* cookie

Как видно из кода в листинге 6.11, установленное значение cookie мы извлекаем из суперглобального массива `$_COOKIE`. Почему мы не извлекли его прямо в файле `cookie.php`? Да потому, что эти значения передаются в браузер при обращении к веб-серверу, то есть требуют перезагрузки страницы с сервера.

Чтобы удалить установленное cookie, следует вызвать функцию `setcookie()` с теми же параметрами, что и при установке, но с пустой строкой "" в качестве аргумента `value`. Например, чтобы удалить установленное в листинге 6.10 значение cookie, следует вызвать функцию:

```
setcookie("color", "");
```

cookie можно организовывать в массивы, например:

```
setcookie("cookie[1]", "color");
setcookie("cookie[2]", "data");
setcookie("cookie[3]", "time");
```

Будет сформирован массив `$_COOKIE["cookie"]`, с которым можно работать обычным образом.

Итак, мы научились работать с cookie, и теперь нам ничто не мешает организовывать авторизованные сеансы работы с веб-сервером, передавая данные сеанса через cookie. Вы сами можете убедиться в их эффективности, когда браузер автоматически регистрирует вас на веб-сайтах и устанавливает нужные вам параметры, — все это результат применения cookie.

Напоследок обсудим еще одну возможность, обеспечиваемую PHP, — отправку почты. Данный инструмент используется на очень многих сайтах, как профессиональных, так и любительских, домашних страницах.

Отправка почты

Для отправки почтовых сообщений необходимо включить эту возможность для сервера. Это делается в файле `php.ini`, в котором следует отредактировать секцию `php.ini`:

```
[mail function]
;For Win32 only.
smtp=localhost
; For Win32 only.
sendmail_from=me@example.com
```

Здесь в параметре `smtp` следует указать адрес сервера SMTP, а в `sendmail_from` — обратный адрес отправителя писем. Если вы будете размещать свои сценарии на удаленном сервере, эти сведения вам предоставит ваш интернет-провайдер. Но если вы захотите протестировать работу почтовой службы локально, вам придется развернуть у себя на компьютере серверы SMTP и POP3. В случае использования пакета XAMPP это будет сделано по умолчанию при установке пакета, вам потребуется только запустить работу сервера Mercury из панели управления XAMPP. Далее вы сможете либо создать новый почтовый ящик в клиенте, либо воспользоваться встроенными по умолчанию ящиками, например `postmaster@localhost`.

Отправку писем в сценарии PHP выполняет функция `mail()`, имеющая следующий синтаксис:

```
bool mail (string to, string subject, string message [, string additional_
headers [, string additional_parameters]])
```

Здесь параметр `to` содержит адрес получателя почты, `subject` — тему письма, а `message` — само сообщение. Остальные параметры необязательны и применяются для отправки дополнительных заголовков послания (`additional_headers`) и параметров (`additional_parameters`).

Функция `mail()` возвращает значение `true` при успешном принятии письма для доставки и `false` в ином случае (отметим, что это не означает успешной *доставки* письма конечному получателю).

В листинге 6.12 представлено веб-приложение, которое выполняет отправку почты со страницы сайта, используя введенные в поля формы сведения.

Листинг 6.12. Отправка почты с веб-страницы

```
<html>
<head>
   <title>
     веб-приложение отправки писем
   </title
</head>
<body>
   <center>
       <?php
// Этот оператор if управляет выполнением нашего веб-приложения
           if(isset($_POST["lb"]))
               {
```

```
        ?>
                <h1> Отправка почты </h1>
        <?php
                if(!mail($_POST["to"], $_POST["subject"], $_POST["message"]))
                    echo "Ошибка отправки письма";
                else
                    echo "Письмо отправлено";
             }
           else
             {
        ?>
        <h1> Форма для ввода письма </h1>
        <form method="post"  action="mail.php">
             Кому:  <input name="to" type="text"> <br>
             Тема:  <input name="subject" type="text"> <br>
             <textarea name="preferences" rows="5" cols="40"> </textarea>
<br>
             <input name="lb" type="hidden" value="Yes"><br>
             <input type="submit" value="Отправить">
        </form>
      <?php
         }
       ?>
     </center>
  </body>
</html>
```

Сценарий достаточно прост, но обратите внимание на следующее. Для управления исполнением приложения мы использовали скрытое поле формы:

```
<input name="lb" type="hidden" value="Yes">
```

Это поле имеет имя `lb` и в самом начале исполнения сценария проверяется его значение:

```
 if(isset($_POST["lb"]))
```

...

При первом открытии страницы значение `$lb` не определено, поэтому будет исполнен блок `else` оператора `if`, который содержит код формы ввода почтового адреса и сообщения. После подтверждения отправки письма сценарию передается значение скрытого поля, которое теперь равно `Yes`, и исполняется функция `mail()`, параметры которой извлекаются из суперглобального массива `$_POST`.

Если вы зададите все данные формы правильно, то после нажатия кнопки Отправить отобразится сообщение об успешной отправке письма. Обратите внимание, что для этого данную страницу либо размещают на удаленном сервере с поддержкой почтового сервиса, либо вы должны воспользоваться локальными почтовыми службами, развернутыми на вашем компьютере.

Резюме

Итак, в этой главе мы закрепили теоретические сведения о языке PHP, полученные в гл. 5. Теперь вы умеете строить веб-приложения, содержащие формы ввода данных, запоминать переданную информацию в базе данных MySQL и работать с ней средствами PHP, организовывать сеансы работы с сайтом и сохранять параметры сеанса в cookie и даже отправлять почту с веб-страницы. Этого багажа вполне достаточно для построения полнофункционального сайта, реализующего основные сервисы, предоставляемые посетителям веб-порталов.

Однако рассмотренными возможности PHP не исчерпываются. В следующей главе мы обсудим технику асинхронных запросов (Ajax), которая специально разработана для получения клиентом информации большого объема в процессе работы с сайтом, причем так, чтобы посетитель сайта не замечал никаких неудобств. Это совершенно необходимая вещь, без которой не обходится ни один полноценный интернет-магазин, предлагающий покупателям обширный каталог своих товаров. Средства Ajax позволят вам так построить обмен данными с сайтом, что пользователь даже не заметит и не почувствует каких-либо задержек в работе сайта, связанных с закачкой данных большого объема, что чрезвычайно важно для работы коммерческого сервиса.

Глава 7

Технология AJAX

Как работает AJAX

Работа с данными XML

Работа с MySQL

Резюме

В предыдущей главе мы рассмотрели несколько веб-приложений, созданных на основе языка PHP. Это были простые, учебные примеры, назначение которых состоит в демонстрации технических приемов, применяемых при написании сценариев PHP. Однако если вы займетесь разработкой «настоящего», имеющего практическое назначение веб-приложения, перед вами немедленно встанет задача, которую можно назвать важнейшей из всех, встающих перед разработчиком: обеспечить пользователю (на сленге «юзабилити», от англ. usability — «удобство, практичность, простота использования»), работающему с вашим приложением, удобный интерфейс.

Веб-приложение состоит из серверного сценария, исполняемого на интернет-сервере (например, сервера Apache), и клиента, работающего на локальном компьютере. Последний чаще всего использует браузер, подключенный к соответствующему сервису Интернета. В этой связке клиент обеспечивает интерфейс работы с пользователем, а сервер выполняет обработку данных, пересылаемых ему клиентом. Например, если вы работаете с поисковой машиной, то с помощью браузера вы посылаете серверу запрос на поиск данных определенного вида, а сервер выполняет этот поиск в своей базе данных и передает вам информацию. Все это взаимодействие вы наблюдали при исполнении примеров из предыдущей главы.

Но что будет, если объем передаваемых клиенту данных будет велик или поиск в базе данных будет достаточно длительным? Тогда, если вы останетесь в рамках той технологии общения клиента с сервером, которую вы использовали в своих примерах, пользователь вашего веб-приложения будет испытывать большие неудобства, связанные с длительными задержками при передаче по сети данных, вследствие того, что они большого объема, или вследствие загруженности сервера. Эта проблема особо остро встала буквально в последние годы, когда наметилась явная тенденция использования Интернета в тех сферах, для которых он не был предназначен.

В самом деле, изначально Интернет создавался как средство для извлечения данных небольшого объема и передачи их клиентам по маломощным линиям связи (кто не помнит эпоху «диал-апа», когда все мы подключались к Сети через телефонные провода, проложенные в эпоху, когда понятия «Интернет» не было и в помине). Однако нынешние веб-приложения стали претендовать на те же функции, которыми ранее оснащались «обычные» приложения, работающие или на пользовательском компьютере, или в локальной сети офиса.

Чтобы гарантировать юзабилити веб-приложений, которые обеспечивают работу с большими объемами данных или значительно нагружают сервер, была придумана технология AJAX, которая расшифровывается как Asynchronous JavaScript and XML (Асинхронный JavaScript и XML). И сейчас мы приступим к его освоению.

Как работает AJAX

Суть технологии AJAX, как это и следует из ее названия, состоит в создании запросов серверу, которые исполняются в асинхронном режиме. Это значит, что серверная часть веб-приложения, использующего AJAX, работает независимо от клиентской части. После того как клиент послал запрос серверу, последний начинает его обработку, а клиент продолжает свою работу, не вынуждая пользователя дождаться ответа. Например, пользователь может вводить данные в форму, отображенную в окне клиента (в данном случае браузера), а клиент, не дожидаясь окончания ввода и нажатия кнопки подтверждения формы, отсылает данные на сервер, который скрытно от пользователя обрабатывает их и отсылает результаты клиенту.

Что это дает? В наших простых примерах мы получали ответ от сервера практически мгновенно, и никакого AJAX нам не требовалось. Но в случае, допустим, решения задачи поиска, при котором сервер должен выполнить большую работу и переслать объемные данные, мы сможем снизить время ожидания результатов, если сервер будет выполнять поиск параллельно с вводом данных в форму поиска.

Ярким примером сказанному является поиск по каталогу. По мере того как пользователь открывает щелчками кнопки мыши новые разделы, веб-приложение с помощью AJAX загружает информацию раздела, предполагая ее использование при последующих шагах выбора. Это очень удобно при реализации интернет-магазинов: посетитель магазина ищет какой-то товар в каталоге, скажем автомобиль, и сначала выбирает производителя, затем модель, потом какие-то модификации и параметры поставки, комплектацию, детали дизайна — цвет корпуса, салона и т. д. Так вот, AJAX очень сильно поможет вам удержать такого потенциального покупателя от бегства на другой сайт. Ведь покупатель не захочет ждать на каждом шаге долгого отклика сервера, ему нужно, чтобы на его действия следовала мгновенная (или быстрая) реакция приложения. Что тут может сделать AJAX? Да все! Пока ваш покупатель разглядывает каталог автомобилей данного производителя, раздумывает над вариантами выбора, указывает нужные параметры покупки, вы с помощью AJAX загружаете на его компьютер всю информацию об автомобилях данного типа. И когда покупатель решится на выбор модели, у вас на клиентском компьютере уже все будет готово для быстрого ответа.

Примеров можно привести множество. Мы только упомянем, что технологию AJAX используют крупнейшие поисковики (Yahoo, Google), известные интернет-магазины и многие сервисы Интернета.

На чем же основана технология AJAX? Опять-таки, согласно названию, это языки JavaScript и XML. Язык JavaScript мы обсуждали в гл. 3 этой книги, а с XML познакомимся в следующем разделе.

Знакомство с XML

Аббревиатура XML означает eXtensible Markup Language, что переводится как «расширяемый язык разметки». Более развернутое значение аббревиатуры — это расширяемая спецификация языка, предназначенного для создания страниц WWW. В чем состоит назначение языка XML? Язык HTML, который мы обсуждали в гл. 1, представляет собой некий необязательный и нестрого поддерживаемый свод правил оформления гипертекстовых документов. Множество интернет-браузеров интерпретируют его теги и синтаксические конструкции на свой лад, вводят в язык свои дополнения и расширения и даже нарушают его требования. Плюс к тому появилось множество устройств, работающих с Сетью, например, смартфоны, коммуникаторы, для которых требуются средства, не включенные в классический HTML.

Все это было учтено организацией W3C при определении пути развития языка HTML, которое преодолело бы указанные недостатки. Указанный путь состоит в следующем: вместо жесткого ограничения и пресечения различных вольностей в трактовке языка HTML или создания универсального, всеобъемлющего языка гипертекстовой разметки было решено создать метаязык, предназначенный для создания собственного языка гипертекстовой разметки или его модификации. В результате был создан язык XML.

Язык XML содержит средства, позволяющие создавать теги для данного конкретного языка разметки, определять их структуру, вложенность, набор атрибутов и их значений. Соответствующие конструкции XML реализуют так называемые правила DTD (Document Type Definition — определение типа документа), которым должен удовлетворять XML-документ, подчиняющийся созданному языку. А чтобы браузер или любая другая программа могла обрабатывать XML-документы, для каждого из них указывается соответствующий набор правил DTD.

Рассмотрим, как это выглядит на практике. Ниже приведен пример документа XML, который, несмотря на простоту, содержит все необходимые компоненты XML.

```
<?XML version="1.0".?>
<hello>
<title>Это документ XML</title>
<body>Наш первый XML-документ</body>
</hello>
```

Первый тег содержит указание браузеру, что этот документ подготовлен согласно языку XML версии 1.0. Знак ? после угловой скобки сообщает браузеру, что далее

следуют инструкции по обработке документа, которыми браузер должен руководствоваться. Эти инструкции завершаются знаком вопроса (?) перед угловой скобкой, то есть все инструкции браузеру помещаются между записями <? и ?>.

Саму гипертекстовую разметку задают дескрипторами, аналогично языку HTML. Дескрипторы могут быть одиночными и парными, вкладываться друг в друга и обладать набором атрибутов. Все это определяется набором правил DDT документа, подготовленного средствами XML. В качестве примера напишем правила DTD для вышеприведенного документа XML:

```
<!ELEMENT HELLO (TITLE, BODY)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
```

Эти правила указывают, что в нашем документе XML могут использоваться дескрипторы HELLO, TITLE и BODY. При этом TITLE и BODY обязательно должны лежать внутри HELLO и содержать символьные данные #PCDATA.

Для связи нашего документа XML с соответствующим набором правил DTD понадобится ключевое слово DOCTYPE. Рассмотрим, как это делается в случае использования внешнего, то есть не включенного в сам документ, набора правил DTD.

Вначале сохраним наш набор правил DTD в файле hello.dtd. Тогда документ XML будет следующим:

```
<?XML version="1.0"?>
<!DOCTYPE HELLO SYSTEM "hello.dtd">
<hello>
<title>Это документ XML</title>
<body>Наш первый XML-документ</body>
</hello>
```

Здесь дескриптор <!DOCKTYPE содержит ссылку на файл правил DTD hello.dtd, а ключевое слово SYSTEM означает, что этот набор является системным, то есть недоступным для общего использования (в противном случае применяется ключевое слово PUBLIC).

Набор правил DTD можно просто включить в документ XML, написав его следующим образом:

```
<?XML version=1.0"?>
<!DOCTYPE HELLO [
<!ELEMENT HELLO (TITLE, BODY)>
```

```
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
]>
<hello>
<title>Это документ XML</title>
<body>Наш первый XML-документ</body>
</hello>
```

Теперь браузер или другая программа будет знать, что ей делать при обработке XML-документа. Все, что ей нужно будет сделать, — найти корневой элемент (в нашем случае — HELLO) и выстроить иерархическую структуру из вложенных дескрипторов. В нашем примере все потомки корневого дескриптора ограничиваются двумя тегами TITLE и BODY, но более сложные конструкции DTD позволяют определять иерархическую структуру любой глубины. Это очень удобно при использовании структурированных данных, передаваемых для обработки в различные программы, не обязательно связанные с отображением ресурсов Интернета. Поэтому язык XML получил широкое распространение в других сферах, в частности его используют в технологии AJAX для обмена данными между сервером и клиентом в асинхронном режиме.

Перейдем теперь к рассмотрению техники программирования веб-приложений средствами AJAX.

Первое веб-приложение с использованием AJAX

Создадим и сохраним в папке examples файл ajax.html следующего документа HTML (листинг 7.1).

Листинг 7.1. Веб-страница с запросами AJAX

```
<html>
  <head>
    <title>Пример AJAX</title>
    <script type="text/javascript" src="ajax.js"></script>
  </head>
  <body onload='request()'>
    Enter your login:
    <input type="text" id="myLogin" />
    <div id="Message" />
  </body>
</html>
```

Как видим, он содержит дескриптор загрузки сценария JavaScript, сохраненного в файле `ajax.js` (листинг 7.2). В свою очередь, сценарий содержит операторы функции JavaScript для инициации запросов AJAX и их исполнения.

Листинг 7.2. Сценарий инициации и обработки запросов AJAX

```
if(window.ActiveXObject)
   var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
else
   var xmlHttp = new XMLHttpRequest();
if (!xmlHttp)
   alert("Error creating the XMLHttpRequest object.");
function request() {
   if (xmlHttp.readyState == 4 || xmlHttp.status == 0)  {
     name = document.all("myLogin").value;
     xmlHttp.open("GET", "ajax.php?login=" + name, true);
     xmlHttp.onreadystatechange = ResponseHandler;
     xmlHttp.send(null);
    }
   else
     setTimeout('request()', 1000);
 }
function ResponseHandler() {
    if (xmlHttp.readyState == 4)  {
       if (xmlHttp.status == 200)     {
           xmlResponse = xmlHttp.responseXML;
           xmlDocumentElement = xmlResponse.documentElement;
           Response = xmlDocumentElement.firstChild.data;
           document.all("Message").innerHTML =
                                                '<i>' + Response + '</i>';
           setTimeout('request()', 1000);
         }
       else  {
         alert("Ошибка доступа к серверу");
   } } }
```

Сохраним файл этого сценария в папке examples под именем `ajax.js`. Далее создадим сценарий PHP и сохраним его в папку examples под именем `ajax.php` (листинг 7.3).

Листинг 7.3. Серверная часть приложения

```
<?php
header('Content-Type: text/xml');
echo '<?xml version="1.0" ?>';
echo ,<response>';
$login = $_GET[,login'];
$userNames = array(,PETYA', ,SERGEY', ,IVAN', ,MASHA', ,LENA');
if (in_array(strtoupper($login), $userNames))
  echo ,Hello, user , . $login . ,!';
else if (trim($login) == ,')
  echo ,Empty login';
else
  echo $login, ,,  - unregistered user';
echo ,</response>';
?>
```

Откроем документ `ajax.html`, введя `localhost/examples/ajax.html` в адресную строку браузера. В окне браузера отобразится поле `Enter your login` для ввода входного имени пользователя. Начнем вводить в него какой-нибудь текст и заметим, как параллельно с вводом в строке под полем отображается сообщение сервера о результатах обработки введенных данных (рис. 7.1).

Обратите внимание, что вам не требуется нажимать кнопку подтверждения формы, как мы это делали в примерах предыдущей главы. Сервер сам, независимо от ваших действий, выполняет считывание введенных данных и их сравнение со списком известных ему имен пользователей. Если введенный текст не совпадает с каким-либо известным серверу именем, отображается сообщение `Unregistered user` (Незарегистрированный пользователь). Но если ввести имя, известное серверу, то отобразится приветствие (рис. 7.2).

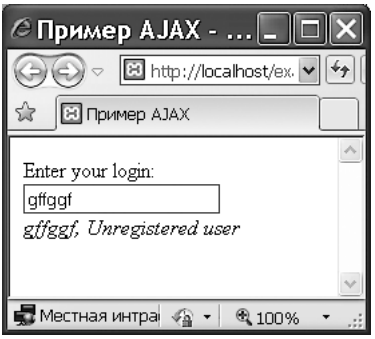


Рис. 7.1. Вводимые в поле данные обрабатываются сервером по запросам AJAX

Теперь обсудим, как работает наше веб-приложение. При загрузке страницы вместе с кодом HTML из листинга 7.1 загружается код программы JavaScript, представленный в листинге 7.2. При этом согласно правилам обработки кода HTML-документа будет выполнена часть программы JavaScript, стоящая в начале листинга 7.2:

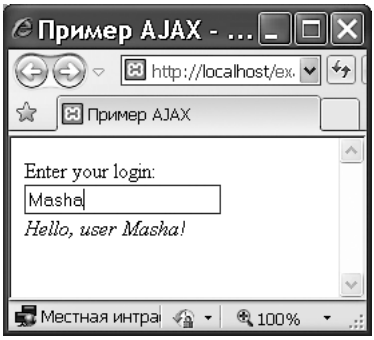


Рис. 7.2. Введенное имя распознано сервером

```
if(window.ActiveXObject)
   var xmlHttp = new ActiveXObject("Microsoft.
XMLHTTP");
else
   var xmlHttp = new XMLHttpRequest();
if (!xmlHttp)
   alert("Error creating the XMLHttpRequest object.");
```

Здесь выполняется создание объекта `XMLHttpRequest`, отвечающего за выполнение запросов AJAX. В зависимости от того, какой браузер используется, объект создается вызовом функции либо `ActiveObject()`, либо `XMLHttpRequest()`. При успешном создании объекта `XMLHttpRequest` переменной `xmlHttp` присваивается значение ссылки на объект, в случае неудачи создания объекта генерируется сообщение.

Теперь обратимся к строке кода HTML из листинга 7.1:

```
<body onload='request()'>
```

Он означает, что при загрузке страницы вызывается функция `request()` из сценария `ajax.js`:

```
function request() {
   if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)  {
     name = document.all("myLogin").value;
     xmlHttp.open("GET", "ajax.php?login=" + name, true);
     xmlHttp.onreadystatechange = ResponseHandler;
     xmlHttp.send(null);
    }
   else
     setTimeout('request()', 1000);
 }
```

Работа указанной функции состоит из двух частей: подготовка запросов AJAX к исполнению и запуск механизма таких запросов. Вначале проверяется состояние готовности объекта `XMLHttpRequest`. Это делается с помощью параметра объекта `readyState`, который может иметь следующие значения:

- `0` — запрос не инициализирован;
- `1` — идет отправка запроса;
- `2` — запрос отправлен;
- `3` — происходит обмен данными;
- `4` — запрос завершен.

Итак, мы проверяем, был запрос завершен или не инициализирован, и в любом из этих случаев начинаем его исполнение. В противном случае с помощью функции `setTimeout()` устанавливается задержка повторного вызова функции `request()`:

```
setTimeout('request()', 1000);
```

На первом этапе подготовки запросов AJAX с помощью коллекции `all` мы определяем значение элемента `MyLogin` в нашем документе HTML, то есть поля, содержащего введенный пользователем текст:

```
name = document.all("myLogin").value;
```

Далее с помощью метода `open` объекта `XMLHttpRequest` мы вызываем исполнение серверного сценария PHP:

```
xmlHttp.open("GET", "ajax.php?login=" + name, true);
```

Как видно из строки вызова сценария

```
"ajax.php?login=" + name
```

ему передается текст, введенный пользователем в поле нашего документа HTML. Кроме того, мы задаем асинхронный режим обмена информации с сервером, поскольку второй параметр метода равен `true`.

После этого запускается механизм AJAX асинхронных запросов: клиент будет автоматически, без всякого вмешательства пользователя, посылать запросы серверу и в случае получения отклика передавать полученные данные на обработку специальной функцией. Эта функция назначается следующим оператором:

```
xmlHttp.onreadystatechange = ResponseHandler;
```

Здесь мы просто присвоили параметру `onreadystatechange` объекта `XMLHttpRequest` ссылку на функцию `ResponseHandler()`.

Теперь все готово для запросов AJAX. Вызываем метод `send()` объекта `XMLHttpRequest`:

```
xmlHttp.send(null);
```

и отправляем запрос на сервер.

На стороне сервера исполняется очень простой сценарий PHP (см. листинг 7.3). Вначале мы отправляем в ответ на запрос заголовок XML-сообщения:

```
header('Content-Type: text/xml');
echo '<?xml version="1.0" ?>';
```

Первая строка содержит важный элемент заголовка, указывающего, что мы передаем данные в формате `xml`. Далее мы отправляем инструкцию браузеру, указывающую на используемую в сообщении версию языка XML, и первый тег этого сообщения:

```
echo '<response>';
```

Остальная часть кода выполняет сборку текста ответного сообщения в формате `xml`. Чтобы посмотреть, что получается в результате, мы вызовем наш сценарий PHP вводом в адресную строку браузера значения `http://localhost/ajax.php?login=`

Ответное сообщение XML представлено на рис. 7.3.

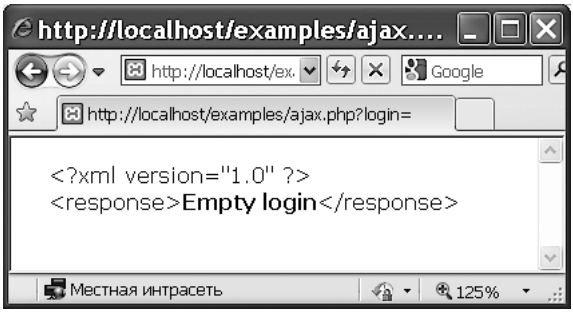


Рис. 7.3. Ответное сообщение XML нашего сервера

Формирование этого сообщения начинается с извлечения из строки вызова сценария переданного ему параметра

```
$login = $_GET['login'];
```

Далее мы просто сравниваем его со значениями в массиве `$userNames`, используя для этого встроенную функцию PHP `in_array()`, и, если совпадение имеется, передаем сообщение

```
echo 'Hello, user ' . $login . '!';
```

В противном случае мы проверяем, не пустая ли строка была передана серверу. Это не лишняя проверка, поскольку именно с такого случая начинаются запросы к серверу, когда пользователь еще не успел ввести никаких данных. Чтобы проверка была корректной, вначале с помощью функции `trim()` из переданной переменной `trim($login)` удаляются все пробелы, которые могут находиться в начале и конце строки. Если переданный параметр пуст, отображается сообщение о пустом поле (рис. 7.4).

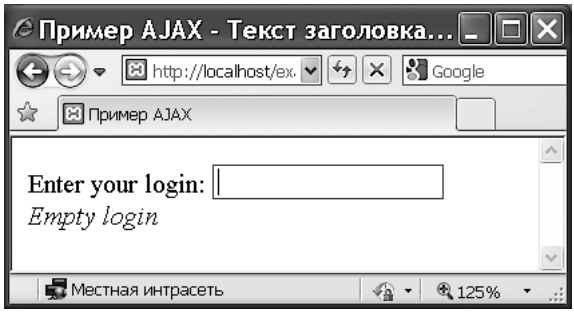


Рис. 7.4. Поле для ввода имени пустое

Если вы введете имя, известное серверу, то отобразится приветствие (рис. 7.5).

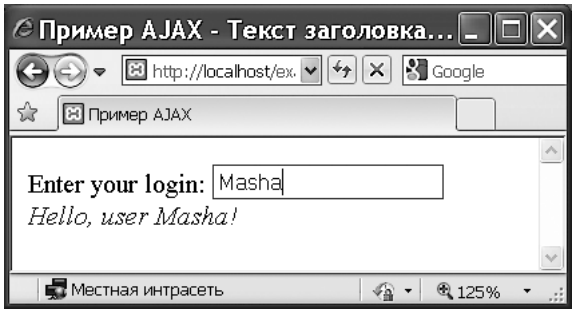


Рис. 7.5. Сервер узнал введенное имя

Но сервер передает свои сообщения в непонятном браузеру формате `xml`, как же такие сообщения отображаются на экране браузера? Ответ прост — это делает программа — обработчик прерываний:

```
function ResponseHandler() {
    if (xmlHttp.readyState == 4)  {
       if (xmlHttp.status == 200)     {
           xmlResponse = xmlHttp.responseXML;
           xmlDocumentElement = xmlResponse.documentElement;
           Response = xmlDocumentElement.firstChild.data;
           document.all("Message").innerHTML='<i>'+Response+'</i>';
           setTimeout('request()', 1000);
         }
       else  {
         alert("Ошибка доступа к серверу");
  } } }
```

Данный сценарий автоматически вызывается клиентом при каждом изменении в статусе отправленного запроса AJAX. Поле вызова на первом шаге проверяется состояние запроса согласно значению уже упомянутого параметра `readyState` объекта `XMLHttpRequest`:

```
if (xmlHttp.readyState == 4)
```

Если запрос успешно завершен, проверяется другой параметр `status` объекта `XMLHttpRequest`:

```
if (xmlHttp.status == 200)
```

Значение `200` сообщает об успешном завершении запроса. Если запрос успешно завершен, сценарий приступает к разборке полученного сообщения XML для извлечения сообщения, переданного сервером. Для этого используются параметры объекта `XMLHttpRequest`:

- `xmlHttp.responseXML` — извлекает полученное сообщение XML;
- `xmlDocumentElement = xmlResponse.documentElement` — извлекает корневой элемент в полученном сообщении XML;
- `xmlDocumentElement.firstChild.data` — извлекает содержимое корневого элемента XML.

В объекте `XMLHttpRequest` содержатся и другие средства для работы с запросами XML, но нам для нашего простого примера достаточно вышеперечисленных.

Извлеченное сообщение сервера присваивается переменной `Response`, с помощью которой составляется ответ серверу:

```
'<i>'+Response+'</i>'
```

Наконец, результат всех этих действий отображается на экране браузера в элементе `<div id="Message" />`.

Теперь, детально разобрав код нашего веб-приложения, повторим шаги его работы по выполнению запросов AJAX.

1. Посетитель сайта загружает страницу `ajax.html` с запросом AJAX и начинает ввод данных.
2. Браузер запускает программу `request()` для генерации запроса AJAX.
3. Программа `request()` проверяет готовность объекта `XMLHttpRequest` к отправке запросов.
4. Если объект `XMLHttpRequest` не готов, назначается интервал ожидания в 1 секунду и выполняется повторный вызов `request()`.
5. Если механизм AJAX готов к работе, `request()` готовит запрос AJAX:
   - извлекает содержимое поля ввода и помещает его в строку вызова серверного сценария;
   - назначает программу `ResponseHandler()` обработчиком запроса AJAX.
6. Программа `request()` запускает механизм асинхронных запросов серверу.
7. Серверный сценарий PHP обрабатывает запросы и генерирует ответные сообщения XML.
8. Клиент при любом изменении состояния переданного запроса вызывает обработчика `ResponseHandler()`.

   Если запрос выполнен успешно, `ResponseHandler()` обрабатывает сообщение XML и отображает ответ сервера.
   - Если поле ввода пустое, в ответ отобразится сообщение о пустом логине (см. рис. 7.4).
   - Если поле ввода содержит неизвестное имя, отображается сообщение о незарегистрированном пользователе (см. рис. 7.3).
   - Если введенное имя совпадает с известным серверу, отображается приветствие (см. рис. 7.5).
9. Обработчик назначает задержку исполнения программы `request()` в 1 секунду и вызывает ее по истечении срока. Все повторяется.

Таким образом, мы сумели с помощью объекта `XMLHttpRequest` передать запрос серверу, получить ответ и обработать его. Написанное нами веб-приложение реализует общепринятую последовательность действий, состоящую в создании объек-

та XMLHttpRequest и асинхронного запроса, назначении функции автоматической обработки этого запроса и повторном создании асинхронного запроса. Самое главное преимущество технологии AJAX состоит в том, что выполнение асинхронных запросов никак не влияет на работу пользователя. Вы оцените указанное преимущество, когда будете работать с большими объемами информации.

Мы использовали для обмена данными между сервером и клиентом сообщения в формате xml. Это вовсе не обязательное требование к приложениям, создаваемым на основе AJAX, — мы можем использовать любые структуры данных, в зависимости от решаемой задачи. Но применение XML имеет множество преимуществ, поскольку позволяет, во-первых, структурировать данные с помощью тегов и, во-вторых, далее работать с такими данными, используя множество методов, предоставляемых AJAX. Покажем это на примере.

Работа с данными XML

Напишем новое приложение, которое будет обрабатывать данные, сохраняемые в файле в формате xml. Вначале создадим сам файл данных (листинг 7.4).

Листинг 7.4. Данные о пользователях в формате XML

```
<?xml version="1.0" ?>
<response>
  <users>
    <user>
      <name>
        Petia
      </name>
      <age>
        25
      </age>
    </user>
    <user>
      <name>
        Vasia
      </name>
      <age>
        20
      </age>
```

```
    </user>
  </users>
</response>
```

Назовем **файл именем** `users.xml` **и сохраним его в папку** example. Как видим, он содержит имена пользователей и их возраст в тегах `<user>...</user>`, `<name>...</name>` и `<age>...</age>`.

Теперь создадим **HTML-файл страницы обращения к серверу (листинг 7.5).**

Листинг 7.5. Страница запроса сервера

```
<html>
  <head>
    <title>Пример работы со структурой XML</title>
    <script type="text/javascript" src="users.js"></script>
  </head>
  <body onload="request()">
    Сервер, кто твои пользователи?
    <br/>
    <div id="DivOutput" />
  </body>
</html>
```

Здесь кроме вопроса к серверу содержится элемент `<div id="DivOutput" />`, в который мы будем помещать ответы сервера так, как мы это делали в предыдущем примере. Сохраним приведенный код в файле `users.html` в папке examples.

Наконец, создадим сценарий **JavaScript для выполнения запросов к серверу (листинг 7.6).**

Листинг 7.6. Сценарий запросов серверу

```
if(window.ActiveXObject)
   var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
else
   var xmlHttp = new XMLHttpRequest();
function request() {
  if (xmlHttp)
  {
    // Пытаемся подключиться к серверу
    try
```

```
     {
        xmlHttp.open("GET", "http://localhost/examples/users.xml", true);
        xmlHttp.onreadystatechange = ResponseHandler;
        xmlHttp.send(null);
      }
    // Сообщение об ошибке подключения в случае неудачи
    catch (e)
     {
         alert("Ошибка подключения к серверу");
     }
  }
}
// Функция-обработчик запросов AJAX
function ResponseHandler()   {
 if (xmlHttp.readyState == 4)
 {
    if (xmlHttp.status == 200)
    {
       var xmlResponse = xmlHttp.responseXML;
       xmlRoot = xmlResponse.documentElement;
       nameArray = xmlRoot.getElementsByTagName("name");
       ageArray = xmlRoot.getElementsByTagName("age");
       var html = "";
       for (var i=0; i<nameArray.length; i++)
          html += "<b> Пользователь: "+ nameArray.item(i).firstChild.data +
                ", Возраст: " + ageArray.item(i).firstChild.data + "</b><br/>";
       myDiv = document.all("DivOutput");
       myDiv.innerHTML = "<i>Сервер ответил - вот они: </i><br />" + html;
      }
      else
      {
         alert("Ошибка доступа к серверу");
      }
   }
}
```

Сохраним его в файле `users.js` в папке examples.

Загрузим наш документ HTML в окно браузера, введя `localhost/examples/users.html` в его адресную строку. В результате сервер выдаст ответ, представленный на рис. 7.6.

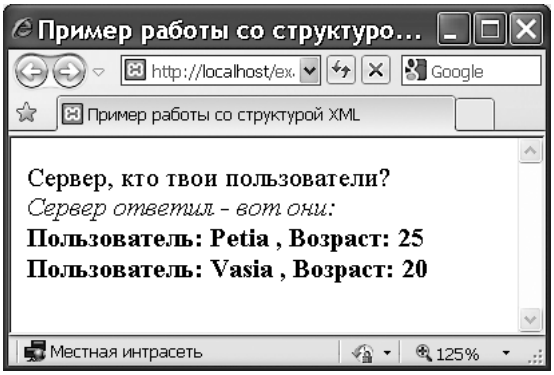


Рис. 7.6. Список пользователей сервера

Рассмотрим, как работает код сценария `users.js`. Вначале создается объект `XMLHttpRequest` с использованием методов, описанных в предыдущем примере. Так же как и в предыдущем примере, при загрузке страницы вызывается функция `request()`, однако она работает несколько иначе.

Сперва выполняется проверка успешности создания объекта `XMLHttpRequest`, по результатам которой мы применяем механизм обработки исключительных состояний `try/catch`, предусмотренный в JavaScript. Суть его состоит в перехвате возникших ошибок (то есть исключительных состояний) и их обработке. Синтаксис конструкции `try/catch` имеет следующий вид:

```
try
{
  код, способный породить ошибку
}
catch
{
 код, исполняемый при возникновении ошибки
}
```

Такая конструкция позволяет нам контролировать ход выполнения запросов. При возникновении ошибок, вполне вероятных при работе с сервером, мы сможем перехватить исключительное состояние и отобразить пользователю понятный и дружественный ответ.

В остальном программа `request()` действует аналогично предыдущему примеру: вначале готовится запрос AJAX, затем он отправляется на сервер, которому передается имя файла с данными XML:

```
xmlHttp.open("GET", "users.xml", true);
```

После этого назначается функция, вызываемая при всяком изменении состояния запроса `ResponseHandler()`. Данная функция представляет для нас особый интерес, поскольку именно она занята обработкой полученных данных XML.

Рассмотрим код, выполняющий эту задачу. Вначале выполняется обычная проверка успешного завершения запроса, после чего из запроса извлекаются переданные данные XML:

```
var xmlResponse = xmlHttp.responseXML;
```

Указанные данные сохраняются в переменной `xmlResponse`, и затем начинается их разборка. В первую очередь извлекаем корневой элемент XML-структуры:

```
xmlRoot = xmlResponse.documentElement;
```

Далее с помощью методов объекта `XMLHttpRequest` извлекаем и сохраняем в массивах имена и возраст пользователей:

```
nameArray = xmlRoot.getElementsByTagName("name");
ageArray = xmlRoot.getElementsByTagName("age");
```

После этого нам остается только собрать ответное сообщение, которое помещается в содержимое элемента `<div>` конкатенацией строк из содержимого массивов с данными по имени и возрасту.

Как видим, в приведенном примере вовсе не используются какие-либо серверные сценарии PHP: мы просто получаем от сервера файл данных и далее обрабатываем их клиентом.

Однако хранение данных на сервере в виде файла не является оптимальным решением. Более удобное средство — использование баз данных, в частности MySQL. Посмотрим, как в этом случае может быть применена технология AJAX.

Работа с MySQL

Сейчас мы продемонстрируем работу с сервером MySQL средствами, включающими в себя асинхронные запросы к серверному сценарию PHP, который будет извлекать

нужную информацию из базы данных MySQL и передавать ее клиенту с помощью сообщений XML. Для простоты мы используем наше самое первое веб-приложение с использованием AJAX, описанное выше в подразд. «Первое веб-приложение с использованием AJAX». Мы оставим без изменений коды в листингах 7.1 и 7.2, которые реализуют клиентскую часть приложения, и перепишем сценарий PHP, снабдив его средствами работы с MySQL.

Для начала создадим базу данных MySQL с таблицей, содержащей имена пользователей, которые в сценарии в листинге 7.3 хранились в массиве `$UserNames`: `'PETYA'`, `'SERGEY'`, `'IVAN'`, `'MASHA'`, `'LENA'`. Это можно сделать запросами MySQL или из панели администрирования сервером MySQL методами, описанными в гл. 3. Назовем созданную базу данных users, а таблицу — UsrLogin.

Теперь подготовим код нашего веб-приложения. Вначале подготавливаем страницу ввода входного имени пользователя (листинг 7.7).

Листинг 7.7. Страница ввода логина

```
<html>
  <head>
    <title>Пример AJAX</title>
    <script type="text/javascript" src="mysql.js"></script>
  </head>
  <body onload='request()'>
    Enter your login:
    <input type="text" id="myLogin" />
    <div id="Message" />
  </body>
</html>
```

Сохраним его в файле `mysql.html` в папке examples. Теперь напишем сценарий JavaScript для организации запросов MySQL. Он отличается от предложенного в листинге 7.2 только использованием механизма перехвата исключительных состояний (листинг 7.8).

Листинг 7.8. Сценарий отправки и обработки асинхронных запросов

```
if(window.ActiveXObject)
   var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
else
   var xmlHttp = new XMLHttpRequest();
if (!xmlHttp)
```

```
   alert("Error creating the XMLHttpRequest object.");
function request() {
  if (xmlHttp)
  {
    // Пытаемся подключиться к серверу
    try
     {
        name = document.all("myLogin").value;
        xmlHttp.open("GET", "mysql.php?login=" + name, true);
        xmlHttp.onreadystatechange = ResponseHandler;
        xmlHttp.send(null);
      }
    // Сообщение об ошибке подключения в случае неудачи
    catch (e)
     {
        alert("Ошибка подключения к серверу");
     }
  }
}
function ResponseHandler() {
    if (xmlHttp.readyState == 4)  {
       if (xmlHttp.status == 200)     {
           xmlResponse = xmlHttp.responseXML;
           xmlDocumentElement = xmlResponse.documentElement;
           Response = xmlDocumentElement.firstChild.data;
           document.all("Message").innerHTML =
                                                  '<i>' + Response + '</
i>';
           setTimeout('request()', 1000);
         }
       else  {
         alert("Ошибка доступа к серверу");
        }
     }
  }
```

Здесь, как вы можете заметить, используется конструкция `try/catch` перехвата ошибок и их обработки. Сохраним этот сценарий в файле `mysql.js` в папке example.

Теперь напишем сценарий PHP работы с базой данных MySQL (листинг 7.9).

Листинг 7.9. Серверный сценарий выбора входных имен из MySQL

```
<?php
require_once('handle_err.php');
header('Content-Type: text/xml');
echo '<?xml version="1.0" ?>';
echo ,<response>';
$login = $_GET[,login']];
$connection=mysql_connect(„localhost", „user", „password");
$my_db=mysql_select_db(„users", $connection);
$query="SELECT * FROM UsrLogin";
$result=mysql_query($query);
$userNames=mysql_fetch_array($result);
if (in_array(strtoupper($login), $userNames))
  echo ,Hello, user , . htmlentities($login) . ,!';
else if (trim($login) == ,')
  echo ,Empty login';
else
  echo htmlentities($login) . ,, Unregistered user';
echo ,</response>';
?>
```

Сохраним этот сценарий в файле `mysql.php` в папке example и обратим внимание на ту деталь, что в начале сценария находится оператор

```
require_once('handle_err.php');
```

который выполняет однократную загрузку файла `handle_err.php` со сценарием функции обработки ошибок PHP (листинг 7.10). Этот файл мы также создадим и запишем в папку example.

Листинг 7.10. Функция обработки ошибок

```
<?php
set_error_handler('handle_err');
function handle_err($errNo, $errStr, $errFile, $errLine)
```

```
{
  if(ob_get_length()) ob_clean();
  $message = ‚Номер ошибки: ‘ . $errNo .
             ‘Сообщение: ‘ . $errStr .
             ‘Расположение: ‘ . $errFile .
             ‘, Строка ‘ . $errLine;
  echo $message;
  exit;
}
?>
```

Если сейчас загрузить в браузер страницу `mysql.html`, введя `localhost/examples/mysql.html` в его адресную строку, то на экране будут отображаться те же сообщения, что и на рис. 7.1 и 7.2. Однако теперь все данные о пользователях сервер извлекает из базы данных MySQL. Если потребуется, мы всегда можем дополнить наш сценарий инструментами пополнения этой базы данных, редактирования хранимых данных, добавления новых таблиц. Принцип построения веб-приложения, использующего технологию AJAX для обмена данными с сервером PHP и MySQL, остается неизменным. Конечно, в случае, когда мы просто извлекаем несколько имен пользователей из базы данных и отображаем их на экране, применение AJAX не так и обязательно. Но если при обращениях к базе данных вы намереваетесь извлекать информацию большого объема, причем уточнение характера получаемых данных вы будете делать вводом параметров в сложную форму, то применение AJAX — это единственный выход оптимизировать время ожидания отклика сервера.

Теперь рассмотрим, как работают наши сценарии PHP. В самом начале программы в листинге 7.9 мы подключаемся к серверу MySQL, после чего выбираем нашу базу данных `users` и извлекаем данные из таблицы `UsrLogin` с помощью запроса:

```
$query="SELECT * FROM UsrLogin";
```

Далее обработка полученной таблицы выполняется аналогично примеру сценария в листинге 7.3. Наш сценарий проверяет, содержится ли введенное имя в списке имен, хранимых на сервере, и, в зависимости от результата, генерирует ответ в формате сообщения XML.

Наибольший интерес представляет сценарий обработки исключительных состояний, представленный в листинге 7.10. В нем в самом начале с помощью функции

set_error_handler() определяется пользовательский обработчик прерываний:

```
set_error_handler('handle_err');
```

Здесь handle_err — название пользовательской функции. Она имеет следующие параметры: код ошибки ($errNo), строку с описанием ошибки ($errStr), имя файла, в котором появилась ошибка ($errFile), номер строчки ($errLine):

```
function handle_err($errNo, $errStr, $errFile, $errLine)
```

Вначале с помощью функции ob_clean() выполняется очистка буфера вывода, а затем генерируется сообщение, собираемое конкатенацией параметров нашей функции — обработчика ошибок. По завершении работа сценария прерывается оператором exit, поскольку теперь мы должны сами останавливать работу программы. Если этого не сделать, работа программы продолжится с оператора, следующего за вызвавшим сбой.

Имейте в виду, что такая функция не в состоянии предотвратить ошибки синтаксического анализа кода — такие ошибки прерывают работу интерпретатора еще до выполнения сценария. Однако пользовательские программы обработки ошибок чрезвычайно полезны, поскольку позволяют создавать осмысленные сообщения о проблемах в работе веб-приложения.

Резюме

Итак, мы познакомились с основами технологии AJAX, научились создавать асинхронные запросы, генерировать ответы сервера в виде сообщений XML и обрабатывать такие сообщения средствами разборки структуры XML. Далее мы освоили средства перехвата исключительных состояний с помощью конструкции try/catch для сценариев JavaScript и назначения пользовательских функций обработки ошибок сценариев PHP. К тому же мы показали, как веб-приложения, созданные с использованием AJAX, могут работать с базами данных MySQL, и проиллюстрировали это, создав простое приложение.

Возможности AJAX не ограничиваются описанными в этой главе — вне нашего внимания остались многие важные вопросы. Однако теперь вы готовы писать простые сценарии, использующие асинхронные запросы, что чрезвычайно важно во многих практических случаях, например, при создании интернет-магазинов, обеспечивающих выбор товара из огромных баз данных, или приложений, позволяющих просматривать географические карты наподобие Google Earth.

Глава 8

Проверка работоспособности сайта

Проверка актуальности ссылок на сайте

Оптимизация кода веб-страниц

Резюме

Итак, в предыдущих главах вы получили все знания, необходимые для создания сайта с интересным содержанием, прекрасной подачей материала и эффектным оформлением. Бесспорно, когда вы создадите сайт, у вас будет желание работать с ним, постоянно поддерживать его и обновлять. У вас, естественно, появится огромное желание как можно скорее выставить его на всеобщее обозрение и сделать все возможное, чтобы ресурс стал известным, популярным и посещаемым. Но не торопитесь: сайт нуждается в окончательной доводке и оптимизации.

Прежде чем выложить сайт в Интернете, нужно проверить его работоспособность, чтобы выявить ошибки, которые, кстати, встречаются довольно часто. Самые распространенные среди них — орфографические ошибки, неработающие ссылки, неточности HTML-кода и большой размер HTML-файлов, увеличивающий время загрузки страниц.

Орфографию удобнее всего проверить с помощью программы Microsoft Word, которая, как известно, позволяет открывать HTML-документы. Для проверки орфографии достаточно загрузить HTML-файл в рабочее окно программы и выполнить команду меню Сервис ▸ Правописание (Tools ▸ Spelling) или нажать клавишу F7.

Если сайт создан в русской и английской версиях, необходимо проверить соответствие версий и правильность перевода.

Как известно, разные программы просмотра веб-страниц могут по-своему отображать один и тот же веб-документ. Поэтому необходимо посмотреть, как выглядит сайт в наиболее популярных браузерах, таких как Internet Explorer, Mozilla Firefox и Opera. Просмотреть веб-страницы следует при разных размерах шрифта и различных разрешениях экрана: 800 × 600, 1024 × 768 и 1280 × 1024, так как большинство пользователей сети используют мониторы именно с этими разрешениями.

В результате проверки вы должны решить, какие изменения следует внести в веб-документ, чтобы он наилучшим образом отображался в разных браузерах при различных разрешениях экрана и размерах шрифта.

Проверка актуальности ссылок на сайте

Следующим шагом будет проверка работоспособности ссылок на веб-страницах. Такая проверка очень важна, так как сайт, ссылки которого никуда не ведут, сразу теряет доверие посетителей. Даже если вы уверены, что все ссылки работоспособны, все равно проверьте их еще раз. Особое внимание обратите на внешние ссылки, по которым посетитель переходит на другие сайты. Такая проверка

позволит вам убедиться, что веб-страницы, на которые вы ссылаетесь, существуют.

Как проверить сайт с большим количеством ссылок? Не делать же это вручную! Если сайт объемный и содержит несколько сот или тысяч ссылок, то для проверки их актуальности можно воспользоваться специальными программами, например Web Link Validator.

Web Link Validator — это программа, предназначенная для проверки актуальности и работоспособности ссылок. С ее помощью веб-мастер может найти неработоспособные ссылки, а также ссылки, содержащие синтаксические ошибки. Программа позволяет одновременно проверять несколько сайтов, причем в течение одного сеанса она может проанализировать свыше 100 000 ссылок. Результаты проверки могут быть сохранены для дальнейшего просмотра.

Рабочее окно программы Web Link Validator

После установки Web Link Validator значок программы помещается на рабочий стол, упрощая таким образом ее запуск. Под строкой меню окна Web Link Validator (рис. 8.1), как обычно, располагается панель Toolbar (Панель инструментов) с кнопками, ускоряющими выполнение основных операций.

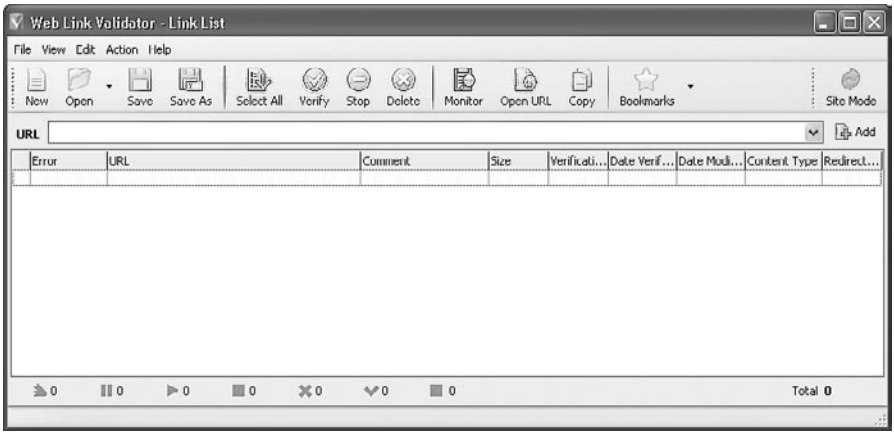


Рис. 8.1. Рабочее окно программы Web Link Validator

В верхней части окна находится раскрывающийся список URL, предназначенный для ввода URL-адреса ссылки, которую следует проверить. У нижнего края рабочего окна находится строка состояния, в которой отображаются состояние программы, ход выполнения операций, назначение команд меню и элементов управления.

Оставшаяся область рабочего окна программы, занимающая большую его часть, занята панелью списка ссылок, на которой будут представляться проверяемые ссылки с подробной информацией о них.

Программа Web Link Validator может работать в двух режимах.

- Link List (Список ссылок) — используется для проверки отдельных ссылок, если вы точно знаете путь к HTML-файлу или же если URL-адрес ссылки находится в буфере обмена. В этом режиме можно, например, проверить актуальность закладок, хранящихся в папке Избранное (Favorites).
- Web Site Verification (Проверка веб-сайта) — используется для проверки ссылок всего сайта.

Переключить режим работы программы Web Link Validator можно с помощью команды меню View (Вид), установив флажок рядом с нужным режимом в открывшемся меню.

Вы теперь знаете, как запустить программу, как выглядит ее окно и в каких режимах она может работать. В следующем подразделе мы рассмотрим проверку ссылок и закладок в режиме Link List (Список ссылок).

Проверка отдельных ссылок и закладок

Программа Web Link Validator проста в применении, в то же время она отлично справляется со своей задачей — проверкой работоспособности ссылок. Посмотрим, как в режиме Link List (Список ссылок) проверить работоспособность отдельных ссылок.

1. Выберите команду меню View ▸ Link List (Вид ▸ Список ссылок) или нажмите клавишу F5. Если включен другой режим, программа переключится в режим Link List (Список ссылок) (см. рис. 8.1).

   Для включения этого режима можно также нажать кнопку Switch to link verification mode (Переключение в режим проверки ссылок) на панели Toolbar (Панель инструментов). Здесь английское название кнопки — это всплывающая подсказка, которая появляется при наведении указателя мыши на данную кнопку.

2. В раскрывающемся списке URL введите путь к веб-документу, актуальность которого вы хотите проверить.

ВНИМАНИЕ

Адрес веб-папки обязательно должен заканчиваться косой чертой /.

3. Нажмите кнопку Add (Добавить), находящуюся справа от раскрывающегося списка URL. Введенный адрес поместится на панель ссылок (рис. 8.2).

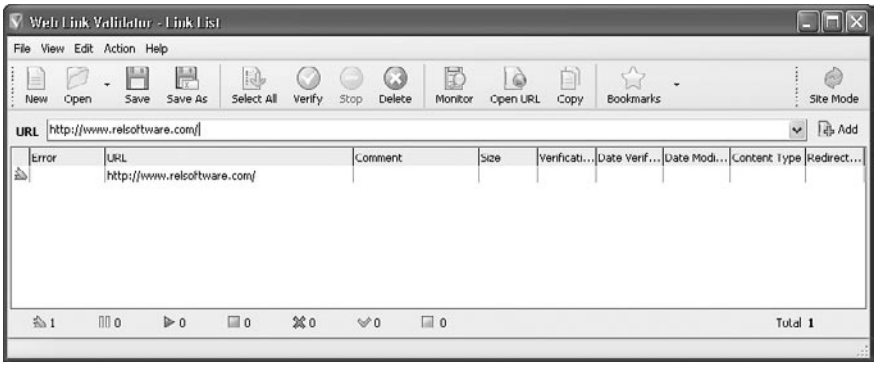


Рис. 8.2. Добавление ссылки

4. Таким же образом добавьте еще несколько ссылок.

   В текущем режиме Link List (Список ссылок) можно проверить работоспособность всех закладок из папки Избранное (Favorites) программы Internet Explorer или аналогичной папки другого браузера, так как все закладки являются ссылками.

5. Нажмите кнопку Bookmarks (Закладки) на панели Toolbar (Панель инструментов) и в открывшемся меню выберите команду All Favorite/Bookmarks (Все избранное/закладки) или нажмите комбинацию клавиш Shift+Ctrl+A. Все закладки, созданные в браузерах, импортируются в программу и отобразятся на панели списка ссылок.

   Каждая добавленная ссылка займет на панели списка ссылок отдельную строку. Общее количество добавленных ссылок указывается в нижней части панели справа параметром Total (Общее).



ПРИМЕЧАНИЕ

Если у вас установлено несколько браузеров для просмотра веб-страниц, то вы можете раздельно импортировать закладки каждой программы, воспользовавшись командами меню кнопки Bookmarks (Закладки) и выбрав далее Internet Explorer Favorites (Избранное Internet Explorer), Netscape Navigator/ Mozilla Firefox Bookmarks (Закладки Netscape Navigator/ Mozilla Firefox) или Opera Bookmarks (Закладки Opera).

   Теперь необходимо указать программе, какие из добавленных ссылок следует проверить.

6. Нажмите кнопку Select All (Выбрать все) на панели Toolbar (Панель инструментов). Все ссылки выделятся темным цветом.



ПРИМЕЧАНИЕ

Для выделения группы из отдельных ссылок, расположенных непоследовательно, щелкайте кнопкой мыши на ссылках группы, удерживая нажатой клавишу Ctrl. Чтобы удалить выделенные ссылки, нажмите кнопку Delete (Удалить) на панели Toolbar (Панель инструментов).

7. Нажмите кнопку Verify (Проверить) на панели Toolbar (Панель инструментов). Начнется проверка выбранных ссылок, ход которой будет отображаться и комментироваться в строке состояния.

Чтобы прервать проверку ссылок, достаточно нажать кнопку Stop (Прервать) на панели Toolbar (Панель инструментов).

В процессе проверки у левого края строки состояния выводятся три числа, разделенные символом /. Первое число означает общее количество ссылок, которое должно быть проверено, второе — количество ссылок, ожидающих проверку, и третье — количество проверенных ссылок.

Текущее состояние проверки каждой ссылки и результаты проверки обозначаются в первой слева колонке панели специальными значками.

Когда проверка завершится, на панели списка ссылок вы увидите ее результаты, представленные в виде таблицы (рис. 8.3).

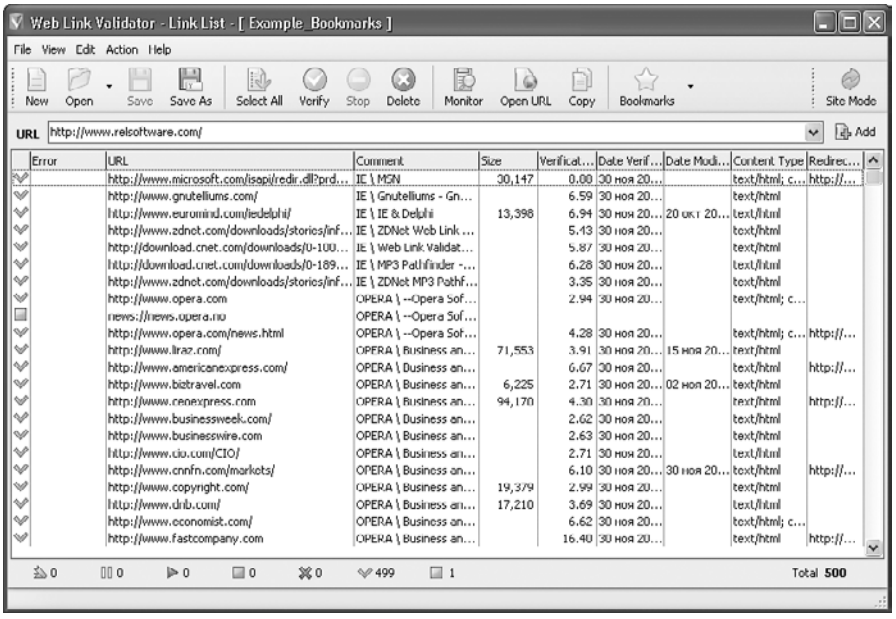


Рис. 8.3. Результат проверки ссылок

В левой колонке, не имеющей названия, значком обозначается текущее состояние ссылки. Далее следуют такие поля:

- Error (Ошибка);
- URL (URL-адрес);
- Comment (Комментарий);
- Size (Размер файла документа);
- Verification Time (Время проверки);
- Date Verified (Дата проверки);
- Date Modified (Дата изменения);
- Content Type (Тип содержимого);
- Redirected to (Переадресована).

Вы можете сортировать список ссылок по любому из полей. Для этого нужно щелкнуть кнопкой мыши на его заголовке. Повторный щелчок кнопкой мыши на этом же заголовке изменит порядок сортировки на обратный. С помощью кнопок, расположенных под таблицей и обозначенных значками, можно выделять ссылки, имеющие тот или иной статус.

Результат проверки можно сохранить в файле, нажав кнопку Save (Сохранить) на панели Toolbar (Панель инструментов). После исправления ошибочных ссылок сохраненный файл можно открыть, нажав кнопку Open (Открыть) на панели Toolbar (Панель инструментов), и, выделив ссылки с ошибками, повторно их проверить.

Мы рассмотрели работу с программой в режиме Link List (Список ссылок). Как видите, использовать программу несложно, а результат поиска ссылок исчерпывающий. В следующем подразделе вы узнаете, как применять программу Web Link Validator в режиме Web Site Verification (Проверка веб-сайта) для проверки работоспособности ссылок.

Проверка ссылок всего веб-сайта

Кроме проверки актуальности отдельных ссылок и закладок, программа Web Link Validator позволяет проверить работоспособность ссылок всего сайта. Это делается в режиме Web Site Verification (Проверка веб-сайта). Чтобы проверить все ссылки сайта, выполните следующие действия.

1. Выберите команду меню View ▸ Web Site Verification (Вид ▸ Проверка веб-сайта) или нажмите клавишу F6, чтобы переключиться в данный режим.

Для включения этого режима можно также нажать кнопку Switch to Web site verification mode (Переключение в режим проверки веб-сайта) на панели Toolbar (Панель инструментов).

В режиме Web Site Verification (Проверка веб-сайта) (рис. 8.4) под панелью Toolbar (Панель инструментов) слева располагается панель фильтров, а справа — панель списка ссылок. Под этими панелями находится поле URL, а еще ниже — панель свойств ссылок.

Теперь перейдем непосредственно к проверке ссылок сайта.

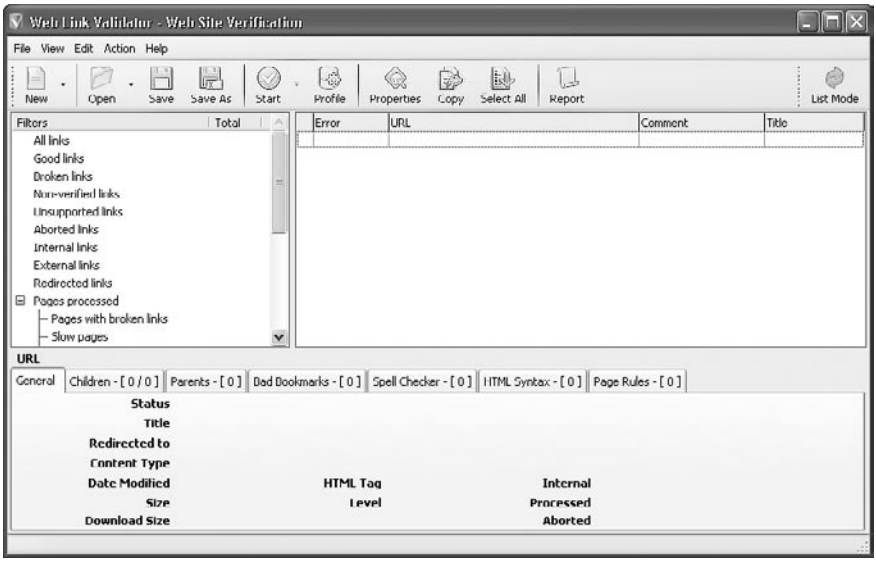


Рис. 8.4. Рабочее окно программы Web Link Validator в режиме Web Site Verification (Проверка веб-сайта)

2. Нажмите кнопку New (Новый) на панели Toolbar (Панель инструментов) и в открывшемся меню выберите команду Qiuck Start (Быстрый старт). Появится диалоговое окно Qiuck Start (Быстрый старт), в котором следует указать URL-адрес сайта (рис. 8.5).

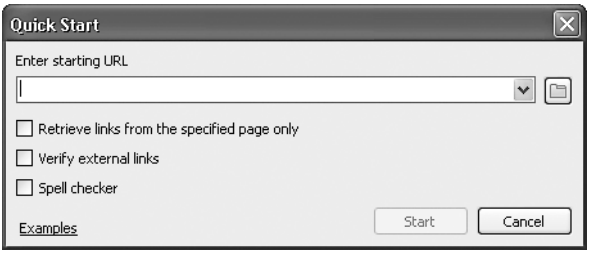


Рис. 8.5. Диалоговое окно Qiuck Start (Быстрый старт)

3. В поле Enter starting URL (Введите стартовый URL) введите путь к веб-документу, актуальность которого вы хотите проверить, например http://www.relsoftware.com/.
4. Снимите флажок Retrieve links from the specified page only (Поиск ссылок только на указанной странице). При этом работоспособность ссылок будет проверяться не только на заданной странице сайта, но и на всех связанных с ней страницах.



ПРИМЕЧАНИЕ

Если установить флажок Verify external links (Проверить внешние ссылки), то проверится актуальность всех ссылок, ведущих на другие сайты.

5. Нажмите кнопку Start (Начало), находящуюся в нижней части диалогового окна. Начнется проверка указанного адреса, за ходом которой можно следить в открывшемся диалоговом окне Web Site Verification Process (Процесс проверки веб-сайта) (рис. 8.6).

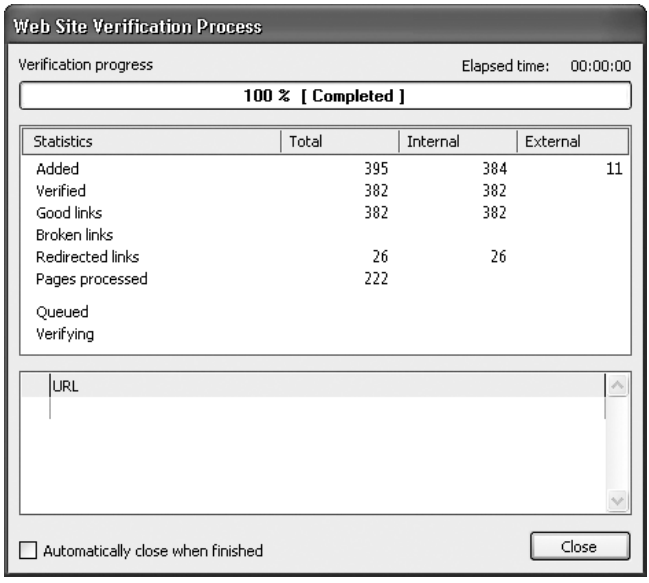


Рис. 8.6. Диалоговое окно Web Site Verification Process (Процесс проверки веб-сайта)

Ход процесса отображает зеленая полоса на линейном индикаторе Verification progress (Ход процесса).

Под этим индикатором приводится статистика, которая отображает количество ссылок:

- проверенных (Verified);
- проигнорированных (Ignored);

- доступных для просмотра и анализа после проверки (Added). Это те ссылки, которые были найдены и непроигнорированы.

В поле списка в нижней части диалогового окна отображаются URL-адреса обрабатываемых ссылок.

Если вы хотите прервать процесс проверки, нажмите кнопку Stop (Стоп).

6. Когда проверка ссылок завершится, закройте диалоговое окно Web Site Verification Process (Процесс проверки веб-сайта), нажав кнопку Close (Закрыть). На панели списка ссылок в рабочем окне программы отобразится таблица с результатами проверки (рис. 8.7).

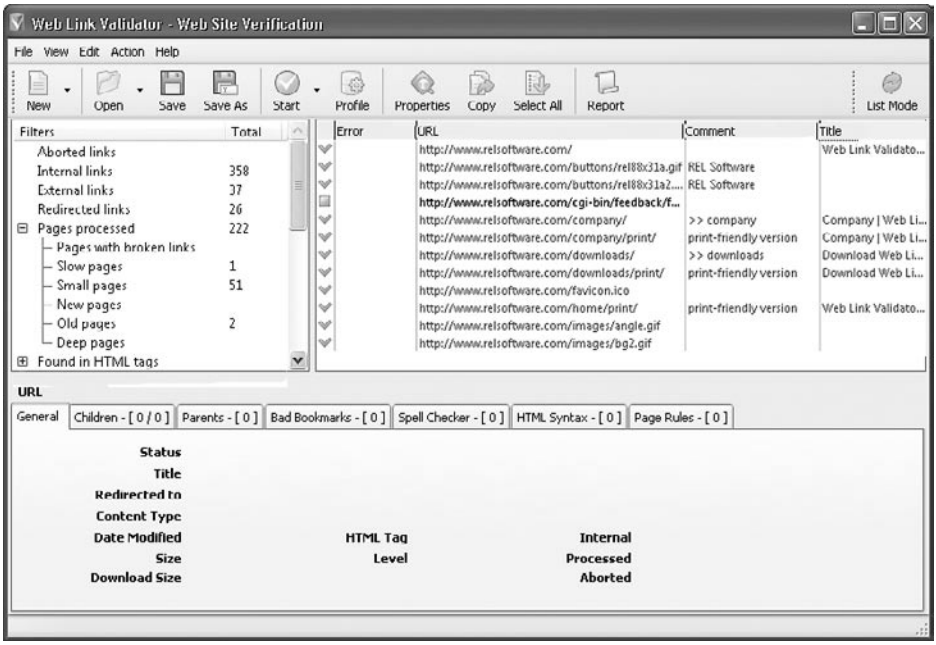


Рис. 8.7. Проверка ссылок сайта выполнена

В первой слева колонке этой таблицы, не имеющей названия, специальными значками показываются работающие, неработающие и непроверенные ссылки.

С помощью панели фильтров вы можете отобразить на панели списка только те ссылки, которые соответствуют определенным критериям:

- All links (Все ссылки);
- Good links (Работающие ссылки);
- Broken links (Неработающие ссылки);
- Non verified links (Непроверенные ссылки);

- Unsupported links (Неподдерживаемые ссылки);
- Aborted links (Прерванные ссылки);
- Internal links (Внутренние ссылки);
- External links (Внешние ссылки);
- Pages processed (Ссылки на просмотренных страницах);
- Redirected links (Переадресованные ссылки);
- Found in HTML Tags (Найденные в HTML-тегах);
- Protocols (Ссылки, использующие определенный протокол);
- Filename Extension (Расширение имени файла);
- File groups (Группы файлов);
- Directory tree (Дерево папок).

Чтобы произвести фильтрацию ссылок, достаточно щелкнуть кнопкой мыши на нужном фильтре.

Панель свойств ссылок состоит из следующих основных вкладок:

- General (Общие) — отображает общие свойства ссылки;
- Children (Дочерние) — представляет все адреса URL, найденные на веб-странице. Первое число, стоящее рядом с названием вкладки, означает общее количество дочерних ссылок, а второе — количество неработающих ссылок;
- Parents (Родительские) — содержит информацию о веб-страницах, имеющих ссылки на выбранный URL;
- Bad Bookmarks (Закладки) — отображает список закладок, найденных на странице.

Чтобы просмотреть свойства той или иной ссылки, следует щелчком кнопкой мыши выделить ее и нажать кнопку Properties (Свойства) на панели Toolbar (Панель инструментов).

На вкладке General (Общие) панели свойств ссылок отображается общая информация о выбранной в данный момент ссылке:

- Status (Состояние);
- Title (Заголовок);
- Redirected to (Переадресована);
- Content Type (Тип содержимого);
- Data Modified (Дата изменения);
- Verification time (Время проверки);

- Size (Размер);
- Download Size (Загружаемый размер);
- HTML Tag (HTML-тег);
- Level (Уровень);
- Internal (Внутренняя);
- Processed (Проверенная);
- Aborted (Прерванная).

Если на сайте обнаружены неработающие ссылки, то для их восстановления требуется определить, на каких именно веб-страницах они находятся. Для этого следует выделить такую ссылку на панели списка и нажать кнопку Properties (Свойства) на панели Toolbar (Панель инструментов), после чего открыть вкладку Parents (Родительские).

Полученные результаты проверки ссылок сайта можно сохранить в файле, нажав кнопку Save (Сохранить) на панели Toolbar (Панель инструментов).

После восстановления неработающих ссылок и решения прочих проблем, обнаруженных на вашем сайте, следует выполнить повторную проверку, открыв сохраненный файл, нажав кнопку Open (Открыть) на панели Toolbar (Панель инструментов), а затем нажав кнопку Start (Пуск).

Можно также проверить только некоторые ссылки. Для этого следует воспользоваться фильтром, после чего выделить нужные ссылки на панели списка и нажать кнопку Copy (Копировать). Затем следует переключиться в режим Link List (Список ссылок), выделить нужные ссылки и начать проверку, нажав кнопку Verify (Проверить).

Если веб-сайт содержит простые файлы HTML, не требующие доступа к веб-серверу, то вы можете проверить работоспособность ссылок на локальном диске. Для этого следует нажать кнопку New (Новый) на панели Toolbar (Панель инструментов) и в появившемся диалоговом окне Quick Start (Быстрый старт), нажав кнопку Open (Открыть), выбрать на локальном диске файл с начальной страницей сайта. Обычно это файл `index.html`.

Вы познакомились с программой Web Link Validator и научились проверять с ее помощью работоспособность ссылок сайта. На этом проверка сайта не заканчивается: необходимо соблюсти очень важное правило — проследить, чтобы размер созданной веб-страницы не был большим. Рекомендуется ограничивать размеры веб-страниц в пределах 40–50 Кбайт на все данные. В это число входят размер HTML-файла, все рисунки, баннеры, счетчики — в общем, все то, что есть на вашей

странице. Обычно добиться указанного ограничения размера страницы не составляет особого труда, так как это объем нормальной страницы, не перегруженной графикой и рекламой. Но что нужно сделать, чтобы максимально уменьшить объем страницы? Для этого необходима оптимизация кода веб-страниц. Об этом я и расскажу в следующем разделе.

Оптимизация кода веб-страниц

Чтобы уменьшить объем данных, сократив таким образом размер и время загрузки веб-страниц, их нужно оптимизировать. Эта задача включает оптимизацию HTML-кода и графики. Причем особое значение имеет оптимизация графики, так как именно она значительно влияет на объем веб-страниц. Однако не стоит пренебрегать оптимизацией HTML-кода, так как этим также можно несколько снизить размер документа. В этом разделе мы и поговорим об оптимизации HTML-файлов.

Для оптимизации HTML-кода веб-документов могут использоваться специальные программы, например Advanced HTML Optimizer.

Программа Advanced HTML Optimizer выполняет оптимизацию HTML-документов, в результате которой сокращается время их загрузки и выгрузки. После оптимизации внешний вид веб-страницы не изменяется, но размер ее уменьшается. Размер HTML-документа сокращается за счет удаления необязательных символов и тегов. Хотя процент сокращения размера файла после оптимизации может быть незначительным, время его загрузки может существенно измениться, так как браузер клиента анализирует оптимизированный код более эффективно. Вставленный в HTML-документ код сценария не нарушается.

Программа запускается из главного меню Windows или двойным щелчком кнопкой мыши на значке Advanced HTML Optimizer, который выносится на рабочий стол при установке программы.

В рабочем окне программы Advanced HTML Optimizer (рис. 8.8) под строкой меню располагается панель Toolbar (Панель инструментов), кнопки которой предоставляют быстрый доступ к основным операциям. У нижнего края рабочего окна находится строка Statusbar (Строка состояния), в которой отображается информация о работе программы. Остальная часть рабочего окна представляет собой таблицу File(s) to compress (Файл(ы) для оптимизации) для отображения информации об исходных и оптимизированных файлах.

Чтобы выполнить оптимизацию веб-документов, необходимо сначала открыть нужные файлы. Для этого выполните следующие действия.

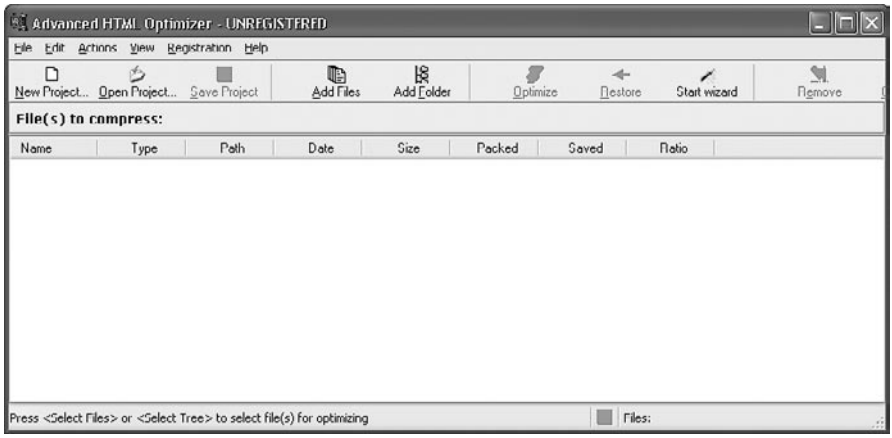


Рис. 8.8. Рабочее окно программы Advanced HTML Optimizer

1. Нажмите кнопку Add Files (Добавить файлы) на панели Toolbar (Панель инструментов). На экране появится диалоговое окно Add File(s) (Добавить файлы(ы)) (рис. 8.9).

   Это стандартное диалоговое окно открытия файла.

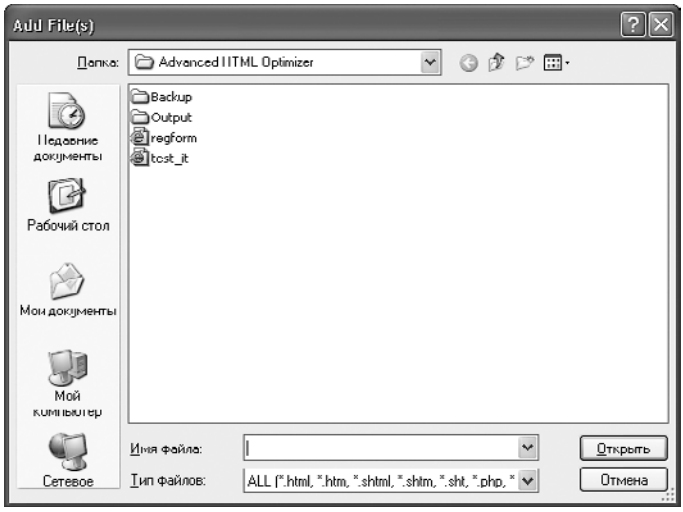


Рис. 8.9. Диалоговое окно Add File(s) (Добавить файлы(ы))

2. Выберите HTML-файл, который нужно оптимизировать, нажав кнопку Открыть (Open), а затем закройте диалоговое окно Add File(s) (Добавить файлы(ы)). В таблице File(s) to compress (Файл(ы) для оптимизации) появится информация о выбранном файле (рис. 8.10).

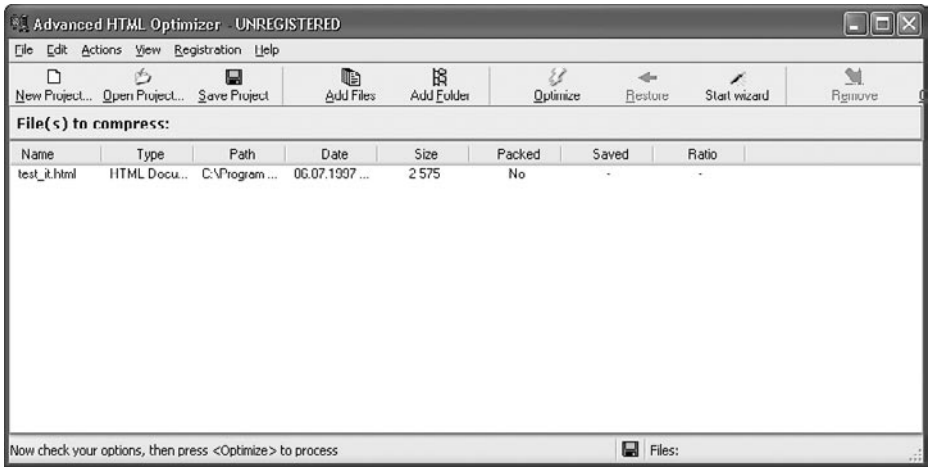


Рис. 8.10. Открытие файла для оптимизации

Информация о файле включает следующие поля:

- Name (Имя) — имя файла;
- Type (Тип) — тип файла;
- Path (Путь) — путь к файлу;
- Date (Дата) — дата последнего изменения;
- Size (Размер) — исходный размер файла;
- Packed (Сжато) — размер файла после оптимизации;
- Saved (Спрятано) — разница между исходным и оптимизированным размером;
- Ratio (Коэффициент) — процент сокращения размера файла после оптимизации.

До выполнения оптимизации последние три колонки таблицы не содержат информации.

Кроме отдельных файлов, в программе можно выбрать для оптимизации группу папок с HTML-документами. Для этого нажмите кнопку Add Folder (Добавить папку) на панели Toolbar (Панель инструментов).

После того как файлы выбраны, можно провести их оптимизацию. Для этого выполните следующие действия.

1. Нажмите кнопку Optimize (Оптимизировать) на панели Toolbar (Панель инструментов). Появится диалоговое окно Compress (Сжатие) (рис. 8.11).

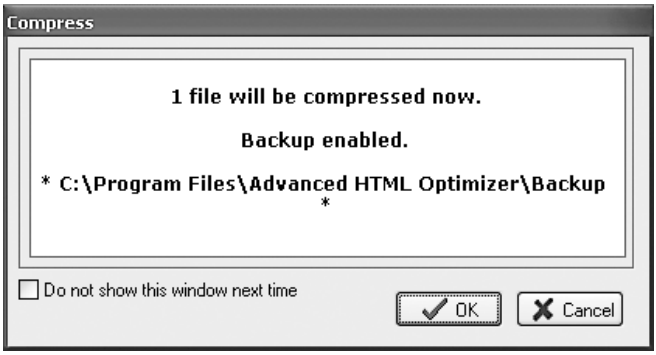


Рис. 8.11. Диалоговое окно Compress (Сжатие)

В этом диалоговом окне программа предупреждает о том, что будет выполнена оптимизация. Установленные по умолчанию параметры оптимизации предусматривают создание копий исходных файлов и сохранение их в папке Backup, вложенной в ту папку, в которой установлена программа. Обычно это папка `C:\Program Files\Advanced HTML Optimizer\Backup`.

2. Закройте диалоговое окно Compress (Сжатие), нажав кнопку OK. Программа выполнит оптимизацию и сообщит о результатах в диалоговом окне Overall compressing performance (Обобщенный результат сжатия) (рис. 8.12).

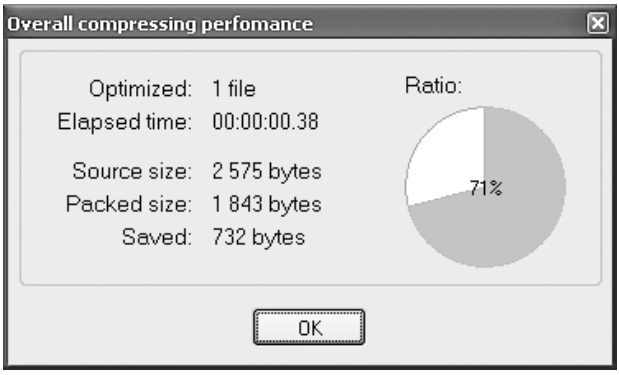


Рис. 8.12. Диалоговое окно Overall compressing performance (Обобщенный результат сжатия)

В этом диалоговом окне программа выводит определенную информацию в соответствующих полях:

- Optimized (Оптимизировано) — количество оптимизированных файлов;
- Elapsed time (Затраченное время) — затраченное на оптимизацию время;
- Source size (Размер источника) — исходный размер файлов;

- Packed Size (Сжатый размер) — размер файлов после оптимизации;
- Saved (Спрятано) — разница между исходным и оптимизированным размером.

3. Закройте диалоговое окно Overall compressing performance (Обобщенный результат сжатия), нажав кнопку OK. Информация об оптимизированных файлах отобразится в таблице File(s) to compress (Файл(ы) для оптимизации) (рис. 8.13).

   Как отмечалось выше, установки по умолчанию предусматривают создание копий исходных файлов в папке Backup. Но вы можете изменить этот параметр, указав на вкладке Backup (Копии) диалогового окна Options (Параметры) другую папку для создания копий исходных файлов. Можно также отказаться от сохранения копий или оставить исходные файлы без изменений, а оптимизированные файлы сохранять в указанной вами папке. Диалоговое окно Options (Параметры) можно открыть, нажав кнопку Options (Параметры) на панели Toolbar (Панель инструментов).

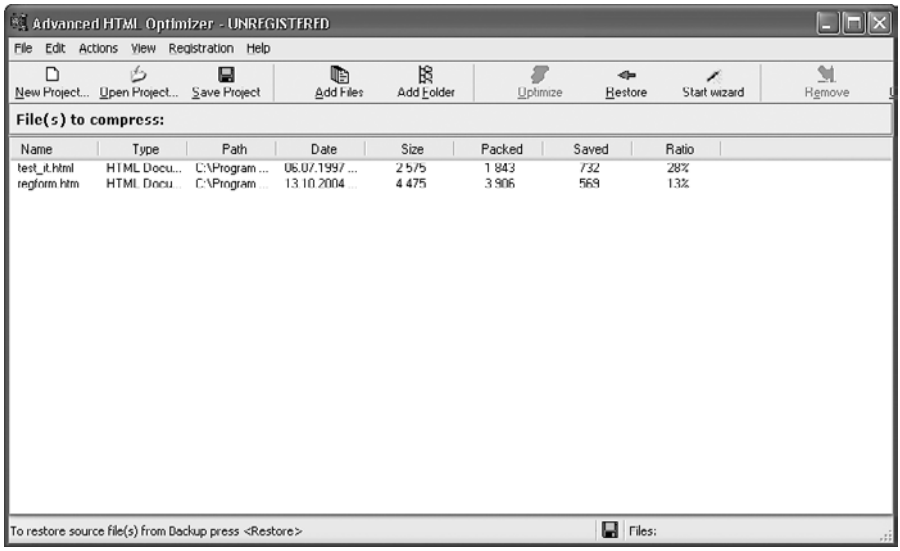


Рис. 8.13. Информация об оптимизированных файлах

В любой момент вы можете восстановить исходное состояние оптимизированных файлов из сохраненных копий. Чтобы произвести восстановление, выполните следующие действия.

1. Нажмите кнопку Restore (Восстановить) на панели Toolbar (Панель инструментов). Появится диалоговое окно Confirm (Подтверждение) с запросом на подтверждение восстановления файлов.

2. Нажмите кнопку Yes (Да), чтобы подтвердить необходимость восстановления. Исходные файлы восстановятся, а информация о них появится в таблице File(s) to compress (Файл(ы) для оптимизации).

Во время оптимизации программа выполняет следующие операции:

- заменяет одиночные теги, например STRONG на B, STRIKE на S;
- удаляет ненужные кавычки в параметрах тегов;
- удаляет ненужные символы двойных пробелов;
- удаляет все символы возврата каретки (CR), если такие имеются;
- удаляет ненужные символы перевода строки (LF);
- удаляет все ненужные символы пробелов и табуляции;
- удаляет комментарии.

На вкладке Optimization (Оптимизация) диалогового окна Options (Параметры) вы можете до мельчайших деталей определить параметры обработки файлов: указать, как и какие теги следует заменять, а какие — оставить без изменений, удалять ли пробелы, символы CR и LF, кавычки и комментарии. Диалоговое окно настройки параметров программы вызывается командой меню Edit ▸ Options (Правка ▸ Параметры). Диалоговое окно Options (Параметры) также появляется на экране при нажатии клавиши F9.

Мы рассмотрели оптимизацию кода веб-страниц с помощью программы Advanced HTML Optimizer. Выполнив все указанные действия, необходимые для оптимизации своей веб-страницы, она не вызовет у посетителей сайта недовольство своей перегруженностью и долгим временем загрузки.

Резюме

В этой главе мы рассмотрели проверку работоспособности сайта. Вы изучили такие действия веб-мастера, как проверка грамотности и актуальности ссылок на сайте, а также применение для этих целей программы Web Link Validator. Мы затронули такой важный вопрос, как оптимизация кода веб-страницы, и рассмотрели, как для решения этой задачи воспользоваться программой Advanced HTML Optimizer.

Теперь можно показать свое творение миру, разместив его в Интернете. Как же выбрать хостинг для сайта? Как разместить информацию на сайте? Ответы на эти вопросы вы получите в следующей главе данной книги.

Глава 9

Размещение сайта и выбор хостинга

Выбор хостинга

Загрузка данных на сайт

Резюме

Разместить сайт в Интернете несложно. Для этого достаточно скопировать файлы, из которых состоит ваш сайт, на один из серверов. Но будьте внимательны! Если вы забудете поместить на сайт какой-либо файл или рисунок, на которые есть ссылка на одной из веб-страниц, браузер посетителя не сможет найти их и отобразить.

Передать файлы на сервер можно несколькими способами: с помощью специальных программ либо используя интерфейс загрузки — файловый менеджер сервера, на котором вы размещаете сайт. Для передачи файлов в обоих случаях применяется протокол HTTP — HyperText Transfer Protocol (Протокол передачи гипертекста), который обеспечивает обмен информацией между передающей программой и веб-сервером.

Для помещения сайта в сеть вы можете воспользоваться разными вариантами. Например, разместить сайт на сервере своего провайдера или же на сервере одной из бесплатных служб в Интернете.

Выбор хостинга

Сегодня в Интернете существует большое количество бесплатных служб, позволяющих всем зарегистрированным пользователям поместить свой сайт на веб-сервере. В свою очередь, службы размещают рекламу на их страницах и за это получают деньги от рекламодателя. Поэтому предоставление бесплатного места на сервере обычно требует от пользователя выполнения определенных условий, например размещения на пользовательских страницах рекламной информации — баннеров или дополнительного окна браузера с рекламой во время загрузки основной страницы. Такие условия не всегда приемлемы, поэтому при выборе бесплатной службы обратите на них внимание.

Чтобы вашу страницу посещали, она должна быть интересной, а информация на ней часто обновляться. Веб-страница может обновляться с помощью файлового менеджера сервера. Зачастую очень важна скорость вступления обновлений в силу. При выборе сервера обратите внимание на то, чтобы обновление данных было для вас максимально удобным.

Для многих пользователей имеет значение вид адреса, который служба присвоит сайту, так как они предпочитают короткие и компактные адреса длинным и трудно запоминающимся.

Часто службы размещения сайтов — хостинги — предоставляют дополнительные услуги, например счетчик посещений, гостевую книгу, чат, форум, возможность

участия в конкурсах на лучшую страницу и др. В зависимости от целей создания сайта перечень этих услуг может оказаться определяющим для вас при выборе службы.

Немаловажное значение имеет также скорость работы сервера, на котором вы хотите разместить свои страницы. Вряд ли посетители захотят часто заходить на ваш сайт, если страницы долго загружаются.

И, наконец, выбирая хостинг, решите, какой объем памяти на диске вам потребуется. Обычно эти службы предоставляют каждому пользователю от 512 Кбайт до 20 Мбайт дискового пространства. Нескольких мегабайт вполне достаточно для небольшого сайта. Однако если разрабатывается масштабный проект, следует реально оценить его перспективы и возможности бесплатной службы по дальнейшему увеличению дискового пространства.

К сожалению, идеальных хостингов не существует. Каждый хостинг направлен на решение каких-то конкретных задач и обладает определенными достоинствами. Например, один хостинг может идеально подходить для хранения файлов больших размеров, а другой обладает интерфейсом, максимально удобным для обновления, подключения и отключения определенных услуг.

Часто пользователь при выборе сервиса для размещения сайта отдает предпочтение бесплатным службам. Достоинством таких сервисов, несомненно, является их бесплатность. К недостаткам же можно отнести обязательное размещение рекламы на сайте.

Приведу краткий обзор наиболее популярных хостингов, остановившись на их особенностях, достоинствах и недостатках.

Обзор популярных хостингов

В начале нашего небольшого обзора рассмотрим платные сервисы как наиболее функциональные и надежные.

Хостинг-компания MHost.ru предоставляет пользователю от 5 до 15 Гбайт дискового пространства для хранения файлов большого размера. Хостинг поддерживает все современные технологии: позволяет обрабатывать PHP-сценарии, работать с базами данных, а также просматривать статистику посещаемости ресурсов. Сайт предоставляет свои услуги пользователям за весьма скромную плату, что, несомненно, является большим плюсом. Электронный адрес этого ресурса — www.mchost.ru. На рис. 9.1 показано окно сайта компании MHost.ru.



Рис. 9.1. Сайт хостинг-компании MHost.ru

Хостинг-центр РБК характеризуется высокой надежностью, малым временем отклика и практически постоянным доступом к размещенным на сервере ресурсам. Электронный адрес хостинг-центра РБК — www.hc.ru. На рис. 9.2 вы видите сайт хостинг-центра РБК.

Хостинг-центр «Джино» отлично подходит для размещения в сети больших проектов. С помощью одной учетной записи можно регистрировать несколько сайтов, есть возможность самостоятельно конструировать тарифы и выбирать подключаемые услуги. Кроме того, сервис отличается высокой надежностью, доступностью и малым временем отклика. Если вы ищете по-настоящему универсальный хостинг, который обладал бы простотой, надежностью и функциональностью, то хостинг «Джино» для вас. Именно его я и выбрал для дальнейшего повествования.

Рассмотрим и бесплатные сервисы. Ими вы всегда сможете воспользоваться, не задумываясь об оплате.

Сервис Eomy.net (электронный адрес сайта — www.eomy.net) считается лучшим бесплатным хостингом для небольших проектов. Зарегистрировавшись, вы получаете

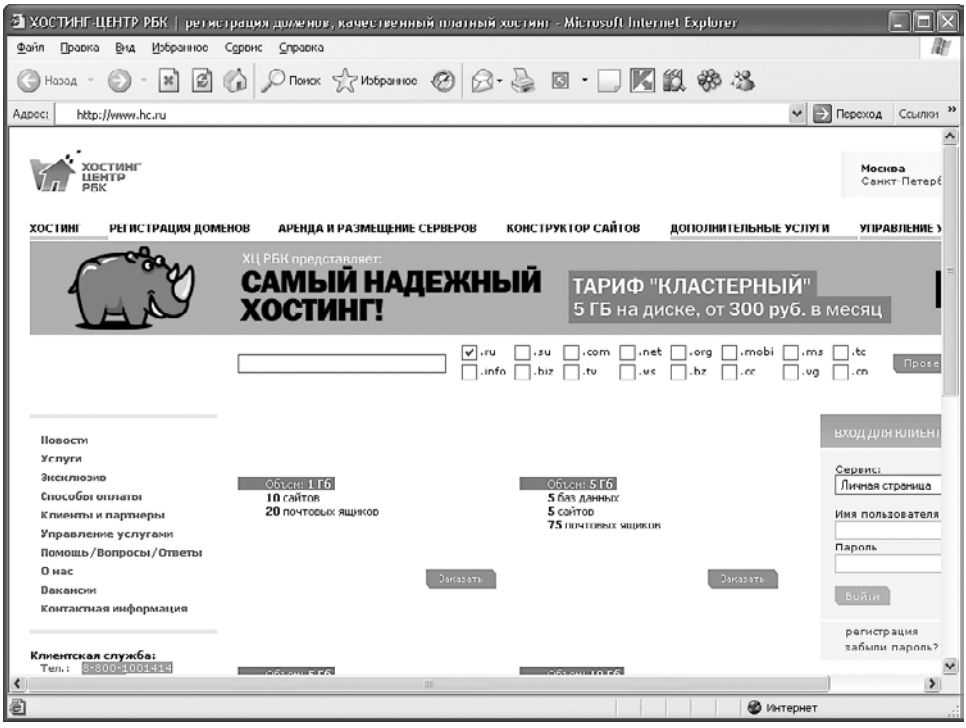


Рис. 9.2. Сайт хостинг-центра РБК

до 300 Мбайт дискового пространства и 30 Гбайт входящего/исходящего трафика в месяц. Сайт поддерживает все современные технологии PHP, SSI и Perl. В свою очередь, сервис будет показывать текстовую рекламу в верхней части всех страниц вашего сайта. К недостаткам можно отнести отсутствие возможности обмена почтовыми сообщениями, так как компания не предоставляет услуги электронной почты.

Простой хостинг Ayola.net (электронный адрес сайта — www.ayola.net) сравним по функциональности с платными службами. Быстрые и надежные серверы, поддержка современных технологий, выделение каждому пользователю до 500 Мбайт дискового пространства, неограниченный трафик, почтовый ящик, поддержка PHP, MySQL — все это несомненные достоинства хостинга.

Сервис Jino-Net.ru (электронный адрес сайта — www.jino-net.ru) использует уникальные технологии платной хостинговой системы «Джино» — надежные сервера и удобную контрольную панель. После регистрации бесплатной учетной записи вы получаете 70 Мбайт места на диске и возможность пользоваться всеми современными технологиями PHP5, MySQL, SSI и FTP. Вы сможете также перевести свою учетную запись на платный хостинг «Джино».

Вы узнали о платных и бесплатных сервисах, к которым можете обратиться для размещения своих сайтов. Теперь рассмотрим более подробно работу по размещению сайта на сервисе «Джино».

Сервис «Джино»

Несмотря на довольно большое количество бесплатных сервисов, для детального рассмотрения мы выбрали платный хостинг «Джино». Примечательно, что наряду с другими возможностями вы можете опробовать услуги сервиса в бесплатном демо-режиме до принятия решения о покупке места на нем для своего сайта.

Хостинг «Джино» обладает следующими достоинствами:

- возможностью подключения нужных пользователю услуг в режиме онлайн;
- возможностью создания и использования единой учетной записи для всех ваших сайтов;
- удобной панелью управления;
- низкими ценами и удобством внесения платежей за услуги хостинга;
- возможностью бесплатного тестирования сервиса в демонстрационном режиме в течение 10 дней.

Итак, сервис выбран. Теперь его нужно зарегистрировать. Об этом и пойдет речь в следующем подразделе.

Регистрация на сервисе «Джино»

Для регистрации на сервисе «Джино» выполните следующие действия.

1. Наберите в строке браузера адрес сервиса «Джино» `http://jino.ru/`. Браузер отобразит стартовую страницу (рис. 9.3).
2. Щелкните кнопкой мыши на надписи Зарегистрироваться, чтобы начать регистрацию в десятидневном демо-режиме. Отобразится форма регистрации учетной записи (рис. 9.4).
3. Заполните все поля формы: введите логин, пароль и контактные данные. Поля, отмеченные красной звездочкой, являются обязательными для заполнения.

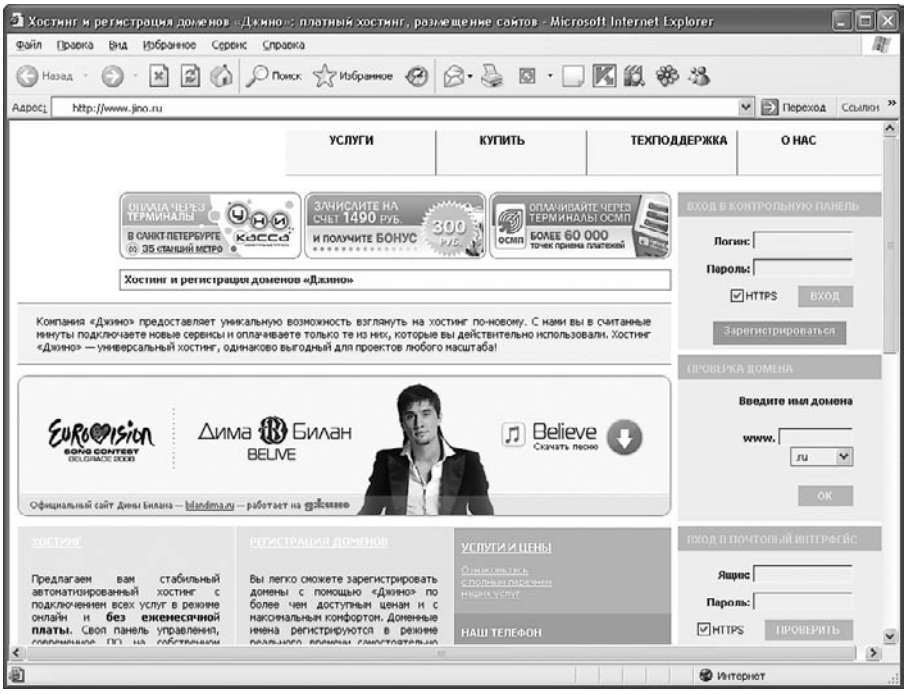


Рис. 9.3. Сайт сервиса «Джино»

4. Установите переключатель Тип аккаунта в одно из трех положений, в зависимости от того, как вы будете регистрироваться: как физическое, юридическое лицо или индивидуальный предприниматель.
5. Ознакомьтесь с условиями предоставления услуг данного сервиса, щелкнув кнопкой мыши на ссылке Условия предоставления услуг в нижней части формы регистрации.
6. Установите флажок Я ознакомлен и согласен с условиями предоставления услуг и введите код проверки в поле Проверка регистрации.
7. Нажмите кнопку Зарегистрироваться. Вы получите извещение об успешной регистрации и предложение войти на сайт (рис. 9.5).

Сразу после регистрации на указанный вами в анкете адрес придет электронное письмо с сообщением об успешной регистрации на сайте, с вашими логином и паролем. Сохраните письмо, чтобы в дальнейшем можно было узнать логин и пароль, если вы их забудете.

Итак, вы зарегистрировались. Теперь пора войти на сайт и познакомиться с контрольной панелью.

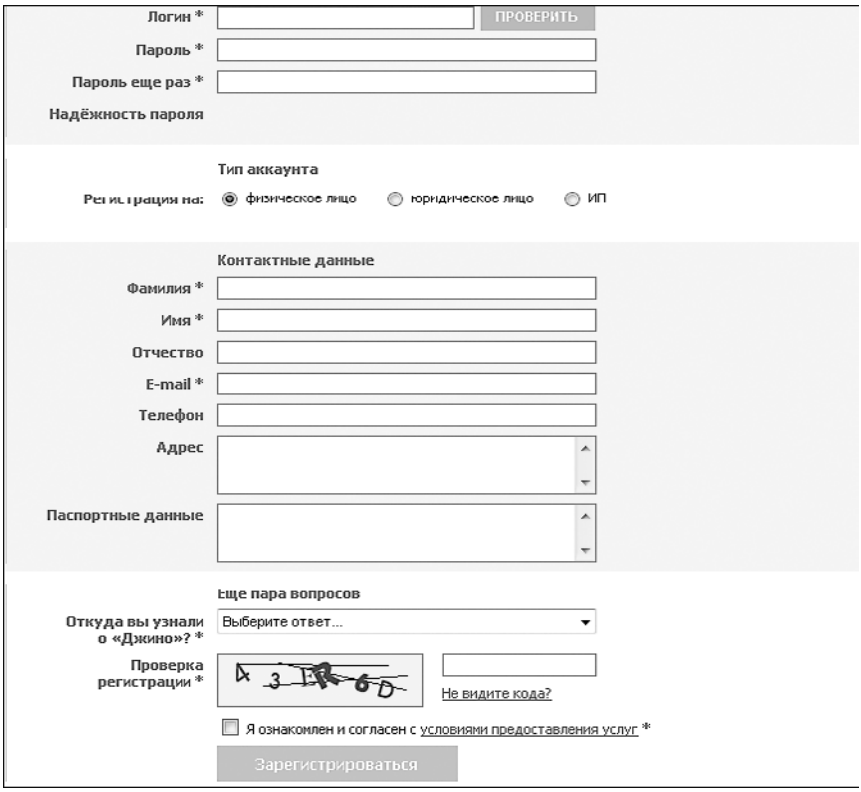


Рис. 9.4. Форма регистрации учетной записи



Рис. 9.5. Регистрация завершена

Контрольная панель «Джино»

Перед вами окно браузера с завершенной регистрацией и предложением войти на контрольную панель.

Нажмите кнопку со стрелкой и надписью Войти, чтобы увидеть свою панель управления Контрольная панель «Джино» (рис. 9.6).

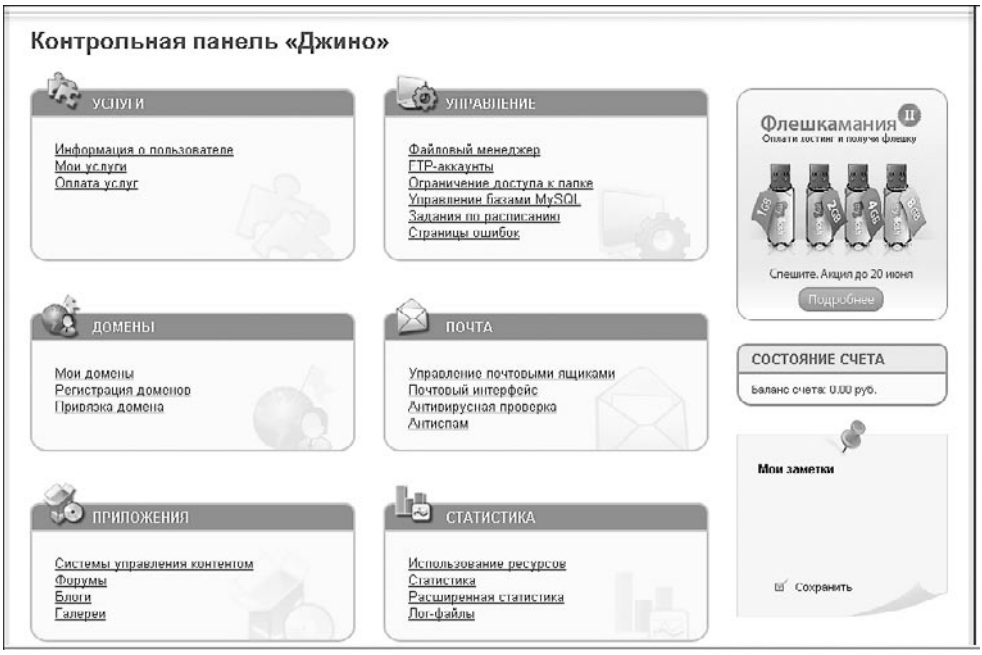


Рис. 9.6. Контрольная панель «Джино»

Контрольная панель делится на следующие категории:

- Услуги — здесь вы можете узнать об оплате услуг, выбрать нужные вам услуги, а также в этой категории содержится информация о пользователе;
- Управление — здесь сосредоточены все компоненты, которые помогут вам управлять сайтом: Файловый менеджер (для загрузки данных), FTP-аккаунты, Ограничение доступа к папкам, Управление базами MySQL, Задания по расписанию и Страницы ошибок;
- Домены — вы можете регистрировать домены, просматривать свои домены и привязывать их к своей учетной записи;
- Почта — категория почтовых отправлений и управления почтовыми ящиками, антиспамом и антивирусной проверкой;

- Приложения — здесь находятся различные дополнения — форумы, блоги, галереи, а также системы управления контентом;
- Статистика — здесь вы можете следить за статистикой вашего сайта.

Теперь необходимо определиться с услугами, которые вы хотите получать на данном сервисе. Можно отказаться от тех услуг, которые вам не нужны. Для управления услугами выполните следующие действия.

1. Щелкните кнопкой мыши на строке Мои услуги категории Услуги. На экране появится страница со списком подключенных услуг.
2. Для корректировки подключения услуг щелкните кнопкой мыши на вкладке Подключение услуг. Отобразится список услуг с их описанием и статусом подключения (рис. 9.7).

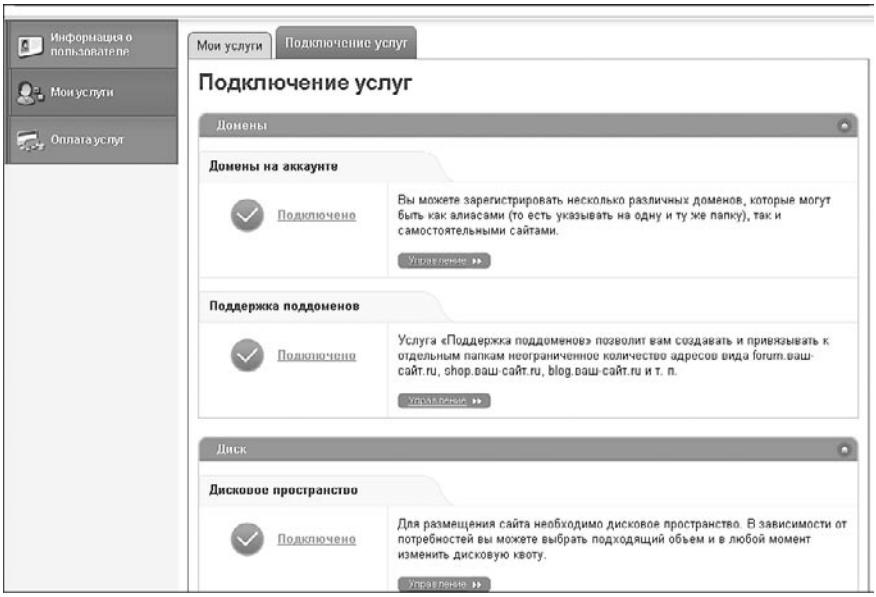


Рис. 9.7. Список подключенных и отключенных услуг

3. Чтобы подключить/отключить услугу, нажмите кнопку Управление, которая находится под описанием каждой услуги. В отобразившемся окне описания услуги и ее стоимости вы можете выбрать интересующий вас вариант и нажать кнопку Изменить (если тарифный план был изменен) или Отключить (если услуга не требуется).

Сервис «Джино» предлагает очень удобное управление, которое осуществляется с контрольной панели. Управление очень простое и понимается интуитивно, оно

не должно вызвать у вас никаких затруднений. Впрочем, если у вас появились какие-либо вопросы, то можете обратиться в службу поддержки или же воспользоваться рубрикой Вопросы и ответы категории Техподдержка.

Теперь, когда вы зарегистрировали свой сайт и получили всю необходимую для доступа информацию, можно приступить к пересылке файлов. В следующем разделе мы рассмотрим, как для этой цели использовать интерфейс загрузки сервера.

Загрузка данных на сайт

Чтобы закачать данные на сайт с помощью интерфейса загрузки сервиса, выполните следующие действия.

1. На контрольной панели щелкните кнопкой мыши на ссылке Файловый менеджер категории Управление. Отобразится страница со встроенным файловым менеджером (рис. 9.8).

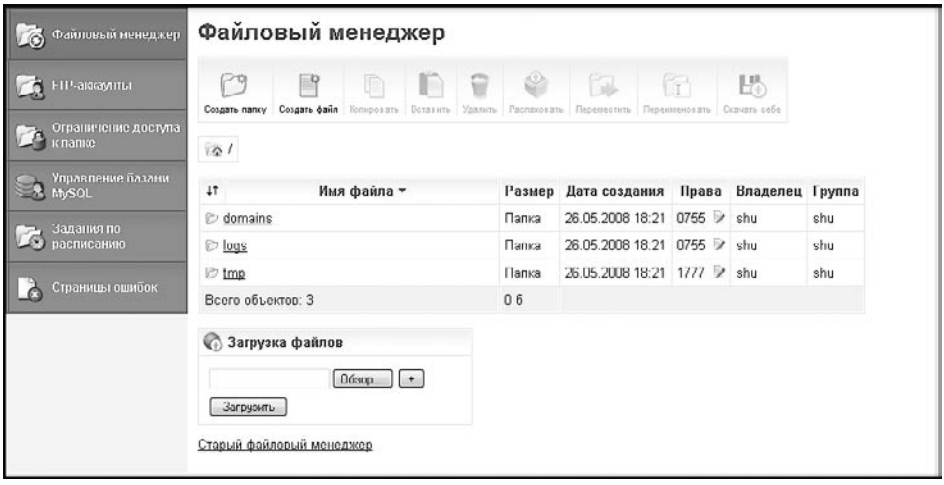


Рис. 9.8. Файловый менеджер

В верхней части вы видите панель инструментов, с помощью которой можно создать новую папку, файл, удалить и распаковать данные, скопировать и вставить данные, переместить и переименовать их.

Под панелью инструментов находится список папок с указанием дат их создания. Под этим списком располагается область Загрузка файлов, с помощью которой данные загружаются на сервер. Все файлы сайта хранятся в папке domains. Поместите в нее загруженные данные, выполнив следующие действия.

1. Выделите в списке папок в окне Файловый менеджер строку с названием папки domains, щелкнув на ней кнопкой мыши. Отобразится содержимое папки.
2. Нажмите кнопку Обзор в области Загрузка файлов. Откроется диалоговое окно Выбор файла, где нужно указать файл, который вы хотите загрузить на сервер.
3. Указав файлы или папку для загрузки, нажмите кнопку Открыть в диалоговом окне Выбор файлов. В поле рядом с кнопкой Обзор появится путь к выбранному файлу и кнопка с изображением плюса.
4. Щелкните кнопкой мыши на появившейся кнопке. Появится поле для выбора еще одного загружаемого файла. Вы можете создать столько таких полей, сколько нужно загрузить файлов.
5. Нажмите кнопку Загрузить. Загруженные файлы появятся в списке файлов папки domains.

Введя в строке браузера адрес своего сайта, вы сможете увидеть свое творение в окне браузера. Теперь ваш сайт могут увидеть посетители и оценить его по достоинству.

Резюме

В этой главе вы узнали, как выбрать хостинг для размещения своего сайта, а также на какие моменты следует обратить внимание при выборе сервера. Мы сделали краткий обзор популярных хостингов, чтобы облегчить вам поиск приемлемого варианта, а также подробно рассмотрели процедуру регистрации учетной записи на одном из платных сервисов для размещения сайтов. Вы узнали, как пользоваться контрольной панелью и загружать данные с помощью встроенного файлового менеджера. Надеюсь, полученных из этой главы сведений достаточно, чтобы вы смогли выбрать хостинг и, зарегистрировавшись на нем, разместить на нем свой сайт.

Размещение сайта в сети, конечно, важный момент в «сайтостроении», но отнюдь не завершающий. Теперь необходимо «раскрутить» сайт, привлечь к нему посетителей и зарегистрировать его в каталогах. Как все это сделать, я расскажу в следующей главе.

Глава 10

Раскрутка сайта

Поисковые машины

Каталоги

Рейтинги

Баннеры

Элементы привлечения посетителей

Резюме

Для успешности вашего веб-проекта недостаточно просто создать информативный и интересный сайт. Если никто не узнает о вашем проекте, то все усилия окажутся напрасными. Поэтому необходимо заняться продвижением вашего детища, чтобы как можно больше людей узнали о вашем сайте и посетили его.

Под *продвижением сайта* понимается комплекс мер, призванных повысить популярность сайта, его посещаемость и известность. Задача популяризации сайта состоит не столько в создании привлекательного имиджа компании, сколько в привлечении на сайт посетителей. Для этого используются следующие способы:

- пассивная реклама — заинтересованный потребитель сам ищет нужный ему товар;
- активная реклама — товар сам ищет своего потребителя (этот способ распространен больше).

Пассивная реклама сводится к регистрации сайта в поисковых системах, каталогах и рейтингах и обмену ссылками. Иногда она способна привлечь достаточно много посетителей. С другой стороны, если вы заинтересованы не в количестве посетителей, а в их способности воспользоваться вашими, возможно, платными услугами, то эти посетители представляют для вас особый интерес: они искали именно ваш сайт и вполне могут потратить на вас свои деньги.

К *активной рекламе* относятся все рекламные ролики, показываемые на телевидении, рекламные тексты на радио, а в Интернете — баннеры, то есть картинки, щелчок кнопкой мыши на которых переводит посетителя на сайт рекламодателя. Именно в этот способ рекламы вкладывается наибольшая часть денег, затрачиваемых на продвижение проекта. Активная реклама подразумевает поиск потребителя товаром, а не наоборот, как в случае пассивной рекламы. Сетевой аспект этого метода состоит в привлечении посетителя на сайт, далее же задача сводится к тому, чтобы не дать посетителю уйти и заставить его что-то купить (если сайт коммерческий).

Создавая сайт, вы вправе рассчитывать на его хотя бы частичную окупаемость. Владельцы любых сайтов могут получить деньги за рекламу на своих страницах. Но не всякий сайт может принести много денег. И лишь некоторые сайты способны полностью оправдать затраты на свое содержание. Чтобы сайт окупился, обязательно должны выполняться несколько условий, одним из которых является хорошая посещаемость. А для этого сайт должен иметь интересное содержание и приемлемое оформление. Ведь никакая популяризация не гарантирует хорошей окупаемости плохому ресурсу.

Каким же образом окупаются затраты на создание сайта и зарабатываются на нем деньги? Сделать это можно двумя способами:

- предоставление платных услуг — в этом случае не нужна большая посещаемость — важна платежеспособность посетителей сайта и их заинтересованность;
- размещение рекламы — этот способ требует большой посещаемости вашего ресурса.

Поместить рекламу на своем сайте можно двумя методами: либо самостоятельно найдя себе рекламодателя, либо воспользовавшись услугами спонсоров. Лучше всего создать интересный, посещаемый ресурс на популярную тематику и найти рекламодателя, готового работать именно с вами. Этот метод самый трудоемкий, но зато приносит наибольший доход.

Для популяризации сайта и привлечения посетителей очень важно, чтобы поисковые машины могли без проблем находить ваш ресурс. Как это сделать? Об этом и пойдет речь в следующем разделе.

Поисковые машины

Как известно, в настоящее время основным средством нахождения информации в Интернете являются поисковые машины, которые ищут десять ссылок и в дальнейшем разворачивают тему по смежным ссылкам.

Ежедневно к поисковым машинам обращаются несколько десятков миллионов человек, которые ищут какую-либо информацию. Возможно, эти сведения находятся на вашем сайте. Несколько российских поисковых машин обрабатывают в день более ста тысяч запросов и являются одними из самых посещаемых сайтов в русскоязычном Интернете. Поэтому очень важно, чтобы ваш сайт был доступен поисковым машинам. Исходя из этого следует обратить особое внимание на оптимизацию вашего сайта для поисковых машин.

Робот, или, как его еще называют, паук, — это основа поисковой машины. Он оценивает текст вашей страницы и определяет его соответствие (релевантность) введенному запросу. Одним из важнейших параметров релевантности документа является частота употребления ключевого слова в тексте. Следовательно, ваша задача — найти ключевые слова, в максимальной степени соответствующие тематике сайта, и оптимизировать страницы именно под эти ключевые слова.

Ключевые слова — это слова, которые наиболее характерны для описываемого вами предмета. Они являются основным и самым важным критерием для работы поисковых машин. Поэтому перед тем как подавать запрос на включение вашего сайта в базу данных поискового робота, внимательно проанализируйте текст на главной странице сайта. Определите, какое слово или выражение встречается на ней чаще всего. Характерно ли это слово для тематики сайта или это разговорный оборот? Как часто встречается это выражение?

Проанализировав текст на странице, определите ключевые слова, которые достаточно точно и ярко характеризуют ваш ресурс. Сохраните их в отдельном текстовом файле. После этого переработайте текст на странице (речь идет только о той странице, которую вы будете рекламировать) таким образом, чтобы выбранные вами ключевые слова встречались там регулярно (в среднем не менее 5–10 раз). Подобным образом переработайте название страницы, которое отображается в заголовке браузера, подписи к рисункам и т. д.

Очевидно, что оптимизировать страницу под большое количество ключевых слов крайне сложно, поэтому постарайтесь обойтись двумя-тремя словами. Для этого выберите несколько самых «ярких» слов и подстройте под них страницу. Учитывая особенности поисковых машин, следует особенно тщательно обработать первый килобайт текста страницы.

Чтобы поисковый робот смог найти и понять ваши ключевые слова, нужно оформить их в специальную конструкцию, называемую *мета-тегами*. Это необязательные атрибуты веб-документа, которые содержат описание страницы, ключевые слова к ней, некоторую информацию об авторе, управляющие команды для поисковых роботов и прочую служебную информацию, не предназначенную для всех посетителей. Мета-теги помещаются в начале HTML-кода между тегами `<head>` и `</head>` и выглядят следующим образом:

```
<head>
<title>Заголовок</title>
<meta http-equiv="autor" content="Автор страницы">
<meta http-equiv="KEYWORDS" content="Ключевые слова через запятую">
<meta http-equiv="DESCRIPTION" content="Описание страницы">
</head>
```

Мета-теги необходимо вставить в код каждой страницы вашего сайта. Это значит, что каждая страница должна содержать собственные ключевые слова и описание, которые ей наиболее соответствуют. Это довольно большая работа, но необходимая. От нее, в частности, зависит количество посетителей, которые зайдут на ваш сайт.

Все современные HTML-редакторы, независимо от того, работаете вы непосредственно с кодом или же в визуальном редакторе, позволяют вставлять мета-теги автоматически. Достаточно указать в соответствующем мастере или диалоге описание страницы вместе с ее ключевыми словами — и мета-теги добавятся.

ВНИМАНИЕ



Каждая страница вашего сайта, кроме своего названия, должна иметь собственные, наиболее подходящие для нее ключевые слова и описание. Часть ключевых слов, общих для всего ресурса, должна присутствовать в мета-тегах на всех страницах сайта, но там могут находиться также и свои, уникальные для каждой страницы слова. Каждая страница вашего сайта имеет оригинальное содержание, а пользователь, возможно, будет искать через поисковую машину. Следовательно, уникальные особенности каждой страницы полезно отразить в мета-тегах.

Очень важно, чтобы ключевые слова не просто описывали вашу тему, но и соответствовали основному тексту страницы, повторялись в нем. В противном случае все ваши усилия пропадут даром, потому что страница, содержащая искомые ключевые слова только в мета-тегах, скорее всего, будет признана роботом нерелевантной. Поэтому чем чаще ключевые слова будут встречаться в основном тексте вашей страницы, тем лучше.

Не менее важным является описание титульной страницы, поскольку оно должно характеризовать ваш сайт целиком: его особенности, направления работы, его уникальность и значимость. Хорошо описав титульную страницу, описать каждую страницу несколько проще — нужно сузить общее описание до темы конкретной страницы. Удачным решением можно считать включение в мета-теги внутренних страниц описания и ключевых слов, составленных для всего сайта.

Длина каждого поля `content` в мета-тегах ограничивается 1024 символами, включая пробелы и знаки препинания. Этого более чем достаточно, чтобы сделать развернутое описание. Однако нужно помнить, что мета-теги включаются в код страницы и увеличение их длины ведет к повышению объема страницы, а следовательно, делает более долгим время загрузки. Поэтому не следует увлекаться. Кстати, в целях экономии места можно помещать в мета-теги внутренних страниц сокращенную версию описания всего ресурса.

Исключительно важна также правильность составления заголовка веб-страницы. Ведь именно он отображается в первой строке окна браузера, а затем — в закладках пользователей. Его будут показывать поисковые машины в результатах поиска и на него в первую очередь обратят внимание обозреватели сетевых ресурсов. Поэтому заголовок следует подбирать очень тщательно и внимательно.

Заголовок помещается в специальной конструкции между открывающим тегом <title> и закрывающим тегом </title> и может содержать 256 символов. На самой странице заголовок не отображается, если только вы его не вставите отдельно. Несмотря на то что в вашем распоряжении есть 256 символов, делать название длиной больше 60–80 знаков не стоит, поскольку ширина окна браузеров и закладки не вмещают длинные строки и «обрезают» их. А такой заголовок выглядит не самым лучшим образом.

Название страницы следует начинать со значимого слова, например названия продукта, торговой марки, услуги или тематики. Проследите, чтобы названия внутренних страниц содержали название самого сайта — таким образом вы подчеркнете их принадлежность. Это особенно важно, когда страниц много и разброс тем велик. Дело в том, что с поисковых машин посетитель часто попадает на внутренние страницы сайта, минуя титульную страницу, а следовательно, он не будет знать ни названия вашего узла, ни его адреса.

После того как вы снабдили страницы мета-тегами с описаниями и ключевыми словами, они готовы к регистрации в поисковых машинах.

Поисковая машина состоит из робота-паука и индекса. Паук постоянно обследует Интернет и все найденные ресурсы заносит в специальную базу данных — индекс. Таким образом, поиск по запросу выполняется не по реальным сайтам, а по записям в индексе. Теоретически паук обязательно найдет ваш сайт, но когда именно это произойдет — неизвестно. Поэтому можно предложить поисковому роботу проиндексировать ваш сайт, сообщив ему адрес. Заполняя специальную форму, вы записываете свой сайт в «очередь» на индексацию, и через несколько дней его адрес появится в базе данных поисковой машины.

На каких поисковых машинах и как следует регистрироваться? Многие пользователи считают, что нужно послать координаты своего сайта в базы данных как можно большего количества поисковиков. На сегодня существует несколько тысяч больших поисковых машин, при этом 98 % запросов обрабатывается не более чем десятью из них. Остальные либо не являются общеизвестными, либо локализованы по языку, а следовательно, бесполезны для вас (если они не русскоязычные). Исходя из этого можно сделать вывод, что реальную ценность представляют не более десятка иностранных и пяти-шести российских поисковых роботов.

Из русскоязычных поисковых систем обратите особое внимание на три поисковые машины — Rambler, Яndex и Апорт. Они не только понимают русский язык, но именно через них проходит основная масса запросов на русском языке. Следовательно, именно русские поисковые машины предпочтительны для русскоязычного сайта.

Rambler (http://www.rambler.ru/) — эта колоссальная поисковая машина, объединенная с самым популярным одноименным рейтингом, без сомнения, является самым популярным комплексом в русскоязычном Интернете. Причем суммарная посещаемость настолько велика, что сравняться с ней практически невозможно.

К сожалению, поисковая машина Rambler игнорирует мета-теги, поэтому работать с ней сложно и не очень удобно. Наибольшее значение в определении релевантности страницы имеют выделенные конструкции — заголовки, подзаголовки, текст полужирного шрифта и т. д., а также текст самой страницы. Для регистрации в этой поисковой системе необходимо заполнить специальную форму, при этом повторять слова нельзя и следует обязательно заполнить все поля.

Многие пользователи считают работу с этой системой неудобной из-за формы вывода результатов, механизма поиска и не очень качественного по сравнению с другими машинами отбора сложной регистрации. Однако ею пользуются десятки тысяч человек в день, поэтому зарегистрироваться на ней необходимо. Форма для регистрации находится по адресу http://www.rambler.ru/doc/add_site_form.shtml. Внешний вид страницы в окне браузера показывается на рис. 10.1.

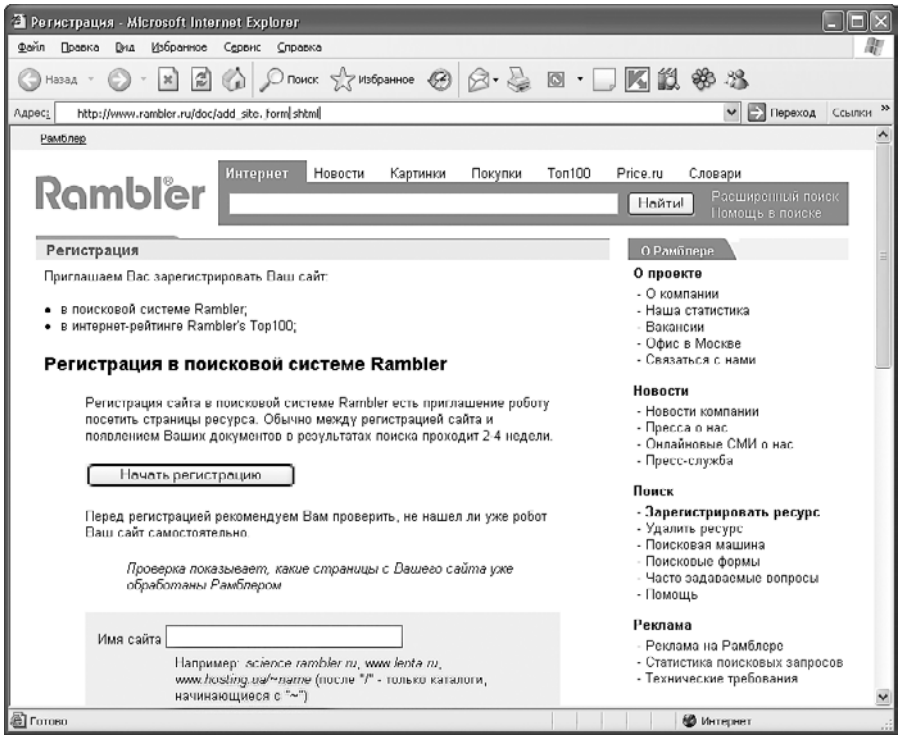


Рис. 10.1. Форма регистрации в поисковой машине Rambler

Поисковая система Яndex (http://www.ya.ru/) понимает русский язык, а также в состоянии понять и образовать любую словоформу русского языка. Это означает, что если в качестве запроса ввести слово, например, «читать», то машина выдаст вам результаты поиска по словам «читал» и «прочитал». Вы можете ввести нужную вам фразу, например «компьютерное железо», — и машина выдаст осмысленные результаты. С точки зрения веб-мастера эта система удобна также тем, что регистрация нового ресурса происходит в течение одного-двух дней. Быстрый робот систематически сканирует практически все пространство русскоязычного Интернета и регулярно вносит изменения в индекс. Форма для регистрации в этой системе находится по адресу http://www.yandex.ru/addurl.html.

Поисковая система Апорт (http://www.aport.ru/) объединяется на одном сервере с каталогом и рейтингом. Она обладает широкими и интересными возможностями, например такими, как поиск сайтов, а не страниц, индекс цитируемости, учет доступности и др. Имеется механизм поиска изображений. Форма для регистрации находится по адресу http://www.aport.ru/addurl.php.

Сравнительно недавно появилась поисковая система Eprst (http://www.eprst.ru/). Но обретет ли она достаточно большую популярность, чтобы сравниться с ведущей тройкой, — покажет время. Форма для регистрации находится по адресу http://lovers.eprst.ru/reg/index.php3.

Рекомендуем также обратить внимание и на другие поисковые системы, например на поисковые машины Russia on the Net (http://www.ru/rus/index.html) и Yahoo! (http://www.yahoo.com/).

Практически все ведущие зарубежные поисковые роботы умеют индексировать русскоязычные документы. Такие гиганты, как AltaVista, HotBot, GoTo.com., Google и другие, посещает несколько миллионов пользователей ежедневно, то есть значительно больше, чем русскоязычные. Поэтому регистрироваться в этих системах тоже нужно, так как это увеличит приток посетителей. Перед тем как зарегистрироваться в англоязычной поисковой машине, переведите описание и ключевые слова на английский язык; их можно продублировать и в мета-тегах. В форме регистрации на сайте поисковой машины лучше вводить либо английский текст, либо русский с переводом на английский.

Поисковая машина AltaVista (http://www.altavista.com/) (рис. 10.2), пожалуй, наиболее известная долгожительница из нынешних лидеров поисковых машин, хотя и не самая удобная. Она способна осуществлять поиск почти на тридцати языках и переводить с одного языка на другой. Эта поисковая машина очень быстро индексирует страницы — обычно в течение двух-трех дней.

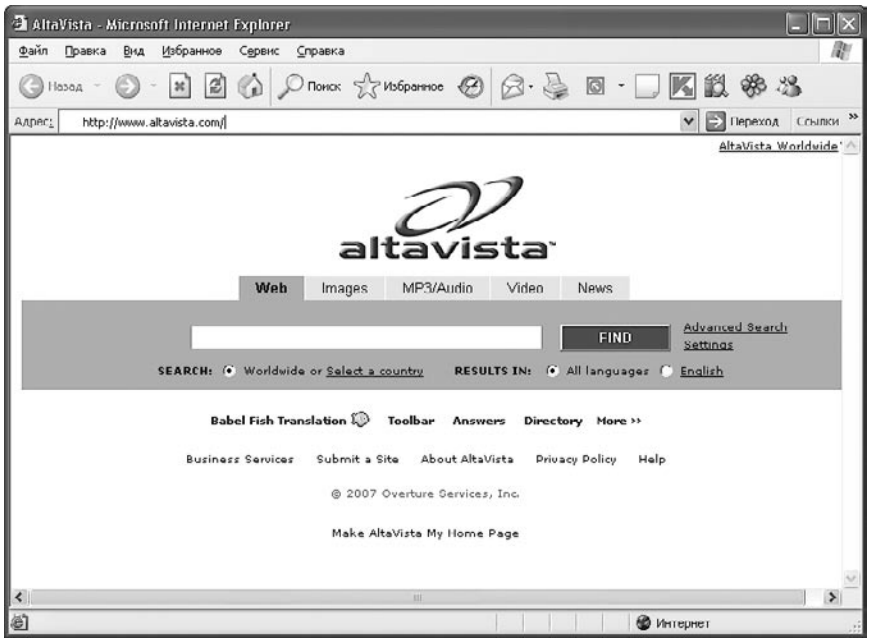


Рис. 10.2. Окно поисковой системы AltaVista

Поисковая машина GO.com (http://www.go.com/) способна искать на многих языках, в том числе по базам MP3-файлов и картинкам. Очень хорошо осуществляет поиск по запросам из целых фраз, при этом безошибочно определяет релевантные документы. Очень быстро индексирует страницы. Считается на данный момент одной из самых удобных и точных поисковых машин.

Поисковая машина HotBot (http://www.hotbot.com/) умеет искать на нескольких языках, в том числе и на русском. К недостаткам данной машины можно отнести то, что она не ищет словоформы и медленно регистрирует сайт.

Поисковая машина Google (http://www.google.com/) во многих случаях дает результаты поиска на русском языке лучшие, чем любая иная русская поисковая машина.

Помимо упомянутых иностранных поисковых машин, стоит обратить внимание и на другие поисковые машины, например Lycos (http://www.lycos.com/), MSN (http://www.msn.com/), AOL (http://www.aol.com/) и Nothern Light (http://www.nothernlight.com/).

Регистрироваться в поисковых системах лучше всего самостоятельно, последовательно заполняя формы на каждом поисковом сервере. Это займет некоторое время, но гарантирует, что ваш сайт включен в списки. Однако если у вас нет

времени, то можно воспользоваться специализированными службами, которые бесплатно регистрируют сразу на многих поисковых машинах.

Существует также специальное программное обеспечение, предназначенное для автоматической регистрации вашего сайта на большом количестве поисковых машин. Например, программы AddWeb (http://www.cyberspacehq.com/) и Register Pro Promotion Spider (http://www.sharewarejunkies.com/9zwd11/register_pro.htm) способны зарегистрировать ваш сайт на сотнях и даже тысячах поисковых машин одновременно. Конечно, эти программы небесплатны, а их демонстрационные версии работают только с несколькими десятками роботов. Тем не менее оптимальной является схема, когда вы регистрируетесь на самых важных поисковых машинах вручную, а на остальных — автоматически.

Если после регистрации на поисковых машинах вы отложите на неделю дальнейшую «раскрутку», то в скором времени увидите результат. В зависимости от темы ваш сайт ежедневно будут посещать от единиц до нескольких десятков человек. Эти люди нашли сайт в тех поисковых системах, в которых вы зарегистрировались. Конечно, у вас должен быть установлен счетчик посещений. Помните, что поисковым машинам и людям, которые ищут информацию, абсолютно безразлично, сколько времени существует сайт и какова его посещаемость на данный момент.

Каталоги

Поиск информации в Интернете можно осуществлять не только с помощью поисковых машин, но и в каталогах. Службы каталогов — это огромные структурированные коллекции ссылок. Люди используют сайты каталогов, чтобы найти необходимую информацию, изучить новую для себя область знаний или же просто поискать что-нибудь интересное. Каталогов в сети огромное количество, среди них есть и супергиганты, и совсем простенькие каталоги. Каждому веб-мастеру, конечно, выгодно, чтобы такие каталоги содержали ссылку на его ресурс. В этом случае все заинтересованные пользователи смогут найти его сайт в «джунглях» Интернета. На первый взгляд очевидно, что чем больше каталогов содержат ссылку на ваш сайт, тем больше посетителей на него зайдет. Однако это верно лишь отчасти. Дело в том, что многие каталоги сами по себе обладают очень низкой популярностью: иногда это сотня посетителей в день, а иногда и того меньше. В таком случае вы, скорее всего, не получите никакой отдачи от каталога, а на регистрацию в нем потратите время.

С другой стороны, каталогам выгодно, чтобы в них регистрировалось как можно больше сайтов. Поэтому, помимо поиска в сети новых ресурсов, они предоставляют

владельцам сайтов возможность публиковать свои ссылки самостоятельно. Для этого во всех каталогах существуют специальные формы, которые находятся за надписями Добавить ссылку, Добавить URL или просто Добавить. Заполнив такую форму, в которой указываются некоторые данные о сайте, веб-мастер имеет шанс попасть в соответствующий каталог, что в дальнейшем привлечет к сайту посетителей. В некоторых каталогах ссылки публикуются сразу, а в других — после предварительной проверки администратором. В зависимости от скорости этой процедуры регистрация ресурса может занять 1–3 недели с момента заполнения формы.

Каталоги — это едва ли не самый распространенный источник информации в сети, поэтому существуют специальные сервисы и даже программы, которые регистрируют ресурс сразу в нескольких (иногда до 1000 и более) каталогах. Правда, при таком подходе правильность прохождения регистрации, то есть занесения ссылки в каталог, не гарантируется. Качественные сервисы такого рода обычно платные, но есть также бесплатные и условно-бесплатные сервисы, предоставляющие пробную регистрацию в ограниченном количестве каталогов (обычно около десятка).

Очевидно, оптимальным методом регистрации нового ресурса является размещение ссылки в самых известных каталогах вручную, без использования автоматических регистрирующих сервисов, с тем чтобы в дальнейшем добавлять ссылки в более мелкие каталоги по мере роста и расширения ресурса. Если вы периодически осматриваете сеть, то должны регулярно встречать новые каталоги и коллекции ссылок. Если у вас есть время, то можете в них зарегистрироваться. Но не стоит делать это специально при «раскрутке» нового проекта.

Где же следует зарегистрировать новый, только что созданный ресурс? Ниже перечисляются наиболее значимые в русскоязычном Интернете (Рунете) службы каталогов.

Каталог Апорт (http://catalog.aport.ru/rus/thememap.aspx) объединяется с одноименной поисковой машиной и рейтингом. Его посещают более ста тысяч человек ежедневно. Однако большинство пользователей интересуются вовсе не каталогом, а такой информацией, как погода, программа телевидения, новости и другие сервисы, предоставляемые сервером, следовательно, это не самый популярный каталог Рунета.

Как уже отмечалось, при каталоге Апорт есть рейтинг. Кроме того, имеется несколько полезных для веб-мастеров сервисов, в том числе лента новостей сайтов, которая, безусловно, очень привлекательна при проведении маркетинговой кампании. Форма для регистрации в каталоге находится по адресу http://www.aport.ru/addurl.php, она показана на рис. 10.3.

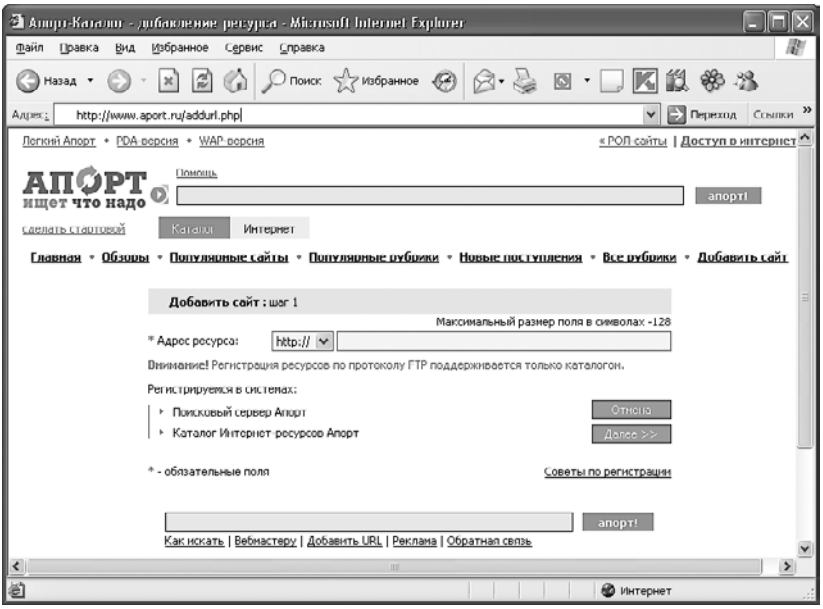


Рис. 10.3. Регистрация в каталоге Апорт

Каталог Mail.ru (http://list.mail.ru/index.html) — один из ведущих каталогов русскоязычного Интернета (рис. 10.4).

В каталоге постоянно открываются новые сервисы, полезные для веб-мастеров, поэтому рекомендуется посетить его не только для регистрации. Внимательно изучив каталог, вы найдете для себя много полезного. Сам каталог постоянно модерируется, поэтому имеет смысл указывать для регистрации только реально соответствующие категории. Если ваш ресурс действительно интересен, то он может попасть в колонку, которую ведет гид вашей рубрики, и в «листовку» — новостной ресурс при каталоге, что существенно увеличит поток посетителей.

Для регистрации в каталоге нужно выбрать соответствующую категорию и зарегистрировать в ней сайт. Предварительно внимательно прочтите инструкцию, которая находится по адресу http://list.mail.ru/registration/reg-advice.html.

В каталоге Russia on the Net (http://www.ru/) (рис. 10.5) имеются русская и английская части, причем ссылки помещаются в каждую из них отдельно. Это значит, что если у вас есть англоязычная версия сайта, то придется регистрировать ресурс дважды.

Ссылки в каталог вставляются вручную, причем делается это крайне медленно. Тем не менее каталог Russia on the Net — известный и посещаемый каталог

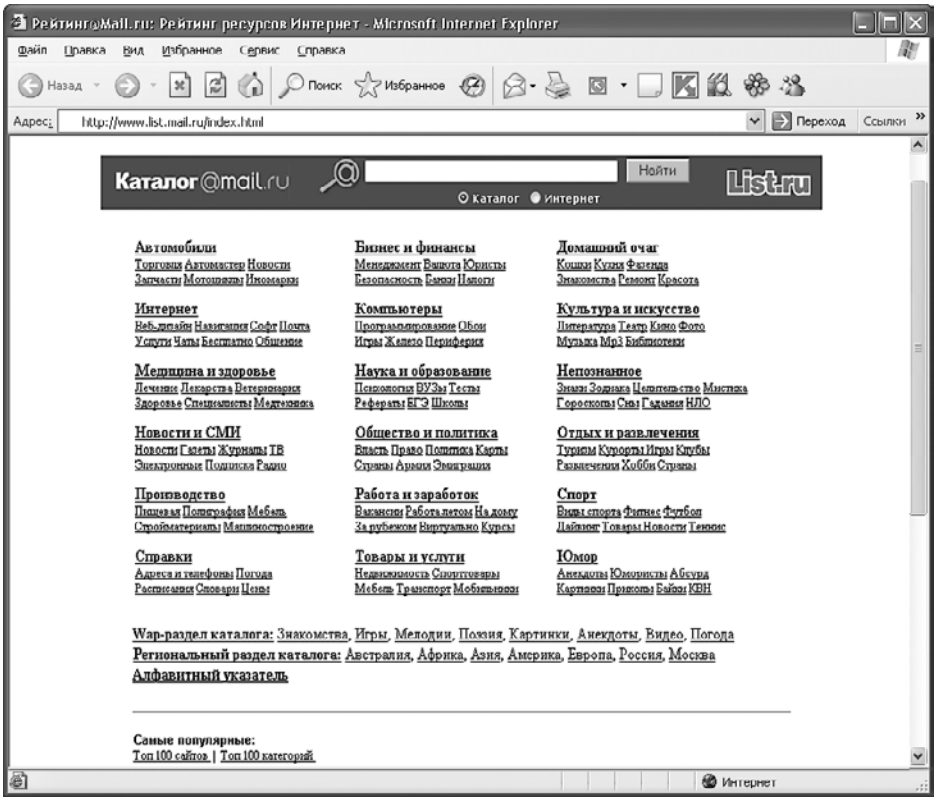


Рис. 10.4. Каталог Mail.ru

в Рунете, потому прописать в нем свой сайт нужно обязательно. Форма для регистрации в каталоге находится по адресу http://www.ru/rus/index.add.php4.

Каталог UP.ru (http://www.up.ru/) (рис. 10.6) очень активно развивается, обеспечивая приток посетителей за счет постоянного запуска новых проектов. Форма для регистрации для этого сервиса находится по адресу http://top.rate.ru/register.html.

Каталог Chat.ru (http://catalog.chat.ru/) тоже достаточно популярен. Форма для регистрации в нем находится по адресу http://www.chat.ru/rulez.html.

Рекомендуется зарегистрироваться также в следующих каталогах:

- Кирилл и Мефодий (http://go.km.ru/);
- УЛИТКА (http://www.ulitka.ru/);
- Иван Сусанин (http://www.susanin.net/);
- Data.ru (http://www.data.ru/).

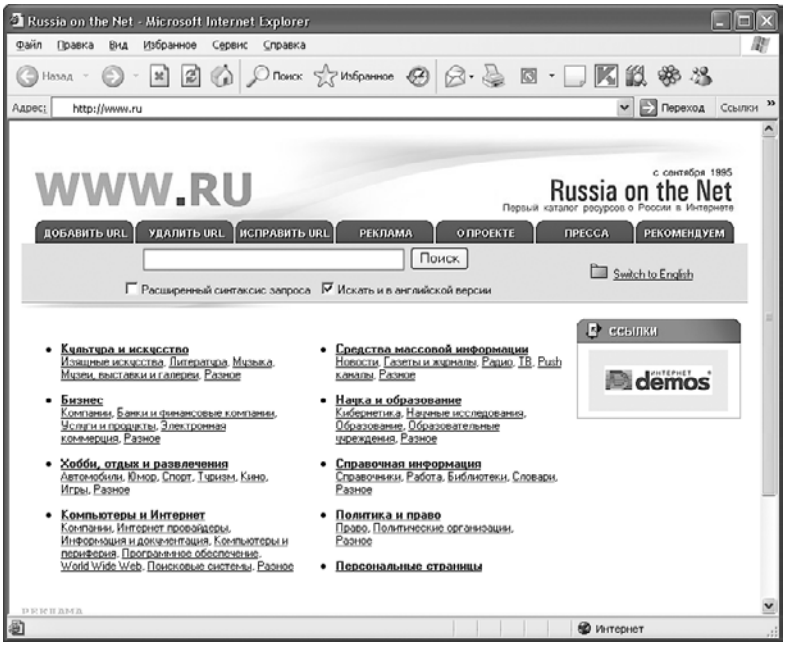


Рис. 10.5. Каталог Russia on the Net

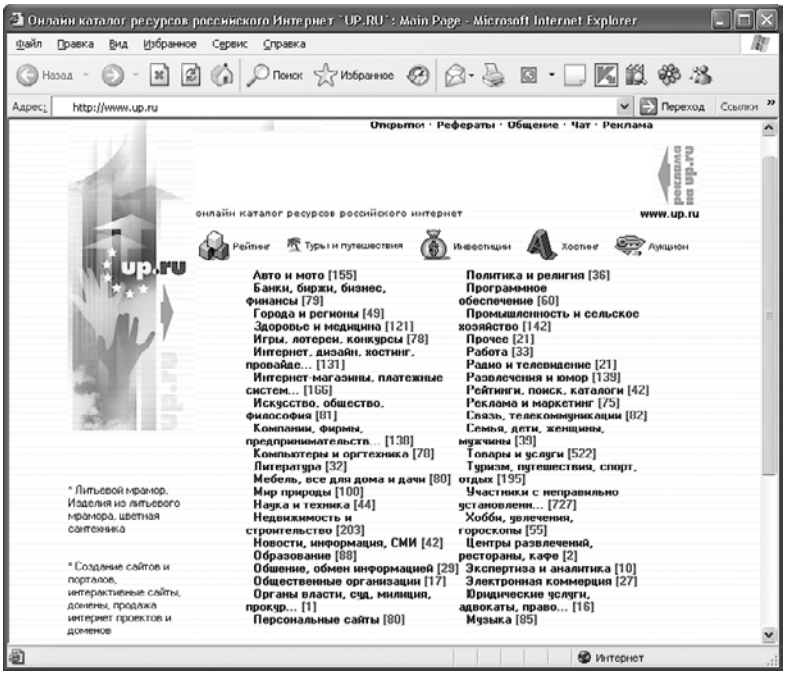


Рис. 10.6. Каталог Up.ru

Обращаю ваше внимание на еще один важный момент. Многие каталоги предлагают поместить кнопку на главной странице ресурса, обещая за это разные бонусы. Это выгодно и вам, и каталогу. Речь идет в первую очередь о крупных каталогах. Кнопку каталога нужно поставить обязательно, если ваш ресурс участвует в рейтинге. В этом случае кнопка является также и счетчиком. После регистрации вы получите индивидуальный код кнопки, который вам необходимо разместить на своем сайте.

Не менее важным для продвижения и популяризации сайта, чем регистрация в каталогах, является регистрация в рейтинговых системах. Об этом и пойдет речь в следующем разделе.

Рейтинги

В отличие от каталогов, где ссылки на веб-сайты отбираются по тематике и могут быть отсортированы по алфавиту, в рейтинге страницы располагаются по посещаемости, популярности и рейтингу. Чаще всего критерием оценки является посещаемость сайта как наиболее объективная оценка качества ресурса. Таким образом устроены рейтинги Рамблер Топ 100, Mail.ru, SpyLOG и многие другие.

Однако существуют и другие рейтинги, в которых сайты сортируются не по посещаемости, а по другим критериям, которые заранее определить сложнее. Так, например, сайты могут располагаться в порядке убывания зрительских симпатий. На каждом сайте-участнике помещается форма для голосования. Существуют также рейтинги, которые могут отражать точку зрения одного человека или некие параметры сайта, например его техническую оснащенность. Тем не менее механизм подсчета в таких рейтингах сложнее, а результаты менее достоверны. Они требуют большего внимания и, что гораздо важнее, большей активности посетителей. Поэтому рейтинги, в основе которых лежит не посещаемость, а какой-либо другой критерий, менее распространены в Интернете.

Рейтинг имеет колоссальное значение при продвижении нового ресурса. Именно для нового, еще не устоявшегося сайта рейтинг является зримым и весомым показателем успеха его владельца, свидетельствующим о его умениях и недаром затраченном упорном труде. Если рейтинг выбран правильно, то он окажется далеко не последним источником трафика для сайта: чем выше позиция вашего сайта в рейтинге, тем больше посетителей придет к вам с него и, следовательно, еще выше поднимется позиция. Для некоторых рейтингов отдача может составить весьма весомую цифру.

После регистрации в рейтинге вы получаете и устанавливаете на своем сайте счетчик, который будет автоматически подсчитывать количество посетителей. Важно, что помимо значения собственно трафика рейтинг является сторонним аудитором, оценивающим рейтинг сайта. Будучи независимым «счетоводом», он может использоваться как показатель посещаемости сайта, а следовательно, как определитель стоимости рекламы на этом ресурсе.

В рейтинге, по большому счету, нужно участвовать только в том случае, если вы твердо рассчитываете оказаться среди его лидеров. Участие ради участия не пойдет на пользу — такая позиция вредит рейтингу. Обилие ресурсов второго ранга снижает его ценность, подрывает вашу волю к победе, а главное, практически не приносит пользы. Если ваш сайт находится дальше второй страницы рейтинга, это почти так же плохо, как если бы он вообще в этом рейтинге не участвовал. Статистика свидетельствует о том, что рубрику Rambler начиная с 60 места посещает не более двух человек в сутки, что не стоит и трудов. Таким образом, если вы не можете и тем более не надеетесь пробиться в рейтинге на первые места, то не тратьте на это время.

Рассмотрим основные на сегодня рейтинги.

Самый популярный рейтинг в Рунете — **Рамблер Топ 100** (http://www.rambler.ru/). Этот рейтинг обычно используется как внешний аудитор посещаемости сайта при расчете с рекламодателями или при обмене ссылками. Такая система внешнего аудита возможна благодаря открытой статистике, которая составляется автоматически, а также известной строгости рейтинга в подсчете сайтов. Следовательно, рейтингу Rambler можно доверять.

Rambler имеет свыше 50 рубрик, в каждой из которых ресурсы сортируются по посещаемости и выкладываются страницами по 20 сайтов в каждой. Таким образом, на каждый момент времени свыше 1000 сайтов Рунета занимают лидирующие позиции в рейтинге и получают от него реальный приток посетителей — обычно не менее 50 человек в день. По умолчанию учет ведется по уникальным посетителям, то есть повторные визиты одного и того же пользователя не засчитываются. Уникальность определяется в пределах одного дня. Положения сайтов пересчитываются каждые 30 минут.

Рейтинги не имеют описаний, поэтому важно, чтобы название, которое вы вписываете при регистрации, соответствовало теме, было интересным и легко запоминающимся. Попытайтесь не использовать английских названий, так как у русскоязычной аудитории они вызывают неосознанное отторжение и воспринимаются значительно медленнее. Постарайтесь не использовать названия длиннее 20 сим-

волов, так как длина строки ограничена и перенос на следующую строку выглядит небрежно и разбивает визуальный образ. Это же относится и к заглавным буквам в названии.

Тщательно выбирайте рубрику, так как при ее несоответствии содержанию сайта посетителей вы соберете, а постоянную аудиторию, то есть людей, заинтересованных в вашем ресурсе, — нет. Если тематике вашего сайта соответствуют сразу несколько рубрик, то внимательно изучите каждую из них. Проанализируйте, в какой из рубрик встречается больше сайтов вашей тематики, где располагаются ссылки ваших конкурентов, на каком месте будет ваш сайт в этой рубрике, какова средняя посещаемость сайтов на первой странице в этом разделе и т. д. Регистрировать сайт лучше на той рубрике, на которой находятся сайты ваших основных конкурентов, так как именно там будут искать материалы по тематике вашего сайта посетители. Если и в этом случае перед вами стоит неоднозначный выбор, то выбирайте рубрику, средняя посещаемость которой больше соответствует вашему сайту.

Избегайте регистрации в рейтинге не полностью готовых сайтов только для того, чтобы набрать количество посетителей на счетчике. Многие из них запомнят ваш сайт как недоделанный, некачественный продукт, и в дальнейшем вам будет трудно привлечь этих посетителей обратно, когда работа над сайтом полностью завершится.

Чтобы участвовать в рейтинге, необходимо поместить на главной странице сайта счетчик посещаемости, полученный после регистрации. Есть несколько вариантов счетчиков: синий квадрат размером 88 × 88 точек с цифрами или без них и стандартные кнопки размером 88 × 31 точек разных расцветок с логотипом Rambler. В последнем варианте счетчик помещается в произвольном месте страницы. Вообще, его лучше помещать в самом верху страницы, что несколько улучшит эффективность работы счетчика. Счетчик Rambler может быть помещен лишь на одной странице и только один раз. Форма для регистрации в рейтинге Rambler, показанная на рис. 10.7, находится адресу http://www.rambler.ru/doc/add_site.shtml.

Рейтинг **Mail.ru** (http://top.mail.ru/) (рис. 10.8) пользуется большой популярностью, хотя конкурировать с Rambler ему пока не по силам. И каталог, и рейтинг привлекают ежедневно немало посетителей, поэтому не стоит пренебрегать возможностью их использования.

Рейтинг содержит несколько десятков разделов. Ссылки на сайты даются без описания. Приводятся данные по общему количеству посещений, количеству уникальных посещений, а также увеличению или уменьшению посещаемости (в процентах). Кроме того, счетчик рейтинга предоставляет довольно широкую

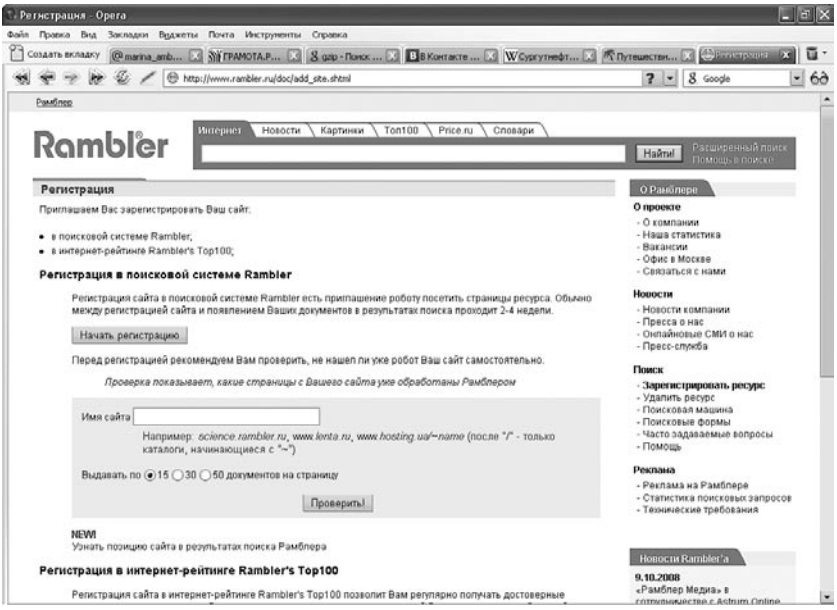


Рис. 10.7. Регистрация в рейтинге Rambler

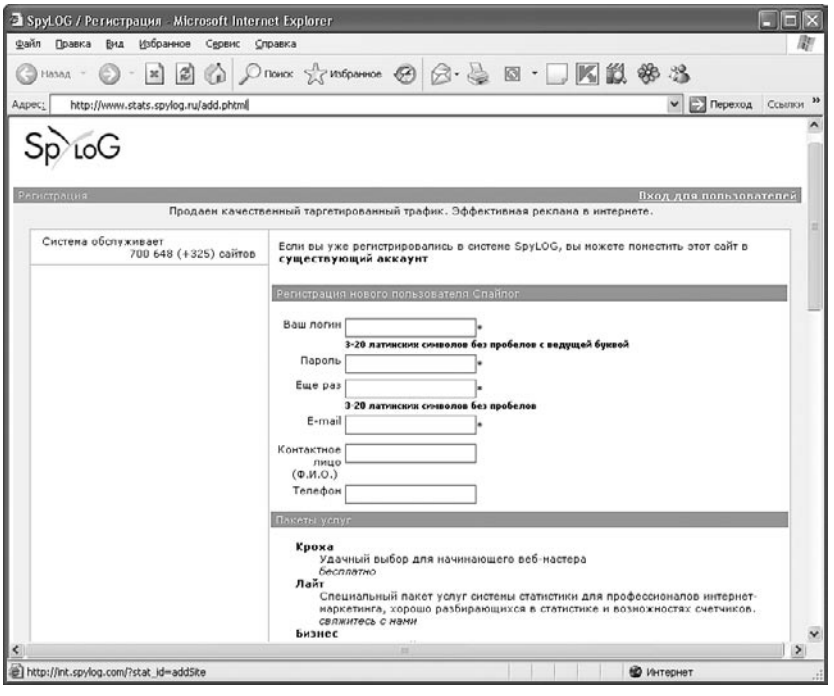


Рис. 10.8. Рейтинг Mail.ru

статистику посещаемости, в том числе и по ссылающимся страницам. Для использования всех возможностей вам придется установить полный код счетчика, включающий Java-сценарий (на это потребуется около 1,5 Кбайт, не считая размера картинки). В этом случае рекомендуется запретить неавторизированный доступ к данным, так как статистика по ссылкам является очень ценной информацией для конкурентов. Однако данные счетчика не всегда достоверны, поэтому его, в отличие от счетчика Rambler, рискованно использовать в качестве внешнего аудитора. Форма для регистрации в рейтинге находится по адресу http://top.mail.ru/add.

Рейтинг **SpyLOG** (http://spylog.ru/), так же как и предыдущие рейтинги, организуется на базе статистического счетчика. Однако у этого сервиса имеются серьезные отличия от предыдущего рейтинга. Предоставляемая сервисом статистика впечатляет: многостраничная обработка результатов, все стандартные данные, статистика программного и аппаратного обеспечений пользователей, статистика распределения по времени, маршруты по сайту и др. Форма для регистрации в рейтинге находится по адресу http://stats.spylog.ru/add.phtml (рис. 10.9).

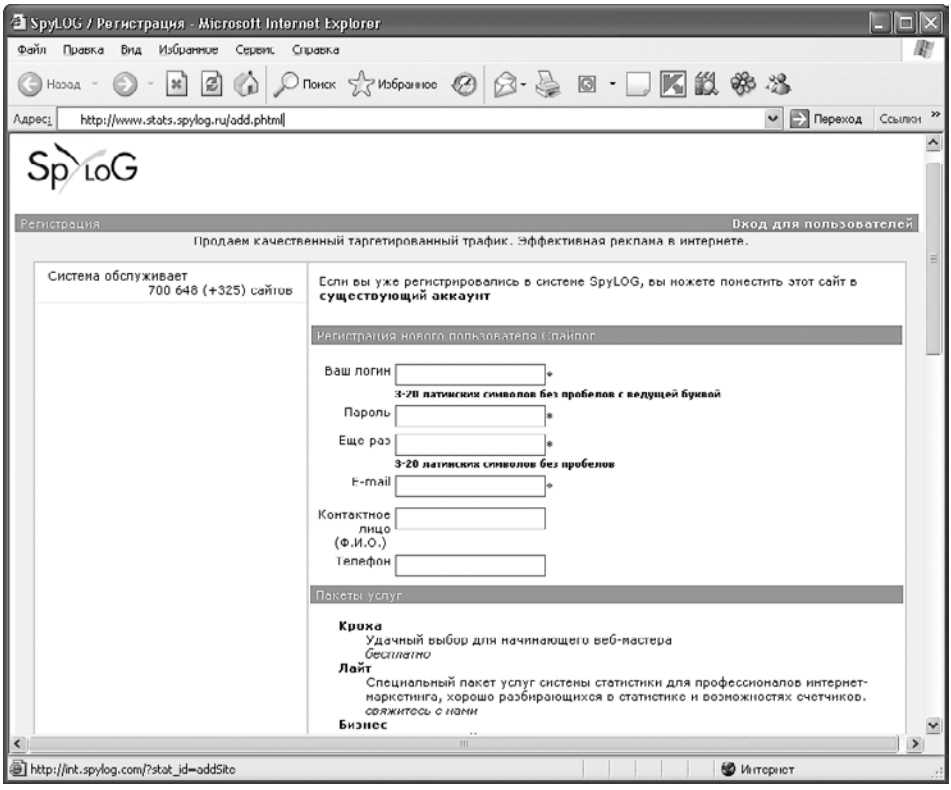


Рис. 10.9. Регистрация в рейтинге SpyLOG

В заключение дам вам несколько советов.

- Старайтесь размещать счетчик как можно выше на странице, чтобы он всегда успевал загрузиться. Особенно это касается счетчиков тех рейтингов, которые позволяют установить невидимую картинку; в этом случае ее можно поместить в верхней части документа без ущерба для внешнего вида страницы.
- Не стоит устанавливать на одной странице несколько явных счетчиков. На самом деле это представляет собой серьезную проблему. С одной стороны, чем в большем количестве рейтингов вы участвуете, тем больше посетителей вы привлечете, с другой же стороны, все рейтинги ведут подсчет результатов по-разному. В итоге уже через несколько дней после регистрации вы обнаружите, что все счетчики показывают разные цифры. Это производит на нового посетителя очень неприятное впечатление беспорядка. К счастью, сейчас практически все рейтинги предоставляют возможность размещения «слепого» счетчика, то есть счетчика, не имеющего цифр. Желательно выбрать самый эстетичный на ваш взгляд счетчик, а остальные скрыть.
- Регистрируясь в разных рейтингах, варьируйте описание или название своего сайта в зависимости от аудитории рейтинга.

Мы уже говорили в предыдущих главах о баннерах, вы также учились их создавать. В следующем разделе речь пойдет об использовании баннеров в целях популяризации вашего ресурса и об участии в баннерном обмене.

Баннеры

Еще один популярный коммерческий метод продвижения сайтов — *баннерная реклама*. В этом направлении есть свои плюсы и минусы. Особенностей, секретов и различных хитростей здесь огромное количество. Хотя баннеры — далеко не единственный рекламный носитель в Интернете, но все же главный: до 57 % рекламы в нем приходится на баннеры. Несмотря на то что их эффективность постоянно падает, они позволяют использовать ваш сайт для вашей же рекламы. Поэтому при разработке маркетингового плана баннерной рекламе стоит уделить особое внимание.

Баннерообменная сеть — это служба, которая помогает всем желающим размещать баннеры на других сайтах.

Система обмена баннерами работает следующим образом. Вы, как владелец сайта, регистрируете его в системе, загружаете свои баннеры и взамен получаете код, который вставляете на свои страницы. При просмотре страниц вашего сайта в том

месте, куда вы вставили код, посетитель увидит баннер одного из участников баннерной сети. Система регистрирует каждый показ чужого баннера на ваших страницах и суммирует их общее количество, с учетом которого баннеры, рекламирующие ваш сайт, появляются на сайтах других участников.

Баннерообменная сеть за свои услуги взимает определенную комиссию. Например, вы показываете на своих страницах 100 чужих баннеров, а сеть взамен показывает 90 ваших баннеров на других сайтах. Десять показов баннеров из ста сеть забирает себе — для продажи желающим или для рекламы самой баннерной службы. Как правило, чем крупнее сеть, тем более высок процент ее комиссии. Есть сети, которые обменивают показы баннеров «один к одному». Обычно это небольшие или только начинающие работать сети.

Большинство баннерообменных сетей предоставляет своим пользователям подробную статистику о количестве показов чужих баннеров на странице пользователя, количестве показов собственных баннеров на чужих сайтах, показатель CTR каждого баннера и другие сведения. Некоторые баннерообменные сети предоставляют возможность своим участникам самим выбирать, на сайтах каких тематических групп будут показываться баннеры участника, позволяют запретить показ на своих страницах баннеров определенных сайтов (например, если эти сайты являются конкурентами вашего), а также показывают свои баннеры только в определенные дни недели, время и для посетителей какого-либо региона страны. В некоторых баннерных службах услуги настройки показов на определенную аудиторию предоставляются не всем участникам, а только тем, которые показывают на своих страницах оговоренное число баннеров других участников.

Почти все крупные баннерные службы разрешают управлять показом баннеров, что позволяет участникам сети создавать гибкие схемы для своих баннерных рекламных кампаний. Например, вы, как владелец сайта, можете, показывая баннеры других участников, запретить показы своих, чтобы накопить их. Когда в сети накопится большое количество показов баннеров, можно будет разрешить их демонстрацию. В этом случае некоторое время на сайтах, входящих в данную сеть, будут преобладать ваши баннеры, что положительно скажется на притоке ваших посетителей. Такой вариант массированной рекламной кампании рекомендуется для сайтов с изначально невысокой посещаемостью. Если бы баннеры этих сайтов действовали по обычной схеме, то они просто потонули бы в массе баннеров других участников сети.

Как вы уже знаете, баннер — это прямоугольный графический объект, обычно формата GIF или JPEG, расположенный на веб-странице и связанный гиперссылкой с другим сайтом. Теоретически баннер может быть любого формата, лишь бы

он умещался на странице. Как мы уже говорили в главе «Графика и дизайн», существует несколько стандартных размеров, которые наиболее распространены и используются при обмене. Самые распространенные форматы — 468 × 60, 400 × 50, 120 × 60, 100 × 100 и 88 × 31 пиксел. Баннер может быть как анимированным, так и статическим.

Баннер должен загружаться на страницу как можно быстрее, поэтому его размер ограничивается (в килобайтах). Так, например, для баннера размером 468 × 60 пикселов максимальный размер не должен превышать 10–15 Кбайт.

Вы можете пользоваться баннерной рекламой платно и бесплатно. Возможно также их сочетание. При бесплатной баннерной рекламе, чаще называемой баннерным обменом, вы получаете от других участников рекламу, объем которой пропорционален тому объему рекламы, который вы предоставили на своем сайте (обычно учитывается количество показов). Таким образом, чем выше посещаемость сайта, тем больше он рекламируется в сети баннерного обмена.

Эффективность баннерного обмена характеризуется откликом. *Отклик* — это отношение количества щелчков кнопкой мыши на баннере к количеству просмотров. По статистике, существующей на сегодня, показываемые на сайте баннеры дают отклик (CTR — Click True Ratio) от 0,2 до 3,5 %, причем средний CTR равен 0,7 %. Это значит, что при отклике 1 % на 100 просмотров баннера приходится один щелчок кнопкой мыши на баннер.

Что можно получить от баннерного обмена при таком отклике? При условии, что вы помещаете баннеры на всех страницах своего сайта, среднее количество просмотров — три страницы за сеанс. Другими словами, посетители в среднем просматривают по три страницы за посещение — это вполне реальная цифра. Средний отклик — 1 %. В таком случае за счет участия в баннерном обмене вы получите около 3 % новых посетителей. Конечно, это небольшая цифра, но все же — некоторая «прибавка». Таким образом, баннерный обмен — это не тот метод, на который можно делать ставку при продвижении нового ресурса, но пренебрегать им не следует.

С другой стороны, при покупке баннерной рекламы вы не привязаны к существующему на вашем сайте трафику. Вы можете купить любое количество баннерных показов, если у вас достаточно для этого денег. Отмечу, что стоимость баннерной рекламы, особенно на вторичном рынке, очень невысока. Тогда даже при невысоком отклике, который предоставляет баннерная реклама, стоимость всей кампании останется невысокой. Покупка баннерной рекламы может стать серьезной предпосылкой повышения популярности вашего проекта. При этом следует понимать, что речь может идти о покупке сотен тысяч и даже миллионов баннерных показов.

Выбирая баннерообменную сеть, необходимо внимательно ознакомиться с условиями вступления в нее, количеством и профессиональным уровнем сайтов-участников, размерами комиссии, правилами сети и оценить, насколько все это соответствует тематике, содержанию и стилю вашего сайта. Например, для некоторых сайтов будет только плюсом, что баннерная сеть строго следит за содержанием сайтов-участников и демонстрируемых баннеров, для других же сайтов это может стать сдерживающим фактором. Поэтому при выборе баннерной сети вам самим придется решить, насколько приемлемы для вашего сайта ее условия. Информация о правилах и условиях вступления в баннерную сеть обычно подробно представлена в специальном разделе на сайте сети.

В Рунете на момент написания этой книги имеется несколько крупных баннерных операторов. Рассмотрим некоторых из них.

Сеть Russian Link Exchange (http://www.rle.ru) (рис. 10.10) охватывает более 20 000 сайтов различной тематики, ежедневно показывает более 19 млн баннеров размером 468 × 60 пикселов. Принимаются все сайты, за исключением сайтов с содержанием «для взрослых» и посвященных заработку в сети на основе финансовых пирамид. Эта баннерная сеть берет комиссию в размере от 14 до 33 %. Чем больше чужих баннеров на своих страницах вы сможете показать, тем меньшую комиссию взимает сеть. Сеть строго следит за содержанием баннеров и соответствием сайтов заявленной тематике. Выбирать аудиторию показа своих баннеров могут только участники сети, показывающие на своих страницах более 400 чужих баннеров в сутки.

Сеть RB2 (http://www.rb2.design.ru) представляет собой баннерную систему, работающую с баннерами формата 100 × 100 пикселов. Сеть ежедневно показывает более четырех миллионов баннеров. В сеть RB2 принимаются не все сайты, а только те из них, которые характеризуются достаточно высоким уровнем оформления и содержания. Для вступления в сеть необходимо подать заявку, после чего работник сети осмотрит ваш ресурс и решит, принимается он в сеть или нет. RB2 приветствует новостные ресурсы, серьезные специализированные сайты различных тематик и сайты культурных проектов. Каждому новому участнику даются «подъемные» в размере 20 000 показов. Все участники имеют возможность настраивать аудиторию показа своих баннеров — по времени, географии и зрителям.

В сеть ИнтерРеклама (http://www.interreklama.ru) принимаются все сайты, тематика которых не запрещается законом. Баннерная служба взимает комиссию в размере 10 %. Сеть строго следит за соответствием сайта заявленной тематике, а баннеры проверяются модератором. Есть возможность тонкой настройки рекламной кампании с применением собственных баннеров. В сеть принимаются баннеры размером 468 × 60 пикселов.

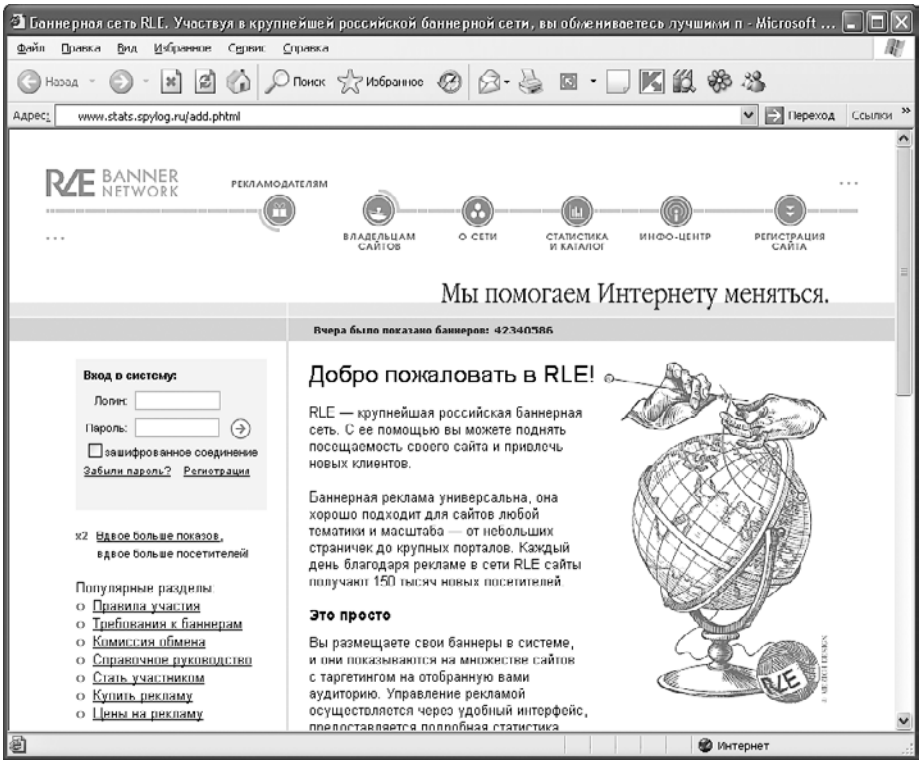


Рис. 10.10. Баннерная сеть Russian Link Exchange

Адреса других баннерных служб вы можете найти самостоятельно, например по адресу http://webrating.ru.

Каждая баннерная сеть с удовольствием продаст вам практически любое количество баннерных показов, если вы решите организовать баннерную кампанию для своего сайта. В отличие от обмена, где они занимают разные ниши, в борьбе за платного клиента эти сети жестко конкурируют, поэтому цены на показы регулярно падают.

Как отмечалось выше, сети оперируют несколькими распространенными форматами баннеров, причем основной среди них — 468 × 60 пикселов. Но, в принципе, если вы не обмениваетесь рекламой, а продаете ее, то можете использовать баннеры любого формата. Баннеры большого формата обычно имеют более высокий отклик, поэтому при проведении широкомасштабной рекламной акции имеет смысл ориентироваться именно на них. Широко распространен также формат 88 × 31 пиксел, которым обладают кнопки. Они характеризуются самым низким откликом, но и практически не занимают места. Обычно этот формат используется для локаль-

ного обмена между веб-мастерами. Существуют также и системы обмена кнопками, но они не пользуются большой популярностью. Следует отметить, что большинство из них находится не в русскоязычном Интернете.

Мы рассмотрели основные способы «раскрутки» нового сайта. В следующем разделе я вкратце расскажу еще о некоторых элементах привлечения посетителей.

Элементы привлечения посетителей

Мощным стимулом посещаемости сайта является его интерактивность, благодаря которой сайт перестает быть статичным набором страниц и становится живым, гибким организмом, не только плодом вашей работы, но и предметом предпочтений вашей аудитории.

Простейшими элементами, обеспечивающими интерактивность сайта, являются гиперссылки. Мгновенный доступ к любому связанному документу — это одна из основ Всемирной паутины, фундамент ее интерактивных возможностей. Не следует пренебрегать гиперссылками, сайт не должен объединяться в единое целое только лишь оглавлением. Не забывайте связывать между собой ключевые понятия внутри текстовых страниц с соответствующими описаниями на вашем или других сайтах. Организуйте перекрестные ссылки на смежные материалы по страницам своего сайта, чтобы пользователю было легче в нем ориентироваться. Но соблюдайте меру: сайт, каждое слово которого представляет собой ссылку на связанные материалы, читать очень трудно.

Интерактивность создается целым комплексом средств, часть которых реализуется сложными программными механизмами, а другие доступны даже новичкам. Ввиду дороговизны и сложности реализации продвинутых механизмов здесь мы рассмотрим только самые простые методы создания интерактивного сайта. Одним из самых распространенных методов реализации любой интерактивной функции является использование CGI-сценариев.



ПРИМЕЧАНИЕ

CGI-сценарий — это программа, которая выполняется на веб-сервере по запросу посетителя сайта. Принципиально она ничем не отличается от обычных программ, установленных на вашем компьютере. CGI (Common Gateway Interface) — это специальный интерфейс, с помощью которого происходит запуск сценария и взаимодействие с ним.

Интерактивность — это в первую очередь возможность высказаться, которую необходимо предоставить посетителю. Для этого предназначаются возвратные

формы, адрес вашей электронной почты, номер вашего телефона (для коммерческого, торгового сайта номер телефона нужно обязательно указывать). С помощью этих элементов посетитель может высказать свое мнение о тематике сайта, его работе и о самом веб-мастере.

Возвратная форма — это специальная форма, которая позволяет пользователю написать письмо, не открывая почтовой программы и, более того, не имея собственного адреса электронной почты. Это особенно удобно для тех посетителей, которые не хотят указывать свой адрес, или же для тех, которые пользуются адресом на бесплатном сервере, не поддерживающем доступ из почтовой программы.

Если же пользователь желает высказаться публично, то сделать это он может с помощью гостевых книг и форумов, которые вы, без сомнения, регулярно встречаете, путешествуя по Интернету.

Гостевые книги

Гостевые книги организуются сравнительно простыми CGI-сценариями. Архивы этих и других сценариев вы найдете на сайте http://www.download.ru/. Если вы не хотите самостоятельно устанавливать сценарии, то можете воспользоваться специальными сервисами, которые предоставляют уже инсталлированные и настроенные сценарии в личное пользование за рекламу небольшого объема на ваших страницах. В этом случае вам достаточно будет просто поставить ссылку на сценарий, работающий на стороннем сервере. Гостевые книги пользуются неизменной популярностью, поскольку обладают очень полезными свойствами: во-первых, они чрезвычайно просты в установке и администрировании; во-вторых, записи посетителей не всегда требуют ответа, что немаловажно. Создать собственную гостевую книгу можно с помощью таких бесплатных служб, как GuestBook.ru (http://www.guestbook.ru/) и Chat.ru (http://www.chat.ru/).

Форумы

В отличие от гостевых книг, *форум*, или *конференция*, — это сложная система обмена мнениями, требующая повышенного внимания со стороны своего создателя. Существуют программы для ведения форумов, которые поддерживают сразу несколько обсуждаемых тематик различных форумов, поиск по сообщениям, а также авторизацию пользователей. Несомненно, форум является оптимальным механизмом создания интерактивности больших и средних информационных ресурсов, поскольку при небольшой нагрузке на издателя он позволяет сформировать

настоящую среду для общения посетителей. Работа с форумом относится больше к маркетинговым акциям по созданию сообщества вокруг сайта.

Планируя использование форума на сайте с помощью собственного, установленного на вашем сервере сценария, проверьте объем доступного вам дискового пространства. Сценарий для форума, так же как и для гостевой книги, не является ресурсоемкой программой: он занимает всего несколько десятков или сотен килобайт. Но при работе сценарий создает файлы с текстами сообщений, которые будут занимать некоторое пространство. Если у вас не очень много посетителей и на форуме появляется несколько десятков сообщений в неделю, то вам не стоит беспокоиться. Однако при трафике в полсотни сообщений в день и более стоит подумать о том, где все эти сообщения будут храниться. Таким образом, для средних форумов требуется до 2–3 Мбайт свободного дискового пространства.

Для создания бесплатных форумов персональных веб-сайтов вы можете также воспользоваться услугами следующих российских служб:

- WebForum.ru (http://www.webforum.ru);
- Форумы @mail.ru (http://talk.mail.ru).

Чаты

Следующий вариант интерактивного общения в режиме реального времени — чат. С технической точки зрения чат обычно организуется с помощью CGI-сценария на собственном сервере или арендуется, как и предыдущие сценарии, в специальных сервисах. В отличие от форума, чат является значительно более ресурсоемкой программой, поскольку объем сообщений в нем значительно больше. Для чата необходимо несколько десятков мегабайт дискового пространства и быстрый процессор на сервере для обработки потока поступающих данных. Следует обсудить с администратором вашего сервера возможность использования чата до того, как вы начнете его устанавливать, так как не все серверы способны поддерживать чаты.

Вы можете воспользоваться услугами специальных сервисов для создания чатов на своем сайте. Вот адреса некоторых российских служб, которые предлагают услуги бесплатных чатов для персональных веб-сайтов:

- Chat.ru (http://www.chat.ru);
- chats.RU (http://www.chats.ru);
- Яндекс. Народ (http://www.narod.ru).

И, наконец, напомню, что интерактивность подразумевает связь между посетителями, самим сайтом и издателем. Эту связь необходимо постоянно поддерживать, так как сама она развиваться не будет. И в этой поддержке вы или специально назначенный вашей фирмой сотрудник должны быть самым активным участником, в частности, потому, что это в ваших же интересах.

Почтовые рассылки

Одним из наиболее популярных маркетинговых мероприятий, цель которого — создание и расширение постоянной аудитории, является почтовая рассылка. Среди ваших посетителей есть много людей, которые заходят на ваш сайт регулярно по разным причинам. Но есть и те, кому интересен именно ваш ресурс и разрабатываемая вами тематика, но у них нет времени часто заглядывать к вам с надеждой, что на сайте появилось что-то новое. А есть и еще одна категория людей, которым интересна тема сайта, но не слишком интересен сам ресурс. И тем, и другим, и третьим посетителям придется по вкусу регулярная почтовая рассылка, издаваемая вами. В этом случае, если у человека недостаточно времени для частого посещения вашего сайта, он сможет заходить на него только за интересующей его информацией. Что же касается остальных пользователей, то им по крайней мере будет приятно своевременно узнавать о новостях вашего сайта.

Очевидно, что для почтовой рассылки вам нужна база данных с адресами, по которым вы будете проводить рассылку, то есть вам нужны подписчики. Теоретически адреса можно купить, не спрашивая на то разрешения владельцев почтовых ящиков. Однако в этом случае ваши действия квалифицируются как спам (принудительная рассылка рекламы без разрешения получателя) и во многих странах являются незаконными. Конечно же, спам неприятен пользователям. Поэтому мы рассмотрим частный случай почтовой рассылки, а именно рассылку по подписке, при которой пользователи сами подписываются на получение нужной им информации.

Письма можно рассылать автоматически и вручную. Однако в последнем случае это становится очень неудобным, как только количество подписчиков приближается к сотне. Для автоматизации процесса рассылки существуют специальные сервисы. Вы можете также установить для рассылки CGI-сценарий или воспользоваться специальной программой. И удаленные сервисы, и сценарии на сервере, и даже специальные программы обладают примерно одинаковыми возможностями: автоматическая подписка и отписка, подтверждение подписки, выпуск в разных кодировках и др. Чтобы ваши посетители смогли подписаться на рассылку, вы

должны установить на своем сайте небольшую форму, причем желательно на каждой странице.

Более сложной рассылкой, требующей больших затрат энергии со стороны издателя, является регулярная рассылка, например еженедельная. Чтобы такая рассылка была интересна читателям, в ней должны публиковаться не только собственно новости сайта, но и тематические новости, различные интересные материалы, заметки, советы и т. д. Несомненно, регулярная рассылка должна быть интересна сама по себе, а не только как приложение к сайту. Естественно, что в нее вы также включаете и собственные новости, тем самым привлекая посетителей на свой ресурс.

В последнем случае подписка превращается в ваших руках в мощный маркетинговый инструмент, а также в хороший источник прибыли. Реклама в почтовых рассылках на сегодня с полным правом считается наиболее эффективной в отношении цены к качеству. Поэтому вы можете использовать рассылку для публикации платной рекламы и получать при этом большую отдачу. Однако для того чтобы привлечь рекламодателей, необходимо набрать несколько тысяч подписчиков. В противном случае рассылка, скорее всего, денег приносить не будет.

Увеличить количество посетителей сайта также можно, поместив на главной странице информацию, пользующуюся постоянным спросом, например новости, сводку погоды, курс валюты и т. д.

Ссылки

Значительно повысят популярность вашего сайта, а следовательно, и привлекут новых посетителей страницы ссылок на ресурсы с близкой или родственной тематикой. Благодаря коллекции ссылок посетители получают доступ к специализированной информации по всем областям близкой тематики, что делает ваш сайт еще более привлекательным. Он будет интересен, во-первых, как источник основной информации и, во-вторых, как полезный справочник-путеводитель по тематическим ресурсам. Как правило, грамотно составленная страница ссылок находится на втором или третьем месте по посещаемости среди всех страниц сайта.

Чтобы правильно составить страницу ссылок, просмотрите свои закладки. Вероятно, у вас скопилось достаточное количество ссылок на сайты по вашей тематике, которые вы регулярно посещаете. Это и есть основа вашей страницы ссылок. Поместите эти ссылки на одну страницу, удалите все малозначительные и случайные,

а также не относящиеся к тематике. Возможно, стоит убрать и ссылки на сайты, которые могут составить вам серьезную конкуренцию. Оставшиеся ссылки обязательно надо разбить по разделам в соответствии с их узкой тематикой. В зависимости от получившегося объема материалов количество разделов может быть различным, но лучше придерживаться следующих рамок: 10–20 ссылок в разделе и не более 80 ссылок на странице. Если ссылок больше, можно организовать несколько страниц, на каждой из которых будет находиться по одному тематическому разделу. Выделив все эти страницы в отдельную область сайта, создайте ее оглавление, задав гиперссылки непосредственно на страницы разделов.

Для каждой ссылки сделайте небольшое описание (на 200–300 символов), которое будет характеризовать не только сам ресурс, но и ваше отношение к нему. Учтите, что чем больше ссылок на странице, тем короче должны быть описания (общий объем текста вместе с тегами не должен превышать 15 Кбайт). Иногда к каждой ссылке достаточно дописать пару слов, которые будут пояснять, чему именно посвящается ресурс, поскольку теоретически пользователь ориентируется в тематике (он уже изучил ваш сайт) и способен самостоятельно определить свои библиографические потребности. Постарайтесь выделить ссылку не только цветом, но и размером шрифта, чтобы она не затерялась на фоне описания.

Теперь нужно упорядочить ссылки по какому-либо принципу, например по степени важности или исходя из ваших личных пристрастий. Крайне нежелательно сортировать ссылки по алфавиту, так как это ведет к официальности и потере информативности. Для небольшого персонального или коммерческого сайта лучше всего подойдет сортировка на основании ваших личных оценок рекомендуемых сайтов. Если же вы создаете ресурс для профессионалов, то хорошим вариантом будет сортировка по степени важности рекомендуемых ресурсов или по полноте раскрытия темы.

Положение рубрик тоже играет важную роль. Подумайте, что именно главенствует на вашем ресурсе, какая тематика интересует вас более всего, и согласно этим соображениям расположите разделы так, чтобы они наиболее полно соответствовали характеру представленной вами тематики. Теперь перед вами классифицированная страница ссылок с описаниями, которую можно смело представить посетителям.

Далее следует воспользоваться механизмом обмена ссылками: вы находите сайт, на котором хотели бы видеть ссылку на свой ресурс, договариваетесь с его вебмастером и устанавливаете ссылку на его сайт на своей странице. В ответ на сайте-партнере появляется ссылка на ваш сайт.

Конечно, всегда следует помнить о том, что, сколько бы шагов по раскрутке сайта вы ни предприняли, сайт не сможет привлечь постоянных посетителей, если сам по себе он неинтересный.

Обобщив вышеизложенный материал, можно выделить основные советы, которые помогут вам повысить популярность своего сайта.

- Подбирайте материалы для сайта с учетом личного взгляда на тему. К материалам можно добавлять собственные комментарии, особенно если вы не совсем согласны с автором.
- Добавляйте на сайт новые материалы. Старайтесь делать это регулярно, не реже одного раза в месяц. Размещайте информацию об обновлениях либо в специальном разделе новостей вашего веб-сайта, либо на главной странице.
- Предложите своим посетителям подписаться на рассылку новостей вашего сайта. Рассылки очень эффективно поддерживают интерес к веб-сайту. В регулярных рассылках сообщайте о новостях сайта, о появлении новых материалов, поздравляйте ваших подписчиков с праздниками и т. д. Не забывайте удалять из списка рассылки посетителей, которые прислали вам запрос на прекращение подписки.
- Если ваш сайт тематический, активно участвуйте в интернет-форумах и почтовых конференциях, соответствующих тематике вашего сайта. Если вы зарекомендуете себя интересным собеседником и знатоком какой-либо темы, то к вашему веб-сайту сформируется серьезное отношение со стороны пользователей.
- При написании писем в тематические форумы и почтовые конференции, а также других писем в конце письма добавляйте адрес вашего веб-сайта (там, где это уместно).
- Не забывайте отвечать на письма посетителей вашего веб-сайта, даже если они содержат критику. Зачастую критики впоследствии становятся активными помощниками владельца веб-сайта. Если кто-то сделал вам критическое замечание, то это означает, что ваш ресурс небезразличен этому посетителю и он хочет, чтобы сайт стал лучше.
- Разместите на сайте раздел с информацией о себе. Это особенно важно, если ваш сайт тематический и вы добились успехов в той области, которой посвящается сайт.

Если вы будете соблюдать все эти правила, а также предпримете все описанные в главе шаги по раскрутке сайта, популярность вашего веб-сайта обязательно должна повыситься.

Резюме

Мы рассмотрели принципы раскрутки сайта и элементы привлечения посетителей на ваш ресурс. Я рассказал, как сделать ваш сайт посещаемым, как привлечь внимание посетителей, а также как правильно и грамотно разработать политику популяризации вашего ресурса. Вы изучили такие важные элементы раскрутки ресурса, как регистрация в поисковых машинах, различных каталогах и рейтингах. Мы разобрали интерактивную связь посетителей сайта с его создателями как важный элемент работы с посетителем сайта. В качестве таких интерактивных элементов мы рассмотрели гостевые книги, форумы, почтовые рассылки и чаты.

Надеюсь, что мои советы помогут вам в продвижении своего сайта, сделают его популярным, что в дальнейшем привлечет внимание пользователей. Но не стоит забывать, что главное — это постоянная работа над сайтом, внимание со стороны создателей веб-ресурса к своему детищу, а также полезность и информативность сайта. Без этого все усилия по популяризации будут напрасными.