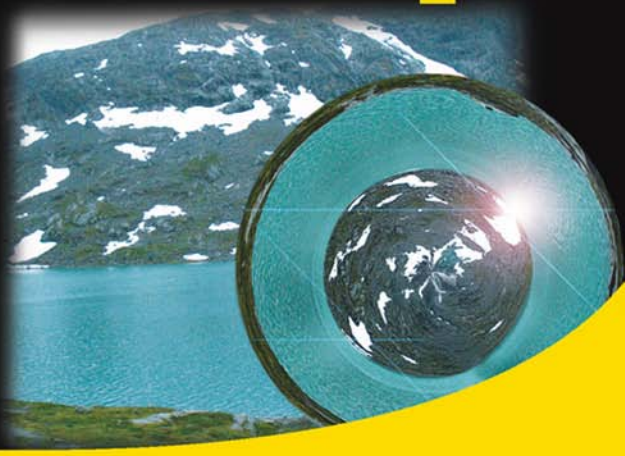
Самоучитель

Вадим Дунаев

Сценарии для Web-сайта PHP и JavaScript

2-е издание



Основы языков PHP 5 и JavaScript

Разработка клиентских и серверных сценариев

Готовые примеры

Установка и настройка PHP и Web-сервера IIS

bhv®

Вадим Дунаев

Сценарии для Web-сайта PHP и JavaScript

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2008

УДК 681.3.06
ББК 32.973.26-018.2
Д83

Дунаев В. В.

Д83 Сценарии для Web-сайта: PHP и JavaScript. — 2-е изд. перераб. и доп. — СПб.: БХВ-Петербург, 2008. — 576 с.: ил. — (Самоучитель)

ISBN 978-5-9775-0112-5

Книга посвящена использованию языков JavaScript и PHP для разработки Web-приложений. Приведены основные понятия, связанные с разработкой Web-сайта, а также сведения о языке HTML и каскадных таблицах стилей (CSS). Рассмотрены основы программирования на JavaScript и PHP 5. Приведены практические примеры различных клиентских и серверных сценариев. Описаны особенности и даны рекомендации по применению этих языков. В приложениях содержатся сведения по объектам документа и браузера, с которыми работают клиентские сценарии, а также рассказывается о том, как установить и настроить PHP и Web-сервер IIS в системе Windows. Во втором издании добавлены новые примеры.

Для начинающих Web-разработчиков

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Татьяна Лапина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 29.12.07.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 46,44.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0112-5

© Дунаев В. В., 2008
© Оформление, издательство "БХВ-Петербург", 2008

Оглавление

Введение **1**

Глава 1. Как устроен Web-сайт? **3**

1.1. Основные понятия 3
1.2. Как устроен HTML-документ? 5
1.2.1. Начальные сведения 6
1.2.2. Структура документа 9
1.2.3. Ссылки 22
1.2.4. Вставка элементов из внешних источников 33
1.2.5. Разметка страницы 45
1.2.6. Форматирование текста 62
1.2.7. Формы и элементы пользовательского интерфейса 75
1.3. Каскадные таблицы стилей 91
1.3.1. Встраивание таблиц стилей в HTML-документ 92
1.3.2. Правила форматирования 94
1.3.3. Применение нескольких таблиц стилей 96
1.3.4. Единицы измерения 97
1.3.5. Шрифты 98
1.3.6. Цвет и фон 100
1.3.7. Размеры, поля, отступы и границы 102
1.3.8. Текст 105
1.3.9. Обтекание и видимость 107
1.3.10. Позиционирование 109
1.3.11. Графические фильтры 110
1.4. Что и как делают клиентские сценарии? 122
1.5. Что и как делают серверные сценарии? 126

Глава 2. Основы JavaScript **129**

2.1. Подготовка к программированию 129
2.2. Ввод и вывод данных 132
2.2.1. Метод *alert* 132
2.2.2. Метод *confirm* 133

2.2.3. Метод *prompt* 134
2.2.4. Метод *write* 134
2.3. Типы данных 135
2.4. Переменные и оператор присваивания 142
2.4.1. Имена переменных 142
2.4.2. Создание переменных 143
2.4.3. Область действия переменных 145
2.5. Операторы 145
2.5.1. Комментарии 146
2.5.2. Арифметические операторы 146
2.5.3. Дополнительные операторы присваивания 149
2.5.4. Операторы сравнения 150
2.5.5. Логические операторы 152
2.5.6. Операторы условного перехода 153
2.5.7. Операторы цикла 159
2.5.8. Выражения с операторами 164
2.6. Функции 166
2.6.1. Встроенные функции 167
2.6.2. Пользовательские функции 171
2.6.3. Выражения с функциями 177
2.7. Строки 178
2.7.1. Создание строкового объекта 178
2.7.2. Свойства объекта *String* 179
2.7.3. Методы объекта *String* обработки строк 180
2.7.4. Методы объекта *String* форматирования строк 185
2.7.5. Функции вставки и замены подстрок 187
2.7.6. Функции удаления ведущих и заключительных пробелов 188
2.8. Массивы 190
2.8.1. Создание массива 190
2.8.2. Многомерные массивы 192
2.8.3. Копирование массива 193
2.8.4. Свойства объекта *Array* 194
2.8.5. Методы объекта *Array* 195
2.8.6. Функции обработки числовых массивов 199
2.9. Числа 201
2.9.1. Создание объекта *Number* 204
2.9.2. Свойства объекта *Number* 204
2.9.3. Методы объекта *Number* 205
2.10. Математические вычисления 207
2.10.1. Свойства объекта *Math* 207
2.10.2. Методы объекта *Math* 207

2.11. Дата и время 209
2.11.1. Создание объекта *Date* 209
2.11.2. Методы объекта *Date* 211
2.11.3. Календарь 217
2.12. Объект *Boolean* 223
2.13. Объект *Function* 224
2.13.1. Создание объекта *Function* 224
2.13.2. Свойства объекта *Function* 225
2.13.3. Методы объекта *Function* 226
2.14. Пользовательские объекты 228
2.14.1. Создание объекта 228
2.14.2. Добавление свойств 231
2.14.3. Связанные объекты 231
2.14.4. Пример создания базы данных с помощью объектов 233
2.15. Специальные операторы 238
2.15.1. Побитовые операторы 238
2.15.2. Объектные операторы 239

ГЛАВА 3. КЛИЕНТСКИЕ СЦЕНАРИИ НА JAVASCRIPT 243

3.1. Объекты, управляемые сценариями 243
3.2. Обработка событий 251
3.2.1. Привязка обработчиков к элементам и событиям 252
3.2.2. Свойства события 257
3.2.3. Прохождение событий 263
3.3. Динамическое изменение элементов документа 266
3.3.1. Использование метода *write()* 267
3.3.2. Изменение значений атрибутов элементов 268
3.3.3. Изменение элементов 271
3.4. Работа с каскадными таблицами стилей 274
3.5. Окна и фреймы 280
3.5.1. Создание новых окон 281
3.5.2. Фреймы 285
3.5.3. Плавающие фреймы 293
3.5.4. Всплывающие окна 295
3.6. Загрузка изображений 299
3.7. Управление процессами во времени 302
3.8. Работа с cookie 304
3.8.1. Общие сведения 304
3.8.2. Функции чтения, записи данных и удаления cookie 307

3.9. Перемещение элементов мышью 311
3.9.1. Перемещение графических объектов 311
3.9.2. Перемещение текстовых областей 312
3.9.3. Перемещение плавающих фреймов 315
3.10. Меню 318
3.10.1. Раскрывающийся список 319
3.10.2. Одноуровневое меню 320
3.10.3. Простое двухуровневое меню 321
3.10.4. Сложное двухуровневое меню 323
3.10.5. Древовидный список 331
3.11. Таблицы 335
3.11.1. Доступ к элементам таблицы 335
3.11.2. Добавление и удаление строк таблицы 338
3.12. Простая база данных в текстовом файле 339
3.12.1. Элемент управления табличными данными 339
3.12.2. Перемещение по записям простых баз данных 345
3.12.3. Сортировка данных таблицы 348
3.12.4. Фильтрация данных таблицы 349
3.13. Поиск в тексте 352
3.14. Движение элементов по заданной траектории 354
3.15. Рисование линий 358
3.16. Обработка данных форм 361

Глава 4. Основы PHP 369

4.1. Подготовка к программированию 369
4.2. Вывод данных 372
4.3. Типы данных 374
4.4. Переменные и оператор присваивания 377
4.4.1. Имена переменных 377
4.4.2. Создание переменных 378
4.4.3. Отображение значений переменных 380
4.4.4. Переменные переменные 384
4.4.5. Область действия переменных 385
4.4.6. Проверка существования и типов переменных 386
4.5. Константы 387
4.6. Операторы 389
4.6.1. Комментарии 389
4.6.2. Арифметические операторы 389
4.6.3. Строковый оператор 391
4.6.4. Дополнительные операторы присваивания 391

4.6.5. Операторы сравнения 392
4.6.6. Логические операторы 393
4.6.7. Побитовые операторы 395
4.6.8. Операторы управления 396
4.6.9. Операторы цикла 399
4.7. Строки 399
4.7.1. Двойные и одинарные кавычки 399
4.7.2. Склейка строк 402
4.7.3. Преобразование строк 402
4.7.4. Форматирование строк 407
4.8. Числа 411
4.8.1. Математические функции 411
4.8.2. Математические константы 412
4.8.3. Представление чисел в различных системах счисления 413
4.8.4. Форматирование чисел 415
4.9. Дата и время 417
4.10. Массивы 420
4.10.1. Создание массива 420
4.10.2. Многомерные массивы 423
4.10.3. Отображение массивов 424
4.10.4. Операции над массивами 425
4.11. Глобальные предопределенные переменные 434
4.12. Функции 436
4.12.1. Пользовательские функции 436
4.12.2. Переменные функции 442
4.12.3. Встроенные функции 443
4.13. Объекты 443
4.13.1. Определение класса 444
4.13.2. Применение объектов 448
4.13.3. Ограничение доступа к свойствам и методам 449
4.13.4. Клонирование и удаление объектов 451
4.13.5. Использование методов несозданных объектов 452
4.13.6. Обработка исключений 452
4.13.7. Пример класса формы 453
4.14. Выполнение PHP-кода в текстовых строках 455

Глава 5. Серверные сценарии на PHP 457

5.1. Получение данных от клиента 457
5.1.1. Получение данных из HTML-форм 457
5.1.2. Загрузка файлов на сервер 466

5.2. Переходы и передача данных между Web-страницами 469
5.2.1. Вывод ссылок 470
5.2.2. Использование форм 470
5.2.3. Применение функции *header()* 470
5.2.4. Добавление информации к URL-адресу 472
5.2.5. Применение cookie 473
5.2.6. Применение сеансов PHP 475
5.3. Работа с текстовыми файлами 484
5.3.1. Открытие файла 484
5.3.2. Закрытие и удаление файлов 486
5.3.3. Чтение файла 487
5.3.4. Запись в файл 491
5.3.5. Простой счетчик посещений страницы 492
5.4. Работа с таблицами в текстовых файлах 493
5.4.1. Функции работы с табличными данными 494
5.4.2. Сложный счетчик посещений страницы 498
5.4.3. Баннер 504
5.4.4. Гостевая книга 508
5.4.5. Форум 514
5.5. Работа с базами данных 515
5.5.1. Общие сведения о базах данных 515
5.5.2. Основные средства PHP для взаимодействия с базой данных 517
5.5.3. Гостевая книга 520

ПРИЛОЖЕНИЯ 529

ПРИЛОЖЕНИЕ 1. ОСНОВНЫЕ ОБЪЕКТЫ БРАУЗЕРА И ДОКУМЕНТА 531

П1.1. Объект *window* 531
П1.1.1. Свойства объекта *window* 531
П1.1.2. Методы объекта *window* 533
П1.1.3. События объекта *window* 534
П1.2. Объект *document* 535
П1.2.1. Коллекции объекта *document* 536
П1.2.2. Методы объекта *document* 537
П1.2.3. События объекта *document* 537
П1.3. Объект *location* 538
П1.3.1. Свойства объекта *location* 538
П1.3.2. Методы объекта *location* 539
П1.4. Объект *history* 539
П1.4.1. Свойство объекта *history* 540
П1.4.2. Методы объекта *history* 540

П1.5. Объект *navigator* 540
П1.5.1. Свойства объекта *navigator* 540
П1.5.2. Коллекции объекта *navigator* 541
П1.5.3. Методы объекта *navigator* 541
П1.6. Объект *event* 541
П1.7. Объект *screen* 543
П1.8. Объект *TextRange* 543
П1.8.1. Свойства объекта *TextRange* 543
П1.8.2. Методы объекта *TextRange* 544

ПРИЛОЖЕНИЕ 2. УСТАНОВКА МОДУЛЯ PHP 546

П2.1. Где взять модуль PHP? 546
П2.2. Установка модуля PHP 547
П2.2.1. Размещение модуля PHP на диске компьютера 547
П2.2.2. Настройка модуля PHP 547
П2.2.3. Установка расширений PHP 548
П2.3. Настройка Web-сервера IIS 549

ПРИЛОЖЕНИЕ 3. УСТАНОВКА WEB-СЕРВЕРА IIS В WINDOWS 2000/XP 551

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 553

Введение

Данная книга посвящена языковым средствам создания Web-узлов (сайтов) и рассчитана на широкий круг читателей, желающих глубже разобраться в "кухне" разработки приложений для Web и интрасетей.

Следуя по пути от простого к сложному, обычно начинают с изучения HTML (HyperText Markup Language, язык разметки гипертекста). Затем для обеспечения дополнительной функциональности Web-страниц в HTML-документы вставляют программы (сценарии, скрипты), написанные на других языках. Сценарии могут выполняться Web-браузером (клиентом) и/или Web-сервером. На языке JavaScript обычно пишутся клиентские сценарии, хотя на нем можно писать и серверные скрипты. На языке PHP создаются серверные сценарии. В настоящее время JavaScript и PHP — наиболее популярные языки для разработки Web-приложений. Кроме того, они могут использоваться и для создания программных проектов, работающих на локальном компьютере или в локальной сети.

Новички обычно, начав с освоения HTML, довольно быстро приходят к использованию на своих страницах хотя бы несложных скриптов, написанных на JavaScript и выполняемых браузером. Однако рано или поздно у них возникает желание создать что-то более сложное, например гостевую книгу, форум, систему учета посещений и посетителей, защиту паролем и т. д. Эти задачи можно решить с помощью серверных сценариев.

Синтаксис JavaScript и PHP довольно прост и вполне доступен даже тем, кто не считает себя программистом или не желает заниматься программированием профессионально. Многие задачи вы можете решить, не имея большого опыта в программировании. Здесь важно не столько знание синтаксиса языка, сколько умение алгоритмически мыслить и создавать модели (обобщенные схемы) объектов и процессов. Так, например, если прочитав *главу 5*, даже новичок поймет, что такие с виду разные проекты, как счетчик посещений страницы, баннер и гостевая книга, устроены аналогично, я буду доволен собой.

В предлагаемой книге вы найдете многое из того, что необходимо знать для разработки более или менее функционально сложных сайтов. Здесь много и справочной информации, и примеров решения конкретных задач. Вместе с тем изложение построено так, чтобы книга оказалась доступной для начинающих.

В *главе 1* рассмотрены основные понятия, связанные с разработкой Web-сайта, а также основные сведения о языке HTML и каскадных таблицах стилей (CSS). Интересующиеся только скриптами могут использовать эту главу в качестве справочного руководства.

Глава 2 посвящена основам языка JavaScript, а *глава 3* — способам программирования и примерам различных клиентских сценариев. Аналогично распределен материал по языку PHP 5: в *главе 4* изложены основы, а в *главе 5* — примеры решения задач.

В *приложениях 1—3* приведены сведения по объектам документа и браузера, с которыми работают клиентские сценарии, а также рассказывается о том, как установить и настроить PHP и Web-браузер IIS в системе Windows.

В силу ограниченности объема в книге рассмотрены далеко не все возможности JavaScript и PHP. Более того, не обсуждается специфика программирования для браузеров, отличных от Microsoft Internet Explorer. Задача книги состоит в том, чтобы ввести читателя в мир JavaScript и PHP и дать ему возможность получить практические результаты хотя бы в системе Windows.

Многие из рассмотренных в книге примеров можно увидеть в действии на моем сайте **http://dunaevv1.narod.ru**.

Глава 1

Как устроен Web-сайт?

В данной главе мы рассмотрим устройство Web-сайта с общей точки зрения, позволяющей увидеть, что и какими средствами делается клиентом и сервером — двумя сторонами, обеспечивающими функционирование сайта.

1.1. Основные понятия

Web-сайт или, другими словами, *Web-узел*, — это совокупность Web-ресурсов, связанных между собой общей темой содержания, дизайном, а также функционально. Ни одна из перечисленных выше составляющих не является обязательной. Сайт — это некий информационный проект, опубликованный в Web. Web-ресурсы, входящие в сайт, — это файлы различного формата и содержания: HTML-документы, простые или отформатированные тексты, графика, аудио, видео, таблицы стилей, скрипты, Flash-ролики и др. Ссылки на эти ресурсы вставляются в HTML-документы, которые составляют основу сайта.

Сайт может состоять из одной или нескольких страниц. Одностраничный сайт создается легко и просто, однако такие сайты в Интернете встречаются редко. Чаще требуется создать хотя бы две-три страницы, и именно в этом случае возникают первые более или менее серьезные вопросы о технологии. В окне браузера можно показать одну или сразу несколько страниц сайта. При необходимости посредством гиперссылок можно организовать переходы между страницами, экспонируемыми в браузере. Сколько страниц в сайте, как они показываются в окне браузера, каким образом осуществлять переходы между ними и как обрабатывать действия пользователя — все это относится к области *разработки функциональности сайта* (development).

Под *дизайном* (design) сайта обычно понимают его внешний вид, а также интерфейс, как средство доступа к функциональным возможностям сайта. Одна

и та же структура сайта, с одной и той же функциональностью, может быть воплощена (представлена внешне) по-разному. При разработке сайта важно помнить, что он создается для посетителей, а потому должен быть содержательным, удобным и, по возможности, красивым. Основной принцип, которым следует руководствоваться при разработке сайта, заключается в том, что информация в нем должна быть представлена в понятном и удобном для использования виде. Если сайт состоит из нескольких страниц (документов), то необходимо продумать хорошую систему навигации, чтобы пользователь не заблудился в информационном пространстве и не тратил зря свои время и деньги на поиски необходимого или, по крайней мере, выхода из вашего лабиринта. Графика и визуальные эффекты должны привлекать внимание и способствовать быстрому и правильному восприятию информации, а не раздражать пестротой и долгой загрузкой в браузер. Считается хорошим тоном, если все страницы сайта выполнены в едином стиле. При этом создается впечатление соотнесенности компонентов сайта с единой концепцией. Именно по внешнему виду отдельных страниц посетитель ощущает, находится ли он еще внутри сайта или уже вышел за его границы.

Дизайн — важная и интересная сфера деятельности. Это то, что стоит впереди или на поверхности ваших многоэтажных разработок. Однако сам по себе, без информационного содержания, он ничего не стоит. Зрелый дизайнер, подобно архитектору, умеет сочетать внешний вид своего сооружения с его функциональностью и окружающей средой. Дизайнер должен немало потрудиться, чтобы проект нашел своего потребителя, принес пользу ему и своему автору, а также не нанес вреда, даже нечаянно, участникам сети. Дизайн представляет содержание в привлекательной, понятной и удобной пользователю форме. Если содержание бедно или вовсе отсутствует, то и представлять нечего. Главное в Интернете — информация в чистом виде. Адаптация этой информации к тем или иным категориям читателей — дело дизайна.

Все информационные ресурсы сайта располагаются на одном или нескольких удаленных компьютерах, входящих в Интернет и играющих роль *серверов*. Компьютер пользователя, обращающийся к ресурсу на сервере, является его клиентом. Web-браузер (например, Microsoft Internet Explorer или Netscape Navigator) является *клиентским программным обеспечением* для приема и отображения информационных ресурсов, а также для передачи данных на сервер.

В простейшем и наиболее распространенном случае браузер обращается к серверу, передавая ему URL-адрес (Uniform Resource Locator, унифицированный адрес ресурса) требуемого файла. Обычно это файл с расширением имени htm или html. Если имя файла в URL-адресе не указано, то подразумевается index.htm или index.html. Сервер ищет этот файл у себя или в сети

и в случае успеха просто пересылает его запросившему клиенту без какой бы то ни было предварительной обработки. Браузер клиента обрабатывает принятый файл и отображает результат в своей области просмотра. Например, если принятый файл содержит HTML-документ, то он обрабатывается интерпретатором языка HTML, а результаты обработки отображаются в окне браузера. Большинство сайтов в Интернете функционируют описанным выше способом. Мы будем называть их статическими, имея в виду лишь то, что сервер только принимает от клиента запрос файла и отправляет ему требуемый файл. Разумеется, это не означает, что содержимое статического сайта не может изменяться. Таким образом, ресурсы *статических сайтов* хранятся на сервере и пересылаются клиентам в ответ на их запрос, а обрабатываются только клиентом.

Кроме статических, существуют и так называемые *динамические сайты*. Они отличаются тем, что все или некоторые запрашиваемые клиентом ресурсы предварительно обрабатываются на сервере, а клиенту передается результат этой обработки. Клиент может передавать на сервер не только URL-адрес, но и данные, вводимые пользователем или генерируемые сценариями. Эти данные затем обрабатываются и/или сохраняются на диске сервера, а результаты обработки могут пересылаться клиенту. Таким образом, ресурсы динамических сайтов обрабатываются не только клиентом, но и сервером.

Многие интернет-провайдеры предоставляют бесплатную возможность размещения статических сайтов на своих серверах. Большинство частных, информационных и рекламных сайтов являются статическими. При этом взаимодействие владельца сайта с его посетителями обычно производится посредством электронной почты.

Размещение и поддержка динамических сайтов, требующих вычислительных ресурсов сервера (выполнения серверных сценариев), обычно осуществляется по специальному договору с провайдером и за определенную плату. Как правило, это сайты поисковых систем (например, Яндекс, Rambler и др.), а также коммерческих и производственных фирм, принимающих от посетителей заказы на товары и услуги не по электронной почте, а непосредственно со страницы сайта. Однако и частные лица, создающие, например, счетчики количества посещений сайта, гостевые книги, форумы и другие приложения, работающие с файловой системой и базами данных, используют серверные сценарии.

1.2. Как устроен HTML-документ?

HTML-документы являются основой любого сайта, поскольку именно они определяют, что и каким образом будет отображаться в окне браузера. В данном разделе мы кратко рассмотрим основные понятия и элементы языка HTML, на котором пишутся коды, формирующие содержимое Web-страниц.

В большинстве случаев HTML-документы создаются с помощью средств визуальной разработки, таких как Macromedia Dreamweaver, Microsoft FrontPage и др. Эти средства избавляют от необходимости писать коды на языке HTML вручную. Однако понимание этих кодов необходимо для применения сценариев — главной цели данной книги.

1.2.1. Начальные сведения

HTML-документы сохраняются на диске компьютера как обычные текстовые файлы с расширением htm или html. Они отличаются от обычных текстов только тем, что содержат специальные метки (команды, инструкции, дескрипторы, теги). Эти метки указывают программе просмотра (обозревателю, браузеру), что и как показывать на экране компьютера. Так, например, в HTML-документе можно указать, что данный текст следует выделить жирным наклонным шрифтом, а в таком-то месте разместить графическое изображение из заданного графического файла. Кроме того, можно создать специальную строку текста или картинку, щелчок левой кнопкой мыши на которой приведет к показу содержимого другого документа. Это так называемая *гиперссылка*. Текст, содержащий гиперссылки, называется *гипертекстом* и формируется с использованием специального языка HTML (HyperText Markup Language, язык разметки гипертекста). Файлы, содержащие собственно информацию, предназначенную для демонстрации посетителю Web-страницы, обычно называются *ресурсами* или *содержимым* (content) *сайта*.

Язык HTML довольно прост для понимания и применения. Все его теги представляют собой некоторые предопределенные ключевые слова (имена тегов), заключенные в угловые скобки < и >. Регистр, в котором записаны теги, не имеет значения. HTML-документ начинается с тега `<html>` и заканчивается тегом `</html>`. Между этими тегами могут располагаться другие теги и/или обычный текст. Большинство тегов являются парными: тегу `<тег>` соответствует тег `</тег>`. Первый из них называется *открывающим*, а второй — *закрывающим*. Такие теги еще называют *контейнерными* (или просто контейнерами), имея в виду, что между открывающим и закрывающим тегами можно разместить как в контейнере другие теги. Ниже приведен пример HTML-документа, который содержит текст, отображаемый в браузере жирным шрифтом, причем слово "друзья" выводится курсивом:

```
<html>
<b>Привет, <i>друзья</i></b>
</html>
```

Здесь в контейнере `<html>` находится контейнер `<b>`, задающий жирный шрифт для всей заключенной в нем строки, а в контейнере `<b>` расположен контейнер `<i>`, задающий шрифт курсивом для содержащейся в нем строки.

Вы можете записать рассмотренный выше код в любом текстовом редакторе, например Блокноте Windows (исполняемый файл notepad.exe), и сохранить на жестком диске в файле с расширением htm или html, а затем открыть его в браузере. Вид данного документа в окне браузера Microsoft Internet Explorer показан на рис. 1.1.

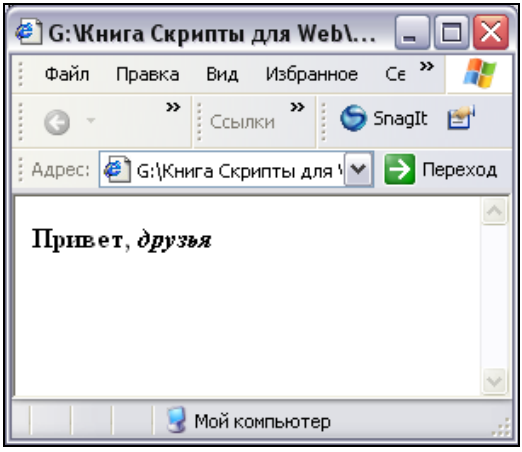


Рис. 1.1. Пример простого HTML-документа

Внимание

Между открывающейся угловой скобкой и ключевым словом (именем) тега не должно быть пробелов. В противном случае тег будет не интерпретироваться, а просто выводиться как обычный текст.

Теги могут иметь параметры, называемые *атрибутами*. Атрибуты, в свою очередь, могут иметь значения (аргументы). Атрибуты со значениями задаются парой вида *имя_атрибута=значение*. Имя атрибута является ключевым словом, а допустимое значение определяется спецификацией этого атрибута — описанием того, какие значения возможны и как их задавать.

Некоторые теги имеют атрибуты, которые обязательно должны быть указаны. Например, для вставки в HTML-документ графического изображения используется тег `<img>` с обязательным атрибутом `src="адрес_файла"`. В качестве значения атрибута `src` используется имя (при необходимости с указанием пути) или URL-адрес графического файла. Например,

```
<img src="http://www.myserver.ru/images/picture.jpg">
```

Если не указывать необязательные атрибуты, то их значения будут равны принятым по умолчанию. Например, тег `<img>` имеет еще атрибуты `width` и `height`, задающие соответственно ширину и высоту изображения в окне браузера. Если их не указать, то изображение будет занимать на экране прямоугольную область в соответствии со своими оригинальными размерами.

Если в теге необходимо указать несколько предусмотренных для него атрибутов, то они могут следовать за именем тега в произвольном порядке. Разделителем между ними является один или несколько пробелов.

Внимание

Теги, которые браузер не может проинтерпретировать (в том числе с неверно записанными именами), просто игнорируются им.

Часть HTML-кода, заключенная между `<!` и `>`, интерпретируется браузером как *комментарий*, т. е. не выполняется и не выводится на экран. Например, при интерпретации браузером следующего HTML-кода ничего не будет отображено:

```
<html>
  <! Это комментарий>
  <! img src="mypicture.jpg">
</html>
```

Содержимое HTML-документа в целом, а также запись отдельного тега может занимать одну или несколько строк. При этом имена тегов, атрибутов и значения атрибутов не следует разрывать для переноса на другую строку. Количество пробелов между элементами тегов и между самими тегами, отступы и количество пустых строк между элементами HTML-документа не влияют на его отображение в окне браузера, а служат лишь для удобства чтения HTML-кода.

Если не принять специальных мер, элементы HTML-документа располагаются рядом друг с другом по горизонтали и вертикали в порядке их записи в коде, а также в зависимости от того, как они вписываются в размеры окна браузера. Это так называемая *базовая линейная модель* интерпретации HTML-кода и отображения ее результатов. Чтобы определить другой порядок размещения элементов в окне браузера, следует использовать специальные теги, атрибуты и/или правила каскадных таблиц стилей (CSS). Например, чтобы некоторый элемент был размещен ниже предыдущего, достаточно перед тегом, определяющим данный элемент, записать тег `<br>`. Для задания нового абзаца при форматировании текста или для размещения элемента ниже предыдущего с пропуском одной строки достаточно использовать тег `<p>`.

Множество всех тегов HTML можно разбить на следующие группы:

- структура документа;
- ссылки;
- вставка содержимого из внешних источников;
- разметка страницы;
- форматирование текста;
- формы и элементы пользовательского интерфейса.

В следующих разделах данной главы мы кратко рассмотрим основные теги этих групп.

1.2.2. Структура документа

Тег *<!DOCTYPE>*

Во многих HTML-документах, созданных в специальных редакторах типа Dreamweaver, перед открывающим тегом `<html>` нередко можно увидеть тег `<!DOCTYPE>`, с помощью которого объявляется тип и формат содержимого документа. Этот же тег вставляется и в XML-документы. Однако современные Web-браузеры обрабатывают документы и без него. Ниже приведен пример содержимого тега `<!DOCTYPE>`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
```

Здесь определен тип документа HTML, публичное DTD-описание которого принадлежит организации W3C, соответствует стандарту HTML 4.0 и ориентировано на англоязычные документы.

Если HTML-документ без тега `<!DOCTYPE>` открыть в браузере Internet Explorer 6.0, а затем выполнить команду **Файл | Сохранить как**, выбрав при этом тип файла html, то браузер добавит в сохраняемый файл тег `<!DOCTYPE>`.

Возможны следующие версии DTD (Document Type Definition — определение типа документа):

- Strict DTD — исключает все нерекомендованные консорциумом W3C теги и атрибуты, такие как `<FONT>`, `<CENTER>`, `align` и другие, связанные с форматированием, которые можно заменить соответствующими параметрами CSS.

  Пример:

  ```
  <!DOCTYPE HTML PUBLIC ”-//W3C//DTD HTML 4.01//EN”
  ”http://www.w3.org/TR/HTML4.01/strict.dtd”>
  ```

- Transitional DTD — включает все элементы версии Strict DTD, а также все нерекомендованные элементы для обеспечения обратной совместимости со всеми существующими браузерами.

  Пример:

  ```
  <!DOCTYPE HTML PUBLIC ”-//W3C//DTD HTML 4.01 Transitional//EN”
  ”http://www.w3.org/TR/HTML4.01/loose.dtd”>
  ```

- Frameset DTD — включает такие же элементы, что и Transitional DTD, а также элементы, необходимые для Web-страниц с фреймовой структурой.

  Пример:

  ```
  <!DOCTYPE HTML PUBLIC ”-//W3C//DTD HTML 4.01 Frameset//EN”
  ”http://www.w3.org/TR/HTML4.01/frameset.dtd”>
  ```

Тег *<head>*

Контейнерный тег `<head>` определяет раздел заголовка HTML-документа и может встречаться в нем не более одного раза. В этом теге размещают другие контейнерные теги:

- `<title>` — текст, отображаемый в заголовке окна браузера;
- `<meta>` — данные для использования серверами и поисковыми системами;
- `<base>` — базовый URL-адрес документа;
- `<link>` — связи между документами;
- `<style>` — таблицу стилей;
- `<script>` — код сценария.

Тег *<meta>*

Информация, содержащаяся в тегах `<meta>`, не отображается браузером, однако имеет важное значение. В частности, тег позволяет задать кодовую страницу языка просмотра документа, ключевые слова, по которым ваш документ будут искать поисковые системы, и т. д. Вот типичный пример использования тегов `<meta>`:

```
<html>
  <head><title>Пример<title>
    <meta http-equiv="Content-Type" content="text/html; charset=koi8-r">
    <meta name="keywords"
          content="Примеры html скрипты сценарии javascript">
  </head>
```

```
  ... здесь располагаются другие теги
</html>
```

Рассмотрим атрибуты тега `<meta>`.

Группа атрибутов *HTTP-EQUIV* (HTTP-эквиваленты)

Данная группа включает следующие атрибуты.

- `EXPIRES` — дата устаревания документа.

  После истечения указанного срока документ будет каждый раз загружаться заново, а не браться из кэша на вашем локальном диске.

  Формат даты: RFC850.

  Пример:

  ```
  <META HTTP-EQUIV="EXPIRES" CONTENT="Wed, 23 Feb 2005 08:25:53 GMT">
  ```

- `PRAGMA` — управление кэшированием.

  Возможно одно значение `NO-CACHE`, т. е. данный документ не кэшируется браузером. В этом случае запрашиваемая страница будет браться с сервера, а не из кэша (буфера) пользовательского компьютера.

  Пример:

  ```
  <META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE">
  ```

- `CONTENT-TYPE` — тип документа и его кодировка.

  Выбор кодовой страницы для правильного отображения символов браузером.

  Примеры:

  ```
  <META HTTP-EQUIV="CONTENT-TYPE" CONTENT="text/html;
        charset=windows-1251">
  <meta http-equiv="Content-Type" content="text/html; charset=koi8-r">
  ```

- `CONTENT-LANGUAGE` — указание языка документа.

  Значение этого параметра может использоваться как поисковыми роботами, так и Web-серверами.

  Формат: `<`*Язык*`>-<`*Диалект*`>`.

  Примеры:

  ```
  <META HTTP-EQUIV="CONTENT-LANGUAGE" CONTENT="en-GB">
  <META HTTP-EQUIV="CONTENT-LANGUAGE" CONTENT="ru">
  ```

- `REFRESH` — время (в секундах), через которое произойдет автоматическая перезагрузка документа или переход на другой документ с заданным URL.

  Формат: "*время*" или "*время; URL*"

Пример:

```
<META HTTP-EQUIV="REFRESH" CONTENT="30">
```

Здесь указано, что периодическую перезагрузку документа следует производить через 30 секунд.

```
<META HTTP-EQUIV="REFRESH" CONTENT="5; http://www.microsoft.com">
```

Здесь указано, что спустя 5 секунд следует перейти к документу по указанному адресу.

Автоматическую периодическую перезагрузку документа обычно назначают в случаях часто обновляемых данных (котировки акций, результаты спортивных соревнований и т. п.).

- `CACHE-CONTROL` — управление кэшированием.

  Возможные варианты: кэширование в общем (`PUBLIC`) или частном (`PRIVATE`) кэше. Документ вообще не кэшируется (`NO-CACHE`) или кэшируется, но не сохраняется (`NO-STORE`).

  Пример:

```
<META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-STORE">
```

Группа атрибутов *NAME* (имя)

К этой группе относятся следующие атрибуты.

- `DESCRIPTION` — описание документа. Один из наиболее важных параметров. Информация, содержащаяся в нем, влияет на результаты поиска, осуществляемого поисковыми системами. В общем случае вид результатов поиска, как правило, выглядит так:
  - URL документа;
  - название документа (содержимое тега `<title>`);
  - описание документа, т. е. `DESCRIPTION` или начальный фрагмент HTML-документа, если `DESCRIPTION` отсутствует (нередко — и то и другое). В первом случае пользователь получает достаточно краткое и информативное описание документа, а во втором случае это может быть бессмысленный набор слов или несколько первых фраз из вашего документа;
  - рейтинг (коэффициент соответствия документа запросу пользователя).

  Пример:

```
<META NAME="DESCRIPTION"
      CONTENT="Описание данного документа, до 100 символов">
```

- `KEYWORDS` — ключевые слова. Набор слов и фраз, наиболее полно характеризующих данный документ, которые являются основным критерием поиска вашего документа поисковыми системами. В конечном счете, эти слова учитываются при выдаче результатов поиска и способствуют повышению рейтинга вашего сайта.

  Пример:

  ```
  <META NAME="KEYWORDS"
        CONTENT="Примеры html скрипты сценарии javascript">
  ```

- `DOCUMENT-STATE` — статус документа. Данный параметр управляет частотой индексации вашего документа поисковыми серверами и может принимать два значения.

  - `STATIC` (документ статичен, т. е. не меняется, и, следовательно, индексировать его нужно только один раз).

    Пример:

    ```
    <META NAME="DOCUMENT-STATE" CONTENT="STATIC">
    ```

  - `DYNAMIC` (для часто изменяющихся документов, которые нужно реиндексировать).

    Пример:

    ```
    <META NAME="DOCUMENT-STATE" CONTENT="DYNAMIC">
    ```

- `ROBOTS` — управление процессом индексации. Варианты:

  - `INDEX` — возможность индексирования данного документа (иначе — `NOINDEX`);
  - `FOLLOW` — возможность индексирования всех документов, на которые есть ссылки в данном HTML-файле (иначе — `NOFOLLOW`);
  - `ALL` — одновременное выполнение условий `INDEX` и `FOLLOW`;
  - `NONE` — одновременное выполнение условий `NOINDEX` и `NOFOLLOW`.

  Пример:

  ```
  <META NAME="ROBOTS" CONTENT="INDEX, NOFOLLOW">
  ```

- `RESOURCE-TYPE` — тип ресурса. Для обычных HTML-документов значение этого параметра устанавливается равным `"DOCUMENT"`.

  Пример:

  ```
  <META NAME="RESOURCE-TYPE" CONTENT="DOCUMENT">
  ```

- UPDATED — дата обновления страницы.

  Пример:

  ```
  <META NAME="UPDATED" CONTENT="25.11.04">
  ```

- URL — базовый URL-адрес; определяет, какой документ следует индексировать (чтобы не обрабатывать "зеркала").

  Пример:

  ```
  <META NAME="URL" LANG="ru" CONTENT="http://www.admiral.ru/hp/dunaev">
  ```

- AUTHOR — информация об авторе данного документа.

  Пример:

  ```
  <META NAME="AUTHOR" CONTENT="Вадим Дунаев">
  ```

- COPYRIGHT — информация об авторских правах.
- GENERATOR — название программы, создавшей HTML-код.

Допустимо, но не обязательно, добавлять в метатеги атрибут LANG, указывающий язык данных.

Пример:

```
<META NAME="KEYWORDS" LANG="ru" CONTENT="музыка видео графика">
```

Примечание

Если HTML-документ без тегов <meta> открыть в браузере Internet Explorer 6.0, а затем выполнить команду **Файл | Сохранить как**, выбрав при этом тип файла html, то браузер добавит в сохраняемый файл тег <!DOCTYPE> и несколько тегов <meta>.

Тег *<base>*

В HTML-документах обычно используются *относительные ссылки* на ресурсы (другие HTML-документы, графику и т. п.). Например, в ссылке нередко приводится только имя файла. В этом случае предполагается, что его местоположение совпадает с расположением документа, из которого этот файл вызывается. Однако можно явно указать базовый URL-адрес, который будет использоваться во всех случаях применения относительных ссылок. Это делается с помощью тега <base> с атрибутом href, значением которого является базовый URL-адрес:

```
<base href="URL-адрес">
```

Пример:

```
<html>
  <head>
    <base href="//www.admiral.ru/hp/dunaev/images">
  </head>
  <img src="picture.jpg">
</html>
```

В этом примере графическое изображение имеет относительную ссылку (просто имя файла). Если базовый URL-адрес не был бы указан, то браузер искал бы файл picture.jpg в той же папке, где расположен HTML-документ со ссылкой на него. Поскольку базовый URL-адрес задан, то браузер будет искать графический файл в папке /hp/dunaev/images на Web-узле **www.admiral.ru**.

При указании в теге `<base>` базового URL-адреса относительные ссылки продолжают работать, даже если вы переместите HTML-документ в другую папку. Это обеспечивает определенные удобства при работе с локальной версией HTML-документа в условиях, когда компьютер подключен к сети.

Тег *<link>*

Тег `<link>` предназначен для обозначения связей между документами в больших и сложных по своей структуре узлах. Атрибут `href` содержит ссылку на другой документ, связанный с текущим. Адрес может быть абсолютным или относительным. Атрибут `rel` указывает отношение между документами:

- `home` — расположение домашней страницы Web-узла;
- `next` — расположение следующего документа серии, связанного с текущим;
- `previous` — расположение предыдущего документа серии, связанного с текущим;
- `up` — расположение следующего документа вверх по иерархии;
- `copyright` — расположение документа с информацией об авторских правах для данного Web-узла;
- `stylesheet` — расположение файла с таблицей стилей.

Используются также и другие значения для атрибута `rel`: `glossary`, `help`, `toc`, `index`, `contents`.

Пример:

```
<link rel="stylesheet" href="http://mysite.ru/styles/mystyle.css">
```

В одном и том же HTML-документе тег `<link>` может присутствовать несколько раз.

Тег *<style>*

В контейнерном теге `<style>` размещаются правила каскадных таблиц стилей, определяющих внешний вид и позиционирование элементов HTML-документа. Более подробно каскадные таблицы стилей будут рассмотрены в *разд. 1.3*. В одном и том же HTML-документе тег `<style>` может встречаться несколько раз. При этом они могут размещаться и вне тега `<head>`.

Тег *<script>*

В контейнерном теге `<script>` размещается код сценария (скрипт), выполняемый браузером. В одном и том же HTML-документе тег `<script>` может встречаться несколько раз. При этом они могут размещаться и вне тега `<head>`. Внутри тега `<script>` не могут находиться теги HTML и правила каскадных таблиц стилей.

Код сценария может быть написан на языке JavaScript или VBScript. Последний понимает только браузер Internet Explorer. Языком по умолчанию считается JavaScript. Чтобы явно указать, на каком языке написан сценарий, применяется атрибут `language`, который может принимать следующие значения:

- `"javascript"`, `"JScript"`;
- `"VBScript"`, `"VBS"`.

Редакция языка JavaScript для Internet Explorer называется JScript. Вместе с тем в Internet Explorer можно использовать и `language="javascript"`. В браузере Netscape Navigator значимым является только `language="javascript"`, ссылка `language="JScript"` будет им проигнорирована. Поэтому при разработке сценариев, рассчитанных для различных браузеров, рекомендуется использовать ссылку `language="javascript"`. В примерах сценариев на языке JavaScript, приводимых в настоящей книге, мы не будем указывать атрибут `language` из соображений экономии места.

Современные браузеры распознают еще и версию языка. Например, Internet Explorer 4.0 и Netscape Navigator 4.0 знают версию языка JavaScript 1.2. Во время написания этой книги была доступна версия JavaScript 1.5, которую распознают Internet Explorer 6.0 и Netscape Navigator 6.0. Если версия языка JavaScript в значении атрибута `language` указана, а браузер не знает ее, то сценарий может быть просто проигнорирован. Если же версия языка не указана, то браузер, которому она не знакома, может неправильно выполнять некоторые выражения сценария или даже выдавать сообщения об ошибках.

Сценарий может размещаться и в отдельном файле, обычно имеющем расширение js. В этом случае в теге `<script>` следует указать атрибут `src` с именем или URL-адресом файла. Например,

```
<script src="http://www.myserver.ru/myscript.js"></script>
```

При этом между тегами `<script>` и `</script>` ничего не записывается.

Старые браузеры, появившиеся раньше JavaScript, игнорируют теги `<SCRIPT>` и `</SCRIPT>`, а все, что находится между ними, интерпретируют как текстовое содержимое HTML-документа. Результат может быть самым неожиданным. Чтобы уменьшить вероятность отображения кода сценария в окне старого браузера, следует заключить его в дескрипторы комментария `<!--` и `-->`. Новые браузеры, поддерживающие сценарии, будут игнорировать эти символы, выполняя код сценария, а старые браузеры (не понимающие сценарии), наоборот, станут игнорировать код сценария. Вот как используются символы комментария предохранения сценария от старых браузеров:

```
<SCRIPT LANGUAGE="JavaScript" >
  <!--
    // код сценария
  //-->
</SCRIPT>
```

Обратите внимание на заключительные символы тега комментария, перед которыми стоят две косые черты. Без них JavaScript будет пытаться интерпретировать символы `-->` как заключительные символы комментария HTML, а с ними он просто их проигнорирует.

Внимание

В примерах, приводимых в настоящей книге, мы не будем указывать символы комментария для адаптации к старым браузерам из соображений экономии места. Однако помните, что современная культура оформления сценариев обязывает нас делать это в своих практических приложениях.

Тег *<body>*

Основной раздел HTML-документа, содержащий текст, ссылки, графические, звуковые и другие объекты, размещается вне раздела заголовка, задаваемого тегом `<head>`. Часто основной раздел заключают в контейнерный тег `<body>`, который может встречаться в HTML-документе не более одного раза и не может содержать тег заголовка `<head>`.

Например:

```
<html>
  <head>
```

```
    ... здесь располагаются теги раздела заголовка документа
  </head>
  <body>
    ... здесь располагаются теги основного раздела документа
  </body>
</html>
```

Вообще говоря, использование тега `<body>` не обязательно. Однако он обладает атрибутами, с помощью которых можно задать параметры сразу для многих элементов, содержащихся в `<body>`:

- `alink` — цвет активной ссылки;
- `link` — цвет еще не использованной (непросмотренной) ссылки;
- `vlink` — цвет уже использованной (просмотренной) ссылки;
- `text` — цвет текста;
- `bgcolor` — цвет фона документа;
- `background` — URL-адрес графического изображения, используемого в качестве фона документа;
- `leftmargin`, `rightmargin`, `topmargin`, `bottommargin` — ширина в пикселах левого, правого, верхнего и нижнего полей (отступов) документа соответственно;
- `bgproperties` — если установлено значение `fixed` (`bgproperies=fixed`), то фоновое изображение (атрибут `background`) не будет прокручиваться;
- `scroll` — значение `yes` устанавливает, а значение `no` удаляет полосы прокрутки окна браузера.

Ни один из перечисленных выше атрибутов не является обязательным. По умолчанию браузер заполняет фон сплошным цветом в зависимости от настройки пользователя. Чтобы настройки пользователя не могли испортить восприятие вашего документа, следует самому зафиксировать все цвета (фона, текста и ссылок). Если вы не зададите цвета явным образом, то настройки пользовательского браузера скорее всего позволят прочитать текст и ссылки, однако при этом дизайн вашего документа окажется не контролируемым вами.

Значение цвета задается либо именем цвета из предопределенного списка, либо шестнадцатеричным числом в формате `#XXXXXX`, где `X` — шестнадцатеричная цифра (0, 1, 2, ..., 9, a, b, c, d, e, f). При этом используется модель RGB представления цвета: первые две цифры задают яркость красной (Red), вторые две цифры — зеленой (Green), а последние две цифры — синей (Blue) составляющей цвета. Каждая составляющая цвета может иметь одно из 256 значений (оттенков). Например, черному цвету соответствует значение `#000000`

(все составляющие имеют минимальную яркость); белому цвету соответствует значение `#ffffff` (все составляющие имеют максимальную яркость); если значения яркости всех трех составляющих цвета одинаковы, то получается серый цвет (возможны 256 оттенков серого от черного до белого). Ниже приведены два варианта задания красного цвета для текста и желтого для фона:

```
<body text=red bgcolor=yellow>
<body text=#ff0000 bgcolor=#ffff00>
```

Список предопределенных имен цветов содержит более сотни элементов и поиск в нем названий цветов, отличных от основных, довольно затруднителен. Однако цвета из этого списка соответствуют так называемой *Web-палитре*. Использование цветов из этой палитры гарантирует их одинаковое воспроизведение всеми браузерами.

Теги * * и **

Для выделения некоторого фрагмента документа, чтобы иметь возможность задать его общие параметры, служат контейнерные теги `<div>` и `<span>`. Контейнер `<div>` может содержать различные элементы (текст, графические изображения, ссылки, объекты и т. п.), а `<span>` обычно применяется к фрагментам текста.

Общие параметры для элементов, включенных в данные контейнеры, обычно назначаются с помощью таблиц стилей (тег `<style>` или атрибут `style`). В частности, можно задать произвольное относительное положение фрагментов документа, в том числе и наложение их друг на друга, имитируя тем самым создание слоев. Дополнительно к этому тег `<div>` имеет атрибут `align` с возможными значениями `center`, `left` и `right`, позволяющий выровнять весь фрагмент документа по центру, левому и правому краю окна браузера соответственно. В следующем примере фрагмент документа, содержащий два изображения, выравнивается по центру окна браузера:

```
<html>
  <head><title>Пример</title></head>
  <body>
    Пример использования контейнера:
    <div align=center>
      <img src="picture1.jpg">
      <img src="picture2.jpg">
    </div>
  </body>
</html>
```

Теги, влияющие на расположение элементов

Тексты и теги видимых элементов записываются в документе друг за другом и в этой же последовательности выполняются браузером. В результате они отображаются в окне браузера. Многие элементы по умолчанию (если не предусмотреть специальных мер) располагаются рядом друг с другом по горизонтали и/или вертикали в зависимости от того, как они вписываются в размеры окна браузера. В крайнем случае, при невозможности отобразить все элементы целиком в окне браузера автоматически появляются полосы прокрутки. Впрочем, отображение этих полос можно и запретить. Для контроля порядка расположения элементов в окне браузера служат специальные средства: теги, атрибуты и таблицы стилей. В данном разделе будут рассмотрены следующие теги, влияющие на расположение любых элементов:

- `<p>` — абзац, т. е. переход на следующую строку и пропуск одной строки;
- `<br>` — переход на следующую строку;
- `<nobr>` — невозможность переноса слов и режима обтекания текстом;
- `<wbr>` — "мягкий" перенос в блоке текста, заключенного в контейнерный тег `<nobr>`;
- `<center>` — выравнивание заключенных в него элементов по центру;
- `<hr>` — размещение горизонтальной разделительной полосы.

Тег *<p>*

Тег `<p>` используется для выделения блоков текста (абзацев). Этот тег приводит к вставке одной пустой строки и размещению находящегося за ним видимого элемента в следующей строке (если, разумеется, параметры стиля этого элемента не задают какое-либо другое расположение). Например, чтобы два элемента находились друг под другом и между ними была пустая строка, достаточно между тегами этих элементов вставить тег `<p>`:

```
<html>
  <img src="picture1.jpg">
  <p>
  <img src="picture2.jpg">
</html>
```

Тег `<p>` является контейнерным. Однако закрывающий тег `</p>` редко используется, поскольку браузер считает, что начало следующего абзаца означает конец предыдущего:

```
<html>
  Ниже показаны две мои фотографии:
```

```
  <p>
  <img src="picture1.jpg">
  <p>
  <img src="picture2.jpg">
</html>
```

Очевидно, чтобы вставить несколько пустых строк, достаточно записать тег `<p>` несколько раз подряд.

Тег `<p>` имеет необязательный атрибут `align` с возможными значениями `center`, `left`, `right` и `jastify`, позволяющими выровнять весь фрагмент документа по центру, левому краю, правому краю и одновременно по двум сторонам окна браузера соответственно. Используя атрибут `style`, можно определить специфические параметры расположения. Необязательный атрибут `title` задает текстовую строку, которая появляется при наведении указателя мыши на содержимое тега `<p>` (всплывающая подсказка).

Теги *
*, *<nobr>* и *<wbr>*

Тег `<br>` служит для перехода на другую строку. Он не является контейнерным и, в отличие от тега `<p>`, не вставляет дополнительной пустой строки. Тег `<br>` имеет необязательный атрибут `clear`, принимающий значения `left`, `right` и `all`. Обычно тег `<br>` используется без указания этого атрибута. Однако иногда требуется прекратить выравнивание элемента документа (например, изображения) относительно текста. В этом случае атрибут `clear` позволяет указать, какое именно выравнивание следует прекратить.

Контейнерный тег `<nobr>` предотвращает перенос заключенной в него строки текста. Если при этом строка выходит за границу окна браузера, то появляется горизонтальная полоса прокрутки.

В тексте, заключенном в контейнер `<nobr>`, можно указать место возможного перевода строки ("мягкий" перенос слов), который будет выполнен браузером лишь при необходимости. Это делается с помощью тега `<wbr>`.

Тег *<hr>*

При оформлении документа, чтобы отделить один фрагмент от другого, нередко используют разделительные полосы, задаваемые тегом `<hr>`. Разделительная полоса всегда выводится в новую строку. Элемент, следующий за тегом `<hr>`, выводится ниже разделительной полосы.

Тег `<hr>` имеет следующие необязательные атрибуты:

- `size` — толщина полосы в пикселах;
- `width` — длина полосы в пикселах;

- `align` — способ выравнивания (принимает значения `center`, `left` и `right` для выравнивания по центру, левому и правому краю окна браузера соответственно);
- `noshade` — атрибут, не имеющий значений; если указан, то создается сплошная черная полоса без тени.

Специальные атрибуты

Большинство тегов имеют, кроме прочих, необязательные специальные атрибуты:

- `class` — для связи с таблицей стилей, определенной контейнерным тегом `<style>`;
- `style` — для определения таблицы стилей только для данного элемента (не путайте тег `<style>` и атрибут `style`);
- `id` — идентификатор для обеспечения доступа к элементу из сценария и/или таблиц стилей (подобно имени переменной);
- *атрибут-событие* — для обеспечения связи со сценарием, который выполняется при наступлении соответствующего события (например, `onclick`, `onload`, `onchange` и др.);
- `title` — для определения строки текста, появляющейся при наведении указателя мыши на элемент, заданный тегом, в котором этот атрибут установлен (т. е. для задания всплывающей подсказки).

При описании тегов HTML перечисленные выше атрибуты не всегда упоминаются. Значения и применение этих атрибутов будут подробно рассмотрены в соответствующих разделах (в нужном месте). Так, атрибуты `class` и `style` будут описаны при рассмотрении каскадных таблиц стилей (CSS); атрибуты `id` и атрибуты-события — при рассмотрении сценариев; атрибут `title` не нуждается в особых комментариях. На данном этапе достаточно только иметь в виду, что кроме описываемых атрибутов, теги могут иметь еще и перечисленные выше специальные атрибуты, обеспечивающие взаимосвязь HTML, CSS и сценариев.

1.2.3. Ссылки

Ссылки (или *гиперссылки*) позволяют щелчком кнопкой мыши на выделенном фрагменте текста или изображении перейти к другому файлу или разделу текущего документа. Ссылки применяются в большинстве существующих Web-страниц и являются их главным инструментом. Если документ содержит ссылки на другие документы, то он называется *гипертекстовым*.

Ссылки могут быть текстовыми и графическими в зависимости от того, что является указателем ссылки (видимым элементом документа, щелчок на котором левой кнопкой мыши инициализирует переход к другому документу). Текстовые ссылки представляют собой выделенное слово или целую фразу. Выделение ссылки производится цветом или подчеркиванием, в зависимости от настройки браузера. В графических ссылках роль чувствительного к щелчку элемента играет изображение. Можно также создать комбинированные ссылки, указателем в которых является и изображение, и текст. Ссылки на фрагменты текущего документа (внутренние ссылки) организуются особым образом. В любом случае ссылки задаются контейнерным тегом `<a>`.

Для создания ссылки на другой документ используется следующий синтаксис:

```
<a href="адрес_ссылки">указатель_ссылки</a>
```

Здесь атрибут `href` (от англ. *hipper reference* — гиперссылка) принимает в качестве значения URL-адрес или имя файла (возможно, с указанием пути), к которому следует перейти при инициализации ссылки; *`указатель_ссылки`* — текст или тег `<img>`, вставляющий графическое изображение.

По умолчанию вызываемый при активизации ссылки документ открывается в текущем окне браузера, заменяя документ, содержащий эту ссылку. Это происходит, если браузер способен открыть документ в своем окне (некоторые документы открываются в окне соответствующих приложений). Однако вы можете определить, в каком окне браузера или фрейме следует открыть вызываемый документ. Это делается с помощью атрибута `target`, который принимает следующие значения:

- *`имя_окна`* — имя окна или фрейма;
- `_parent` — указатель на родительский фрейм;
- `_blank` — указатель на новое окно;
- `_top` — указатель на фрейм верхнего уровня;
- `_self` — указатель на тот же самый (текущий) фрейм.

Если требуется, чтобы при наведении указателя мыши на указатель ссылки появлялась всплывающая подсказка (поясняющий текст), то в теге `<a>` следует использовать атрибут `title`, принимающий в качестве значения текст всплывающей подсказки.

Текстовые ссылки

Структура текстовой ссылки имеет следующий вид:

```
<a href="адес_ссылки">текст_ссылки</a>
```

Например, следующий тег описывает ссылку на HTML-файл мой_документ.htm, при этом ссылка на экране будет представлена текстом **Щелкните здесь**:

```
<a href="мой_документ.htm">Щелкните здесь</a>
```

Отметим, что браузер не выводит на экран имя файла, к которому требуется перейти по ссылке, а лишь показывает текст, заключенный в теге между угловыми скобками > и <. Если же вы хотите, чтобы внешне ссылка выглядела как имя файла, на который она ссылается, то просто напишите его имя вместо текста, например:

```
<a href="мой_документ.htm">мой_документ.htm</a>
```

Можно ссылаться не только на другие файлы, но и на собственный файл.

Текст, заключенный в контейнерный тег ссылки `<a>`, может быть отформатирован как с помощью рассмотренных ранее тегов форматирования, так и с помощью таблиц стилей, которые мы обсудим позже. Например:

```
<a href="мой_документ.htm"><i>Щелкните <b>здесь</b></i></a>
```

Примечание

В приведенных выше примерах ссылка задавалась как имя файла. В этом случае предполагается, что файл находится в той же папке, что и документ, содержащий эту ссылку. Если этого недостаточно, то вы можете указать базовый адрес с помощью тега `<base>`, использовать относительные и абсолютные пути, а также URL-адрес файла.

Графические и комбинированные ссылки

Структура графической ссылки имеет вид:

```
<a href="адрес_ссылки"><img src="адрес_графического_файла"></a>
```

Например, следующий тег описывает ссылку на HTML-файл мой_документ.htm, при этом ссылка на экране будет представлена изображением из файла picture.gif:

```
<a href="мой_документ.htm"><img src="picture.gif"></a>
```

К графической ссылке можно добавить поясняющий текст:

```
<a href="мой_документ.htm"><img src="picture.gif">Щелкните здесь</a>
```

В этом примере переход к документу мой_документ.htm произойдет при щелчке левой кнопкой мыши как на графическом изображении, так и на поясняющем тексте, поскольку и то и другое находится в контейнерном теге `<a>`.

Браузер по умолчанию выделяет графические ссылки рамкой вокруг ее графического изображения. Иногда это нежелательно, и чтобы исключить отображение рамки, достаточно использовать в теге `<img>` атрибут `border=0`, как в следующем примере:

```
<a href="мой_документ.htm"><img src="picture.gif" border=0>Щелкните здесь</a>
```

Графические карты ссылок

В рассмотренных выше графических ссылках одному изображению соответствовал один адрес ссылки. Однако имеется и другая возможность. Она заключается в том, чтобы одному изображению сопоставить несколько ссылок, привязав каждую из них к некоторой области изображения. Такие области называют *горячими* или *активными*, а сам технологический прием — *графической картой ссылок* или *сегментированной графикой*. Горячие области графической карты могут быть различной формы: прямоугольной, многоугольной или в виде окружности. Это очень удобный прием оформления группы ссылок, однако при выборе рисунка, служащего основой карты ссылок, следует стремиться к тому, чтобы границы горячих областей были хорошо очерченными и не пересекались. Кроме того, необходимо позаботиться о том, чтобы пользователь понял, что имеет дело с картой ссылок, а не просто с графическим фоном. Для этого можно использовать поясняющие тексты. При наведении на горячую область указатель мыши изменяет свою форму как и при использовании обычных ссылок. При щелчке на горячей области ее границы становятся видимыми.

Возможны два варианта реализации графических карт ссылок: клиентский и серверный. В клиентском варианте вся информация о конфигурации графической карты ссылок размещается в HTML-документе, в котором она применяется. В серверном варианте предполагается использование сценария, который располагается на сервере и выполняет обработку запросов браузера клиента при работе с графической картой ссылок. Кроме того, на сервере должен находиться конфигурационный файл, в котором описываются горячие области карты. Хотя серверный вариант появился раньше, теперь чаще используется клиентский вариант.

Клиентский вариант графической карты ссылок

Графическая карта ссылок в клиентском варианте задается с помощью нескольких тегов. Первым является контейнерный тег `<map>` (карта) с атрибутом `name` для указания имени карты. Имя карты выбирается как имя переменной. Далее, между тегами `<map>` и `</map>` следуют теги `<area>` (область) для зада-

ния горячих областей. Тег `<area>` имеет ряд атрибутов, описывающих собственно ссылку, а также форму и положение горячей области:

- `href` — строка, определяющая адрес ссылки;
- `shape` — определяет форму области; принимает значения:
  - `"rect"` или `"rectangle"` (прямоугольник);
  - `"poly"` или `"polygone"` (многоугольник);
  - `"circle"` (круг);
- `coords` — координаты области, которые задаются в пикселах перечнем чисел, разделенных запятыми; весь перечень заключается в кавычки (для прямоугольника задаются четыре числа — координаты верхнего левого и правого нижнего угла; для многоугольника указываются координаты каждого угла; для круга задаются три числа — координаты центра и радиус); координаты углов задаются в следующем порядке: сначала горизонтальная, а затем вертикальная координата; начало отсчета координат — верхний левый угол изображения;
- `title` — текстовая строка всплывающей подсказки, появляющейся при наведении указателя мыши на горячую область;
- `target` — указывает окно браузера или фрейм, в котором следует открыть документ, заданный в адресной части ссылки; имеет такие же значения, как и для тега `<a>`;
- `nohref` — атрибут, не имеющий значений; используется для указания, что данная область не имеет адреса перехода. Атрибуты `href` и `nohref` взаимно исключают друг друга. Атрибут `nohref` обычно применяется для того, чтобы исключить переход по ссылке для части уже определенной горячей области.

После закрывающего тега `</map>` следует указать тег, вставляющий изображение и реализующий привязку карты к нему — это уже известный тег `<img>`, в котором помимо прочих возможных атрибутов используется атрибут связи с картой:

usemap="#*имя_карты*"

В качестве имени карты приводится значение атрибута `name` тега `<map>`.

Допустим, в основе графической карты ссылок находится некоторая топографическая карта (файл Карта.jpg). На ней определены прямоугольная и круглая горячие области, соответствующие двум участкам. При щелчке на горячей области в окне браузера будет выведен документ Район1.htm или

Район2.htm, содержащий, например, описания соответствующих участков местности. Вот пример HTML-кода, реализующего эту задачу:

```
<html>
  <head><title>Графическая карта ссылок</title></head>
  <map name="mymap">
    <area href="Район1.htm" shape="rect"   coords="350,50,450,150">
    <area href="Район2.htm" shape="circle" coords="250,250,50">
  </map>
  <img src="Карта.jpg" usemap="#mymap" width=566 height=554>
</html>
```

Графические карты ссылок обычно используются для создания красочных меню. Средства для разработки таких карт предоставляют Macromedia Dreamweaver, Microsoft FrontPage, Adobe Photoshop и некоторые другие пакеты и утилиты.

Серверный вариант графической карты ссылок

В серверном варианте реализации графической карты ссылок при щелчке на горячей области координаты места щелчка передаются на сервер и обрабатываются серверным сценарием. Эта обработка заключается, по существу, в обращении к таблице ссылок и поиску в ней URL-адреса документа, сопоставленного координатам щелчка. Если такой адрес будет найден, то он передается браузеру; в противном случае возвращается сообщение о том, что место щелчка не находится в горячей области. Таблица соответствий координат всех горячих областей карты и URL-адресов документов хранится на сервере в виде так называемого конфигурационного файла с расширением map.

Для применения серверного варианта графической карты ссылок в HTML-документе, загружаемом в браузер клиента, используется тег `<a>`, атрибут `href` которого содержит адрес конфигурационного файла карты ссылок. Внутри тега `<a>` располагается тег `<img>`, указывающий на изображение, которое служит основой карты ссылок, и содержащий атрибут `ismap`:

```
<a href="адрес_конфигурационного_файла"><img
src="адрес_изображения_карты" ismap></a>
```

Пример:

```
<a href=mymap.map"><img src="mymap.jpg" ismap></a>
```

Примечание

Хорошим правилом является сохранение конфигурационного файла и основного графического изображения карты ссылок в одной папке и под единым именем,

но с разными расширениями. Конфигурационный файл имеет расширение map, а изображение карты ссылок сохраняется в любом файле растрового формата, чаще всего GIF, JPEG или PNG.

Конфигурационный map-файл является обычным текстовым файлом, содержащим описание горячих областей графической карты ссылок. Однако форматы представления этих данных могут быть различными. Поэтому перед реализацией серверного варианта карты ссылок следует выяснить у администратора сервера, какой из форматов конфигурационного файла поддерживается. Сейчас существуют форматы конфигурационных файлов CERN и NCSA (аббревиатуры наименований европейского и американского национального научных центров соответственно).

Формат CERN:

тип_области координаты URL-адрес

Тип области:

- `rect` или `rectangle` — прямоугольник;
- `poly` или `polygone` — многоугольник;
- `circle` — круг;
- `default` — по умолчанию (область изображения, не являющаяся горячей).

Координаты *x* и *y* углов прямоугольника и многоугольника или центра круга разделяются запятой и заключаются в круглые скобки.

Примеры:

```
rect (50,100) (150,200) http://www.myweb.ru
poly (20,10) (30,5) (100,25) (50,30) http://www.rambler.ru
circle (100,150) 80 http://www.admiral.ru/hp/dunaev
```

Формат NCSA:

тип_области URL-адрес координаты

Кроме типов областей, принятых в формате CERN, допускается тип `point` (точечная область). Предполагалось, что в случае нескольких точечных горячих областей по щелчку левой кнопкой мыши должна активизироваться та из них, которая ближе к месту щелчка. Однако при наличии областей типа `default` и `point`, последние могут активизироваться только при точном наведении на них указателя мыши, что практически весьма трудно сделать.

Координаты в формате NCSA разделяются запятыми, но не заключаются в скобки. При этом круглая область задается несколько иначе, чем в формате CERN: координатами центра и любой точки, лежащей на окружности.

Примеры:

```
rect http://www.myweb.ru 50,100,150,200
poly http://www.rambler.ru 20,10,30,5,100,25,50,30
circle http://www.admiral.ru/hp/dunaev 100,150,100,70
```

Внутренние ссылки

Ссылки на разделы одного и того же документа, в котором эти ссылки находятся, называются *внутренними* или *закладками*. Например, на своей странице вы размещаете статью, объемом несколько десятков страниц, как единый HTML-документ. Скорее всего, вам захочется сделать ссылки на предыдущие или последующие разделы этого документа. Предметный указатель к книге также можно оформить с помощью внутренних ссылок.

Для организации внутренней ссылки необходимы *якорь* (anchor) и собственно ссылка. Якорь определяет место в документе, к которому происходит переход по ссылке. Ссылка использует имя якоря вместо имени (адреса) файла.

Формат якоря:

```
<a name="имя_якоря">текст</a>
```

Имя якоря может быть произвольным и задается просто как имя переменной. Текст внутри тега `<a>` для создания якоря не является обязательным и чаще всего не указывается.

Формат ссылки:

```
<a href="#имя_якоря">указатель_ссылки</a>
```

Указатель ссылки может быть как текстовым, так и графическим.

В листинге 1.1 приведен пример, демонстрирующий использование внутренних ссылок.

Листинг 1.1. Использование внутренних ссылок

```
<html>
  <head><title>Внутренние ссылки</title></head>
  <body>
    <h2>Содержание</h2>
    <b>
    <a href="#Глава1">Глава 1. Основы HTML</a>
    <br>
    <a href="#Глава2">Глава 2. Примеры</A>
    </b><p>
```

```
    <a name="Глава1"></a><h2>Глава 1.Основы HTML</h2>
    В этой главе мы собираемся рассмотреть основные элементы языка HTML.
    На языке HTML создаются документы, которые могут быть опубликованы
    в Интернете. После изучения основных конструкций языка мы подробно
    рассмотрим примеры программ (<a href="#Глава2">см. главу 2</a>).
   <p>
   <a name="Глава2"></a><h2>Глава 2. Примеры</h2>
   Здесь мы рассмотрим примеры программ с использованием элементов языка,
   которым посвящена
   <a href="#Глава1">Глава 1.</a>
   Прежде всего, рассмотрим применение текстовых ссылок.
  </body>
</html>
```

Адреса ссылок

Адрес на ресурсы Интернета (значение атрибута `href` тега `<a>`) в общем случае задается в виде URL (Uniform Resourse Locator, унифицированный указатель ресурса). URL-адрес имеет следующую структуру:

протокол`://`*адрес_сервера*`/`*адрес_файла*

Адрес файла может быть составным, т. е. кроме имени файла содержать путь к нему; элементы пути разделяются прямым слэшем (наклонной чертой) `/`. Имя файла может быть и не указано. Префикс URL-адреса представляет собой название протокола связи, соответствующий службе Интернета, которая его поддерживает; за названием протокола следует двоеточие и два прямых слэша. В случае URL для электронной почты двойной слэш за двоеточием не указывается. В табл. 1.1 приведены префиксы URL-адресов для основных служб Интернета.

Таблица 1.1. *Наиболее популярные службы Интернета*

Служба в Интернете	Префикс URL-адреса
WWW	**http://**
FTP	**ftp://**
Mail	**mailto:**
Gopher	**gopher://**
UseNet	**news://**
Telnet	**telnet://**

Если вы указываете адрес, начинающийся с **http://**, тем самым вы обращаетесь к ресурсам, доступ к которым осуществляется по протоколу HTTP (HyperText Transfer Protocol, протокол передачи гипертекста). Этот протокол является основным в Интернете при передаче информации, находящейся в HTML-документах. Служба, обеспечивающая обмен HTML-документами, называется *Word Wide Web*.

Префикс адреса **ftp://** означает, что следует использовать протокол передачи файлов (File Transfer Protocol, FTP). Он может применяться при перемещении любых файлов с одного компьютера на другой. В частности, при перекачке файлов вашей страницы на сервер используется именно этот протокол. Протокол FTP обеспечивает высокую надежность передачи файлов. К примеру, если потерю до 10% обычной текстовой информации еще можно пережить, то при передаче программ или сжатых архиватором файлов потери вообще не допустимы — неточно переданная программа просто не будет работать, а архивный файл не распакуется.

Если перед адресом ссылки указывается **mailto:**, это означает, что следует использовать протокол передачи сообщений по электронной почте.

Gopher — служба (значит, и протокол), предназначенная в первую очередь для работы неграфических браузеров. Она предоставляет систему доступа к информации, основанную на меню.

UseNet — служба обеспечения телеконференций, это система типа доски объявлений, на которую вы можете поместить свое сообщение и на которой можно прочитать то, что там разместили другие участники телеконференции.

Ниже приведены примеры ссылок с применением URL-адресов:

```
<a href="http://www.yandex.ru">Яндекс</a>
<a href="http://www.admiral.ru/hp/dunaev">Сам себе Web-дизайнер</a>
<a href="ftp://ftp.admiral.ru/PUB/programs.zip">Бесплатная программа</a>
<a href="mailto:name@anyserver.ru">Отправить почту</a>
```

Пути поиска могут быть абсолютными и относительными. Абсолютный путь описывает местоположение файла, начиная с самого высокого уровня, и включает имена всех папок, ведущих к файлу. Ошибка в записи абсолютного пути (адреса) файла приводит к тому, что файл не будет найден. Относительный путь (адрес) описывает местоположение файла относительно места расположения текущего документа, в котором была инициализирована ссылка с данным адресом. Так, если вы указываете просто имя файла myfile.htm, это означает, что задается относительный адрес. В данном случае браузер будет искать его в той же папке, где находится текущий документ.

Если перед именем файла поставить `../` (например, `../myfile.htm`), то браузер будет искать файл в папке, находящейся на один уровень выше, чем тот, в котором находится текущий документ. Аналогично, если перед именем файла поставить `../../` (например, `../../myfile.htm`), то браузер будет искать файл в папке на два уровня выше, чем текущий.

При создании ссылок вы можете указывать не только на конкретные документы и программы (т. е. конкретные файлы, используя путь к ним), но и на папки (каталоги). Другими словами, адрес — это описание места расположения ресурса (единицы хранения информации). Он может быть точным или "приблизительным" (неполным). Вы можете ссылаться на папку, HTML-документ, документ, созданный каким-либо приложением (например, MS Word, MS Excel), или просто на текстовый файл с расширением txt. Наконец, можно сослаться на файл программы с расширением exe. Однако в последнем случае (по крайней мере, в Internet Explorer) сработает защита вашего компьютера. Появится окно с предупреждением и предложением возможных вариантов: запустить файл программы или сохранить его на диске — способ защиты от потенциальных вирусов. В этом случае вы сами решаете, что делать. Можно сразу запустить файл программы, а можно сохранить его на локальном диске, проверить с помощью антивирусной программы и только потом запустить, вылечить или уничтожить. Если при связи по протоколу HTTP имя файла в URL-адресе не указано, то по умолчанию предполагается файл index.htm или index.html.

Что происходит при инициализации ссылки на некоторый ресурс Интернета? Если запрашивается HTML-документ, то он открывается в окне браузера. Некоторые типы файлов (например, графические файлы форматов GIF, JPEG и др.) браузер также может открыть в своем окне. В ряде случаев для этого требуются специальные подключаемые модули (plug-in). Если браузеру не хватает собственных возможностей, то он пытается открыть файл с помощью ассоциированного с ним приложения, установленного на компьютере пользователя (например, MS Word, MS Excel, Apple QuickTime Player и т. п.).

Ряд файлов (например, с расширениями asp, php, pl и др.) открываются не браузером, а серверной программой, которая их обрабатывает, а браузеру клиента передает лишь результат (обычно в виде html-файла). При инициализации ссылки с адресом электронной почты (`mailto:`*`адрес`*) обычно открывается окно программы почтового клиента (например, Microsoft Outlook Express), в которое указанный адрес получателя вводится автоматически.

При обращении к серверу иногда требуется передать ему некоторые параметры. В этом случае значения параметров следуют за адресом через символ `?`.

Ниже приведен пример графической ссылки на счетчик количества посещений сайта, расположенный на сервере поисковой системы Rambler:

```
<a href="http://top100.rambler.ru/top100/">
<img src=http://counter.rambler.ru/top100.cnt?123456
     alt="Rambler's Top100"
     width=81 height=63 border=0>
</a>
```

Здесь к URL-адресу графического изображения счетчика добавлен идентификационный номер 123456, по которому в базе данных на сервере ищется количество запросов изображения счетчика. Щелчок на изображении счетчика приводит к переходу на страницу Top100 сайта Rambler.

1.2.4. Вставка элементов из внешних источников

Если текст в HTML-документ может быть введен непосредственно, то для вставки на Web-страницу графики, звука, видео и т. п. применяются специальные теги и атрибуты с адресами соответствующих файлов. При загрузке в браузер HTML-документа с такими тегами в него будут загружены и требуемые файлы, если, разумеется, они будут найдены. В данном разделе мы рассмотрим вставку основных элементов, таких как графические изображения, Flash-документы (мультфильмы), звук, видео, и других HTML-документов.

Вставка изображений

Графические изображения из файлов можно вставлять в документ как элемент, занимающий определенную прямоугольную область, и как фон всей страницы.

Вставка отдельных графических изображений

Вставка на страницу изображения из файла графического формата производится с помощью тега `<img>` (от англ. *image* — изображение) с указанием адреса файла в качестве значения обязательного атрибута `src`:

```
<img src="адрес_графического_файла">
```

Адрес графического файла — это либо URL-адрес, либо имя файла, возможно, с указанием пути. Путь может быть абсолютным или относительным. Если в качестве адреса файла указано просто имя файла, то браузер будет искать его в той же папке, в которой расположен вызывающий его HTML-документ.

Примеры:

```
<img src="http://www.myweb.ru/images/picture.jpg">
<img src="C:/myimages/picture.jpg">
<img src="picture.jpg">
```

В теге `<img>` можно использовать атрибут `lowsrc`, который принимает в качестве значения адрес графического файла. Вы можете создать два графических файла: один (например, пусть это файл picture.jpg) содержит изображение, полученное с высоким разрешением, а другой (например, picturelow.gif) — рисунок, полученный с низким разрешением и занимающий меньший объем. Тогда тег

```
<img src="picture.jpg" lowsrc="picturelow.gif">
```

предпишет браузеру сначала загрузить файл picturelow.gif, а затем по мере загрузки страницы заменить его файлом picture.jpg. Этот метод позволяет ускорить формирование страницы в окне браузера.

Для указания размеров изображения на экране предназначены атрибуты `width` (ширина) и `height` (высота), измеряемые в пикселах или процентах. Если указать эти атрибуты, то браузер сначала выделит место под графику, подготовит макет документа, отобразит текст и только потом загрузит картинку. Заметим, что браузер сжимает или растягивает (масштабирует) изображение, встраивая его в прямоугольник указанного размера. Если задан только один из атрибутов, то второй из них рассчитывается браузером так, чтобы сохранить оригинальную пропорцию размеров изображения. Без указания атрибутов `width` и `height` загрузка графики происходит сразу же, как только браузер начинает интерпретировать тег `<img>` и, следовательно, вывод последующего текста и других элементов задерживается. При этом размеры изображения на экране оказываются равными оригинальным. Ниже приведены примеры указания размеров изображения:

```
<img src="picture.jpg" width=40 height=20>
<img src="picture.jpg" width="30%" height="25%">
<img src="picture.jpg" width="140">
```

Чтобы предусмотреть возможность показа документа в режиме отключения загрузки графики, а также в неграфических браузерах, используется атрибут `alt`. Этот атрибут так называемого альтернативного текста принимает в качестве значения текстовую строку, которая будет появляться вместо графического изображения. Обычно в этой строке указывают краткое название изображения и/или размеры графического файла. При наличии текстового описания пользователь сможет сам решить, стоит ли загружать тот или иной графический файл или лучше не тратить на это время. Например:

```
<img src="picture.jpg" alt="Мой портрет 12К">
```

Положение (точнее, выравнивание) изображения относительно других элементов документа можно определить с помощью атрибута выравнивания `align`, который может принимать следующие значения:

- `absbottom` — выравнивание нижней границы изображения по нижней границе текущей строки;
- `absmiddle` — выравнивание середины изображения по середине текущей строки;
- `baseline` — выравнивание нижней границы изображения по базовой линии текущей строки;
- `bottom` — то же, что и `baseline`;
- `top` — верхняя граница изображения выравнивается по самому высокому элементу текущей строки;
- `texttop` — верхняя граница изображения выравнивается по самому высокому текстовому элементу текущей строки;
- `left` — изображение располагается у левого края окна; текст и другие элементы обтекают изображение справа;
- `right` — изображение располагается у правого края окна; текст и другие элементы обтекает изображение слева.

Обычно атрибут `align` тега `<img>` используется для выравнивания изображения относительно текста. Значения `left` и `right` атрибута `align` задают особый тип выравнивания — обтекание изображения текстом.

Для прекращения выравнивания изображений, заданного с помощью `align=left` или `align=right`, относительно текста можно использовать тег `<br>` перевода строки с атрибутом `clear`. Этот атрибут может принимать три значения:

- `left` — для прекращения обтекания текстом элементов (изображений), выровненных по левому краю;
- `right` — для прекращения обтекания текстом элементов (изображений), выровненных по правому краю;
- `all` — для прекращения обтекания текстом элементов (изображений), выровненных либо по левому, либо по правому краю.

По умолчанию изображение встраивается почти вплотную с текстом или другими элементами документа, что может быть некрасиво. Чтобы этого избежать, можно задать пустые поля вокруг иллюстрации. Поля создаются с помощью атрибутов `vspace` для верхнего и нижнего полей и `hspace` для бо-

ковых полей в теге `<img>`. Значения этих атрибутов указываются в виде целых чисел, определяющих размеры полей в пикселах. Например:

```
<img src="picture.jpg" align=left hspace=20 vspace=10>
```

Для задания черной рамки вокруг изображения служит атрибут `border`, числовое значение которого определяет толщину рамки в пикселах. Если этот атрибут не указан, то рамка не отображается.

Примечание

Если тег `<img>`, задающий изображение, вставить в контейнерный тег ссылки `<a>`, то получится так называемая *графическая ссылка (баннер)*. Браузер по умолчанию выделяет ссылки рамками вокруг их графических изображений. Чтобы исключить отображение рамки, достаточно использовать в теге `<img>` атрибут `border=0`.

Фоновая графика

Фон клиентской области окна браузера можно заполнить изображением из графического файла. Фоновая графика задается в теге `<body>` с помощью атрибута `background`, значением которого является URL-адрес или имя графического файла. Например:

```
<body background="myfon.gif">
```

Для браузера Internet Explorer можно указать дополнительный атрибут `bgproperties=fixed`, запрещающий прокрутку фонового изображения. Например:

```
<body background="myfon.gif" bgproperties=fixed>
```

В качестве фонового изображения обычно используется небольшой и простой рисунок (так называемая *текстура*), который многократно выводится на экран, заполняя все окно. Этот рисунок может представлять собой небольшой прямоугольник или же длинную узкую полоску (например, с градиентной заливкой). Такие изображения, подобно листам обоев, покрывают все окно браузера, оставляя незаметными стыки. Кроме того, поскольку изображение небольшое (не более нескольких килобайтов), создание фона происходит быстро.

В качестве фона также можно использовать один экземпляр изображения, например фотографию. Поскольку размеры окна браузера и монитора могут быть самыми различными, то размеры фонового изображения должны соответствовать типичному или даже крайнему случаю. Например, если типичными являются разрешение монитора 1024×768 пикселов и полностью раскрытое окно браузера, то фоновое изображение должно иметь

размеры 1024×768 пикселов или больше. Однако при большем разрешении монитора в полностью открытом окне браузера будут видны 4 экземпляра этого же изображения, что обычно связано с проблемой их стыковки. Следует также иметь в виду, что файлы изображений с большим разрешением и сложные по содержанию имеют большой объем, что не вполне годится для использования в Web-приложениях.

Примечание

Создавать графический фон не только для документа, но и для отдельных его элементов (например, кнопок, таблиц, абзацев и др.) позволяют каскадные таблицы стилей.

Вставка Flash-документов

Flash-документ (мультфильм, ролик, клип) может включать разнородное содержимое: текст, растровую и векторную статическую и анимационную графику, звук, видео и элементы пользовательского интерфейса. Вы можете создать отдельную страницу или даже целый сайт, используя технологию Flash. Flash-документы разрабатываются с помощью пакета Macromedia Flash, выходные файлы которого имеют расширение swf. Просмотр swf-файлов производится с помощью специального Flash-проигрывателя. Чтобы просмотреть swf-файл в окне браузера MS Internet Explorer, требуется элемент управления ActiveX, реализующий функции Flash-проигрывателя, ссылка на который вставляется в HTML-документ с помощью тегов `<object>` и `<param>`:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
        codebase=
"http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab
        #version=7,0,0,0"
        id="myflash">
  <param name="movie" value="URL-адрес swf-файла">
</object>
```

Здесь атрибут `classid` принимает в качестве значения уникальный регистрационный идентификатор объекта, соответствующего Flash-проигрывателю. Атрибут `codebase` содержит URL-адрес исходных файлов Flash-проигрывателя версии 7.0 на случай, если он не установлен на компьютере пользователя. Тег `<param>` служит для указания URL-адреса swf-файла, содержащего собственно Flash-документ, который должен быть воспроизведен Flash-проигрывателем. Атрибут `id` указан только для обеспечения возможности доступа к объекту с помощью сценария.

Чтобы Flash-проигрыватель был вставлен в Web-страницу браузерами, не воспринимающими теги `<object>` и `<param>` (например, Netscape Navigator), внутри контейнера `<object>` используют тег `<embed>`:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
        codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab
        #version=7,0,0,0">
  <param name="movie" value="URL-адрес swf-файла">
  <embed src="URL-адрес swf-файла" name="myflash"
         type="application/x-shockwave-flash"
         pluginspage="http://www.macromedia.com/go/getflashplayer">
</object>
```

Здесь в теге `<embed>` атрибут `name` играет такую же роль, что и `id` в `<object>`, а атрибут `pluginspage` — такую же роль, что и `codebase`.

Кроме рассмотренных выше, можно использовать множество других параметров Flash-проигрывателя, среди которых особо отметим наиболее важные, задаваемые с помощью тегов `<param>`:

- прозрачность фона Flash-документа (важно при согласовании с фоном HTML-документа):
  - в теге `<object>` используется тег:

    ```
    <param name="wmode" value="transparent">
    ```

  - в теге `<embed>` используется атрибут:

    ```
    wmode="transparent"
    ```

- цвет фона Flash-документа:
  - в теге `<object>` используется тег:

    ```
    <param name="bgcolor" value="цвет">
    ```

  - в теге `<embed>` используется атрибут:

    ```
    bgcolor="цвет"
    ```

    Здесь *цвет* задается именем или шестнадцатеричным кодом, например, `#f0f0f0`;

- качество воспроизведения Flash-документа:
  - в теге `<object>` используется тег:

    ```
    <param name="quality" value="уровень_качества">
    ```

  - в теге `<embed>` используется атрибут:

    ```
    quality= "уровень_качества"
    ```

Здесь *уровень_качества* принимает значения: low (низкое), medium (среднее), high (высокое), best (самое лучшее), autolow (автонизкое), autohigh (автовысокое).

В пакете Macromedia Flash имеются средства автоматизации создания HTML-документа, содержащего заданный Flash-документ, которые позволяют легко установить все необходимые параметры.

Вставка звука и видео

Современные браузеры могут воспроизводить видео- и звуковые файлы различных форматов. Для этого они используют встроенные проигрыватели (plug-in, элементы управления ActiveX) или внешние программы-проигрыватели. Вставить звук или видео в HTML-документ можно с помощью различных тегов:

- <bgsound src=*адрес_файла*> — для вставки фонового звука;
- <img dynsrc=*адрес_файла*> — для вставки видео в формате AVI;
- <a href=*адрес_файла*> — для вставки звуковых и видеофайлов;
- <embed src=*адрес_файла*> — для вставки звуковых и видеофайлов.

При решении вставить звук и/или видео в HTML-документ следует учитывать, что соответствующие файлы имеют довольно большой объем. Наиболее популярными в Web сейчас являются файлы звуковых форматов MP3, WMA, AIFF, AU, RealAudio (с расширениями ra и ram), MP4, MIDI и видеоформатов MPEG, MOV. Звуковой формат WAV и видеоформат AVI в Интернете используются довольно редко.

Для операционной системы Windows стандартными звуковыми файлами являются WMA и WAV, а для Macintosh — AIFF. Общепринятым видеоформатом для Windows, Macintosh и UNIX является MPEG. Формат QuickTime (MOV) также широко распространен и может использоваться Windows и Macintosh. При выборе других форматов желательно предупреждать об этом пользователей.

Фоновый звук

Тег <bgsound> позволяет указать звуковой файл формата WMA, AU, MIDI или WAV, который будет проигрываться при загрузке страницы. При этом на экран не выводится панель управления воспроизведением.

Тег <bgsound> имеет следующие атрибуты:

- src — URL-адрес звукового файла;

- `loop` — определяет, сколько раз звуковой файл должен быть воспроизведен; принимает значения:
  - `-1` или `infinite` — воспроизведение в течение всего времени показа документа;
  - *число* — количество воспроизведений;
- `balance` — число, определяющее баланс для стереозвука;
- `volume` — число, определяющее громкость звучания.

Пример:

```
<bgsound src="symphony.wma" loop=3>
```

Применение тега *<img>* и ссылок

Тег `<img>` применяется в основном для вставки графических изображений, адрес файла которых указывается атрибутом `src`. Однако этот тег имеет еще и атрибут `dynsrc`, с помощью которого можно указать адрес видеофайла формата AVI. Если указаны оба атрибута, `dynsrc` и `src`, то действует `dynsrc`.

Атрибуты, определяющие воспроизведение видеофайла, таковы:

- `dynsrc` — URL-адрес видеофайла формата AVI (расширение avi);
- `loop` — определяет, сколько раз звуковой файл должен быть воспроизведен; принимает значения:
  - `-1` или `infinite` — воспроизведение в течение всего времени показа документа;
  - *число* — количество воспроизведений;
- `start=mouseover` — выводится изображение первого кадра, а воспроизведение начинается только при наведении на него указателя мыши и продолжается далее заданное атрибутом `loop` количество раз.

Пример:

```
<img dynsrc="myvideo.avi" loop=2 start=mouseover>
```

Можно также использовать и другие атрибуты тега `<img>`, например, `align`, `border`, `hspace` и `vspace`.

Для воспроизведения звуковых и видеофайлов можно использовать тег ссылки `<a>`, указав в качестве значения атрибута `href` URL-адрес файла. При этом файл будет воспроизводиться в окне приложения (проигрывателя), которое ассоциировано с данным типом файла и установлено на компьютере пользователя. Это может быть, например, Windows Media Player, панель **Медиа** (в Internet Explorer), Winamp, QuickTime Player или RealPlayer G2.

URL-адрес файла RealAudio имеет вид: `pnp://адрес_файла`. Технология RealAudio обеспечивает передачу звука и видео практически в режиме реального времени и позволяет устраивать аудио- и видеоконференции. Звуковые файлы RealAudio имеют расширение ra. Кроме того, могут использоваться метафайлы с расширением ram. Это обычные текстовые файлы, содержащие полные URL-адреса файлов с расширением ra.

Применение тега *<embed>*

Тег `<embed>` позволяет встроить в Web-страницу документы различных типов для просмотра их соответствующим приложением. При встраивании звука и видео в окне браузера появляется панель того проигрывателя, который ассоциирован с данным типом файлов. Обратите внимание, что панель проигрывателя встраивается в документ, отображаемый в окне Web-браузера, а не появляется в виде автономного окна, как в случае использования ссылок.

В следующем примере с помощью тега `<embed>` встраиваются три звуковых и один видеофайл. Размеры панелей воспроизведения можно задать с помощью атрибутов `width` и `height`.

```
<html>
  <head><title>Звук и видео</title></head>
  Песня        <embed sr=song.mp3><p>
  Симфония № 5 <embed src=Symphony5.wav width=80 height=150><p>
  Симфония № 9 <embed src=Symphony9.wma><p>
  Видео        <embed src=examople.avi width=80 height=120>
</html>
```

Тег `<embed>` имеет следующие атрибуты, связанные с воспроизведением звука и видео:

- `src` — URL-адрес файла;
- `align` — горизонтальное выравнивание панели, принимает следующие значения:
  - `absbottom` — выравнивание нижней границы панели по нижней границе текущей строки;
  - `absmiddle` — выравнивание середины панели по середине текущей строки;
  - `baseline` — выравнивание нижней границы панели по базовой линии текущей строки;
  - `bottom` — то же, что и `baseline`;
  - `top` — верхняя граница панели выравнивается по самому высокому элементу текущей строки;

- texttop — верхняя граница панели выравнивается по самому высокому текстовому элементу текущей строки;
- left — панель располагается у левого края окна, текст и другие элементы обтекают его справа;
- right — панель располагается у правого края окна, текст и другие элементы обтекают его слева;

❒ alt — альтернативный текст;

❒ height — высота в пикселах или процентах;

❒ width — ширина в пикселах или процентах;

❒ hspace — боковые отступы в пикселах;

❒ vspace — верхний и нижний отступы в пикселах;

❒ name — имя (идентификатор) встроенного элемента.

Примечание

Кроме перечисленных, тег <embed> имеет еще атрибуты code и codebase, имеющие такой же смысл, что и для тега <object> *(см. следующий раздел)*.

Тег *<object>*

Тег <object> позволяет вставлять в HTML-документы графические изображения, HTML-документы, элементы управления ActiveX и апплеты.

Тег <object> является контейнерным и обладает множеством атрибутов, список которых приведен ниже.

❒ accesskey — определяет клавишу быстрого доступа к объекту, встроенному с помощью тега <object>. Значением атрибута является строка, содержащая букву или цифру. Если нажать указанную клавишу в сочетании с клавишей <Alt>, то объект получит фокус.

❒ align — выравнивание; возможны следующие значения:

- absbottom — выравнивает нижнюю границу объекта по подстрочной линии символов окружающего текста;
- absmiddle — выравнивает середину объекта по центральной линии между top и absbottom окружающего текста;
- baseline — выравнивает нижнюю границу объекта по базовой линии окружающего текста;
- bottom — совпадает с baseline (только в Internet Explorer);
- left — выравнивает объект по левому краю элемента-контейнера;

- `middle` — выравнивает воображаемую центральную линию окружающего текста по воображаемой центральной линии объекта;
- `right` — выравнивает объект по правому краю элемента-контейнера;
- `texttop` — выравнивает верхнюю границу объекта по надстрочной линии символов окружающего текста;
- `top` — выравнивает верхнюю границу объекта по верхней границе окружающего текста.

- `classid` — уникальный регистрационный идентификатор встраиваемого компонента.

  Пример:

  ```
  classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  ```

- `code` — имя файла Java (класса апплета), если он является источником.
- `codebase` — базовый URL-адрес, по которому браузер может взять файл (обычно — упакованный) компонента, чтобы установить его при отсутствии на компьютере пользователя. Вслед за URL-адресом указывается версия компонента в виде строки: `#Version=M1,M2,m1,m2`. Здесь `M1` и `M2` — максимальные, а `m1` и `m2` — минимальные мажорный и минорный номера версии, доступные на сервере. Встречая в HTML-документе ссылку на компонент, браузер проверяет его наличие на компьютере пользователя. Если компонента нет, то производится его загрузка и установка, с разрешения пользователя или без него в зависимости от настроек безопасности браузера. Если на компьютере пользователя установлена более новая версия, чем указанная атрибутом `codebase`, то загрузка компонента не производится.

  Пример:

  ```
  codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab
            #version=7,0,0,0"
  ```

  Здесь указан URL-адрес упакованного cab-файла Flash-проигрывателя единственной версии 7.

- `codetype` — MIME-тип файлов, указанных атрибутом `codebase`. Например, для элементов управления ActiveX значением этого атрибута (если он указан) является `application/x-oleobject`.
- `data` — URL-адрес файла данных (для объектов, работающих с данными).
- `datafld` — имя столбца табличных данных.
- `datasrc` — "#*идентификатор_источника_данных*".

- disabled — если указан, то объект не доступен.
- height — высота в пикселах визуального образа компонента (если таковой имеется).
- width — ширина в пикселах визуального образа компонента (если таковой имеется).
- hspace — расстояние по горизонтали между объектом и соседними элементами.
- vspace — расстояние по вертикали между объектом и соседними элементами.
- id — идентификатор объекта как элемента HTML-документа, с помощью которого обеспечивается доступ к свойствам объекта из сценария.
- name — идентификатор объекта как элемента формы; используется в случае, когда объект находится в контейнере <form>, а его данные требуется передать на сервер щелчком на кнопке типа submit.
- type — MIME-тип файлов, указанных атрибутом data.
- tabindex — целое число, определяющее порядок перехода к элементу с помощью клавиши <Tab>.
- lang — определяет тип языка для событий контейнера, например, script/javascript.
- language — определяет имя языка для событий контейнера. Возможные значения: javascript, jscript, vbscript и vbs.

Примечание

Кроме перечисленных выше, тег <object> имеет атрибуты class и style (для задания параметров каскадной таблицы стилей), title (для задания текста всплывающей подсказки) и атрибуты-события (для связи с их обработчиками).

Встраиваемый компонент может иметь параметры, которые задаются с помощью тегов <param>, вставляемых в контейнер <object>. Для каждого параметра требуется отдельный тег <param>. Этот тег имеет следующие атрибуты:

- name — имя параметра;
- value — значение параметра;
- datafld — имя столбца табличных данных;
- dataformatas — формат данных;
- datasrc — "*#идентификатор_источника_данных*".

Примеры использования тега `<param>` были рассмотрены в *разд. "Вставка Flash-документов"* ранее в этой главе.

Несмотря на обилие различных атрибутов тега `<object>`, на практике используется лишь небольшое их подмножество, определяемое особенностями конкретного встраиваемого компонента.

Кроме элементов ActiveX и апплетов, с помощью тега `<object>` можно вставлять в Web-страницы графические изображения из файлов формата GIF и JPEG, звуковые файлы, а также HTML-документы. Для этого необходимо использовать атрибуты `data` и `type`. Значением атрибута `data` является URL-адрес вставляемого графического или html-файла. Значением атрибута `type` является тип данных в этом файле: `image/jpeg` или `image/gif` для графического и `text/html` — для html-файла.

Вставляемые объекты указанных выше типов отображаются в отдельных окнах, подобных плавающим фреймам *(см. разд. 1.2.5)* и снабженных при необходимости полосами прокрутки. Размеры этих окон устанавливаются с помощью атрибутов `width` и `height`. В отличие от тега `<img>`, специально предназначенного для вставки графики, эти атрибуты не влияют на размеры собственно графического изображения (не масштабируют его). Они задают размеры только окна отображения графики.

Примечание

Для корректного отображения в окне браузера графики и HTML-документов, вставляемых посредством тегов `<object>`, последние либо должны находиться в контейнере `<body>`, либо исходный HTML-документ должен содержать, кроме `<object>`, еще какой-нибудь элемент, способный принять фокус (например, текст).

1.2.5. Разметка страницы

Основными средствами HTML для разметки страницы или композиции Web-страницы являются таблицы и фреймы. Ячейки таблицы создаются контейнерными тегами и, следовательно, могут содержать другие элементы: тексты, графику, элементы пользовательского интерфейса и даже другие таблицы. Некоторые или все разграничительные линии таблицы можно сделать невидимыми. Таким образом, таблицы можно рассматривать как средство позиционирования элементов документа. Фреймы представляют собой объекты, во многом похожие на главное окно браузера. Вы можете разбить клиентскую область браузера на несколько прямоугольных областей (фреймов), загрузив в каждую из них свой HTML-документ.

Таблицы

Данные в таблице организованы в виде прямоугольной сетки, которая состоит из вертикальных колонок (столбцов) и горизонтальных рядов (строк). Таблицы состоят из четырех основных частей: названия таблицы, заголовков столбцов, ячеек с данными и нижней части. Таблица заполняется слева направо, ячейка за ячейкой и строка за строкой, начиная с левого верхнего угла и заканчивая правым нижним углом. Впрочем, таблица может и не иметь названия, заголовков столбцов и нижней части.

Теги таблицы

Все теги формирования таблицы являются контейнерными. Вот их список:

- `<table>` — контейнер для всех других тегов таблицы;
- `<td>` — контейнер для данных, содержащихся в ячейке таблицы;
- `<tr>` — контейнер для ячеек, составляющих одну строку таблицы;
- `<th>` — контейнер для данных, содержащихся в ячейке, используемой как заголовок столбца;
- `<thead>` — контейнер для набора строк, используемых в качестве заголовка таблицы и располагающихся выше ее основной части;
- `<tbody>` — контейнер для набора строк основной части таблицы, обычно располагающейся ниже ее заголовка;
- `<tfoot>` — контейнер для набора строк, располагающихся ниже основной части таблицы; здесь обычно находятся пояснительные подписи и примечания;
- `<caption>` — контейнер для названия таблицы;
- `<col>` — контейнер для задания параметров выравнивания столбцов;
- `<colgroup>` — контейнер для группы столбцов.

Для формирования таблицы тег `<table>` обязателен. Остальные теги используются по желанию разработчика. В большинстве случаев можно обойтись только тегами `<tr>` для строк и `<td>` для ячеек. По умолчанию таблица представляется в окне браузера без рамок. Чтобы отобразить рамки, в теге `<table>` следует использовать атрибут `border`. Для задания толщины внешней рамки в пикселах можно указать целочисленное значение для этого атрибута.

Для задания названия таблицы предназначен тег `<caption>`. Каждый ряд (строка) ячеек начинается с тега `<tr>` и заканчивается тегом `</tr>`. Если ряд должен содержать заголовки столбцов таблицы, то используется тег `<th>`.

Если в ячейках должны размещаться данные, то применяется тег `<td>`. Теги заголовков столбцов и ячеек для данных должны располагаться в теге `<tr>`. Проще говоря, вы сначала определяете таблицу (тег `<table>`), а затем внутри этого определения задаете строки (тег `<tr>`) и содержимое ячеек (тег `<td>` — для данных, тег `<th>` — для названий заголовков столбцов).

В листинге 1.2 представлен HTML-код, создающий простую таблицу.

Листинг 1.2. Пример простой таблицы

```
<html>
  <head><title>Таблицы</title></head>
  <table border=5>
    <caption>Список сотрудников</caption>
    <tr>
      <th>Имя</th><th>Должность</th><th>Оклад</th>
    </tr>
    <tr>
      <td>Иван</td><td>Директор</td><td>1000</td>
    </tr>
    <tr>
      <td>Антон</td><td>Бухгалтер</td><td>870</td>
    </tr>
    <tr>
      <td>Федор</td><td>Водитель</td><td>500</td>
    </tr>
  </table>
</html>
```

Текст в теге `<th>` заголовка столбца выводится жирным шрифтом.

Теги `<col>` и `<colgroup>`, позволяющие задать общие параметры выравнивания столбцов, будут рассмотрены позже.

Параметры таблицы

Для управления структурой и внешним видом таблицы, а также выравниванием данных в ней используются атрибуты и специальные теги. В табл. 1.2 указано, какими атрибутами обладает тот или иной тег.

Таблица 1.2. *Соответствие между тегами и атрибутами*

Атрибуты	<table>	<td>	<tr>	<th>	<thead>	<tbody>	<tfoot>	<caption>
align	+	+	+	+	+	+	+	+
background	+	+		+				
bgcolor	+	+	+	+	+	+	+	
border	+							
bordercolor	+	+	+	+				
bordercolordark	+	+	+	+				
bordercolorlight	+	+	+	+				
cellpadding	+							
cellspacing	+							
cols	+							
colspan		+		+				
datapagesize	+							
datasrc	+							
frame	+							
height	+							
nowrap		+		+				
rowspan		+						
rules	+							
valign		+	+	+	+	+	+	+
width	+							

Толщина и цвет рамок

Атрибут `border` тега `<table>` принимает в качестве значения число, определяющее толщину внешней рамки таблицы в пикселах. Если этот атрибут не указан или его значение равно 0, то рамки не отображаются. Допускается не указывать значение атрибута, и тогда рамка будет иметь толщину 1. Каждая ячейка имеет собственную рамку толщиной 1 пиксел, которая отображается, если в теге `<table>` указан атрибут `border` с отличным от нуля значением или без значения. Толщиной рамок ячеек управлять нельзя.

Цвет рамки определяется атрибутами:

- `bordercolor` — цвет всей рамки;
- `bordercolordark` — цвет нижней и правой частей внешней рамки таблицы; цвет верхней и левой частей рамки ячейки;

- `bordercolorlight` — цвет верхней и левой частей рамки таблицы; цвет нижней и правой частей рамки ячейки.

Перечисленные выше атрибуты применимы к тегу `<table>` и определяют цвет внешней рамки таблицы, а также к тегам `<tr>`, `<td>` и `<th>`. В последнем случае они задают цвет рамок ячеек.

Чтобы назначить одинаковый цвет для всех частей рамки, используется атрибут `bordercolor`. Для создания эффекта объемности рамки служат атрибуты `bordercolordark` и `bordercolorlight`. При этом для внешней рамки и рамок ячеек они задают цвет противоположным образом.

Значения атрибутов цвета задаются шестнадцатеричным числом в соответствии с моделью RGB или именем цвета (допустимые имена предопределены). Например, `bordrcolor=#ff0000` или `bordercolor=red`.

Выборочное отображение рамок

Если в теге `<table>` указан атрибут `border` без значения или с отличным от 0 значением, то по умолчанию таблица отображается с внешней рамкой, а каждая ячейка имеет собственную рамку. С помощью атрибута `frame` можно дополнительно указать, какие части внешней рамки отображать, а какие нет. Атрибут `rules` позволяет определить отображение внутренних разделительных линий между столбцами и строками таблицы.

Атрибут `frame` может принимать следующие значения:

- `box` — отображать все четыре стороны рамки;
- `above` — отображать только верхнюю часть рамки;
- `below` — отображать только нижнюю часть рамки;
- `hsides` — отображать горизонтальные части рамки сверху и снизу;
- `vsides` — отображать только левую и правую вертикальные части рамки;
- `lhs` — отображать только левую часть рамки;
- `rhs` — отображать только правую часть рамки;
- `void` — не отображать внешнюю рамку.

Пример:

```
<table border=5 frame=hsides>
```

Атрибут `rules` может принимать следующие значения:

- `all` — отображать все вертикальные и горизонтальные линии;
- `rows` — отображать только горизонтальные линии между строками;

- `cols` — отображать только вертикальные линии между столбцами;
- `none` — не отображать разделительные линии.

Пример:

```
<table border=5 rules=cols>
```

Расстояние между ячейками и отступы внутри ячеек

По умолчанию расстояние между рамками смежных ячеек таблицы устанавливается равным 2 пикселам. Чтобы задать требуемое значение, в теге `<table>` можно использовать атрибут `cellspacing`, принимающий целочисленные значения. В частности, если установить значение 0, то рамки смежных значений сольются и будут выглядеть как разделительные линии.

Пример:

```
<table border=5 cellspacing=5>
```

Для задания отступов (полей) сверху и с боков внутри ячеек используется атрибут `cellpadding`, принимающий целочисленные значения, определяющие ширину отступа в пикселах. По умолчанию (когда атрибут не указан) ширина отступов равна 1.

Пример:

```
<table border=5 cellpadding=5>
```

Если требуется задать оба атрибута, `cellpadding` и `cellspacing`, то вопреки общему правилу указания атрибутов в произвольном порядке необходимо сначала указать `cellpadding`.

Пример:

```
<table border=5 cellpadding=10 cellspacing=5>
```

Расширение ячеек

Сложные по структуре таблицы получаются путем расширения ячеек, которое можно производить как по вертикали, так и по горизонтали. Для этой цели в тегах `<td>` и `<th>` служат атрибуты:

- `colspan` — расширение ячеек по горизонтали (столбцам);
- `rowspan` — расширение ячеек по вертикали (строкам).

Эти атрибуты принимают целочисленные значения, которые указывают, на сколько ячеек требуется расширить данную ячейку. Пример представлен в листинге 1.3.

Листинг 1.3. Расширение ячеек

```
<html>
  <head><title>Таблицы</title></head>
  <table border=5 cellpadding=5 cellspacing=0>
    <caption><b>Список сотрудников</b></caption>
    <tr>
      <th colspan=2>Имя и Фамилия</th><th>Должность</th><th>Оклад</th>
    </tr>
    <tr>
      <td>Иван</td><td>Иванов</td><td>Директор</td><td>1000</td>
    </tr>
    <tr>
      <td>Антон</td><td>Антонов</td><td>Бухгалтер</td><td>870</td>
    </tr>
    <tr>
      <td >Федор</td><td>Федоров</td>
      <td rowspan=3>Водитель</td><td>500</td>
    </tr>
    <tr>
      <td>Петр</td><td>Петров</td><td>450</td>
    </tr>
    <tr>
      <td>Павел</td><td>Павлов</td><td>350</td>
    </tr>
  </table>
</html>
```

Размеры таблицы

Размеры таблицы можно регулировать с помощью атрибутов `width` (ширина) и `height` (высота) тега `<table>`. Их значения задаются в пикселах или процентах. Числовое значение без символа `%` означает количество пикселов.

Примеры:

```
<table width=200 height=400>
<table width=75% height=450>
<table width=75% height=50%>
```

Если атрибуты `width` и `height` не указаны, то при отображении таблицы браузер автоматически вычисляет ее размеры в зависимости от многих пара-

метров, таких как размер используемого шрифта, размеры окна просмотра, заданных значений других атрибутов тегов таблицы и т. п. Алгоритмы представления таблиц в различных браузерах могут отличаться друг от друга, что при прочих равных условиях сказывается на результатах отображения в окне просмотра.

Если атрибуты `width` и/или `height` указаны, то их значения будут использованы браузером как рекомендуемые. Это означает, что браузер может попытаться выдержать данные размеры, изменив, например, размеры ячеек или расположение текста в них.

Пусть задана таблица без атрибута `width`. Тогда браузер автоматически вычислит некоторое значение w_1 ее ширины. Теперь добавим в тег `<table>` этой же таблицы атрибут `width` с некоторым значением w_2. Если $w_1 < w_2$, то браузер установит значение w_2, пропорционально расширив все столбцы. Если $w_1 > w_2$, то браузер попытается установить ширину таблицы, возможно более близкую к величине w_2. При этом точное значение w_2 может быть и не достигнуто. Например, для всех рассмотренных ранее таблиц установка атрибута `width=10` не изменит их ширину.

Примечание

По спецификации HTML 4.0 атрибутами `width` и `height` обладает только тег `<table>`. Однако браузер Internet Explorer интерпретирует эти атрибуты и в других тегах таблицы, позволяя устанавливать размеры отдельных ячеек. Поскольку таблица имеет сильно связанную структуру, задание размеров для одной ячейки влияет на размеры других ячеек, а также может изменить размеры таблицы в целом.

В теге `<table>` можно использовать атрибут `cols` для указания количества столбцов (колонок) таблицы. Однако в действительности количество столбцов определяется как максимальное количество ячеек в одной строке и не зависит от значения атрибута `cols`. Вместе с тем, указание атрибута `cols` информирует браузер о структуре таблицы до того, как она была проанализирована им в деталях, и, следовательно, должно ускорить процесс отображения. Разумеется, значение атрибута `cols` должно соответствовать действительному количеству столбцов.

Выравнивание

Горизонтальное выравнивание таблицы относительно окна браузера производится с помощью атрибута `align` тега `<table>`. Этот атрибут принимает значения `center`, `left` и `right` (по центру, левому и правому краю соответственно). По умолчанию таблица выравнивается по левому краю окна браузера.

Если атрибут `align` не указан, то все последующие элементы HTML-документа будут расположены ниже таблицы (если только они не позиционированы иначе с помощью каскадных таблиц стилей). Если же использовать этот атрибут со значением `left` или `right`, то можно организовать обтекание таблицы текстом и даже графическими изображениями соответственно справа и слева, подобно тому, как это делается с графическими изображениями.

Дополнительно можно использовать тег `<br clear=...>` для более точного управления обтеканием, аналогично обтеканию изображений текстом. Атрибут `clear` может принимать значения:

- `left` — для прекращения обтекания текстом элементов (изображений), выровненных по левому краю;
- `right` — для прекращения обтекания текстом элементов (изображений), выровненных по правому краю;
- `all` — для прекращения обтекания текстом элементов (изображений), выровненных или по левому, или по правому краю.

Использование атрибута `align` в других тегах таблицы позволяет выровнять по горизонтали содержимое ячеек. Заданное выравнивание действует только на одну ячейку или группу ячеек в зависимости от того, в каком теге используется этот атрибут. Например, если требуются различные выравнивания для заголовков столбцов и основной части таблицы, то можно задать различные значения атрибута `align` в тегах `<thead>` и `<tbody>`.

Для вертикального выравнивания содержимого ячеек предназначен атрибут `valign`, принимающий следующие значения:

- `baseline` — по базовой линии;
- `bottom` — по нижнему краю;
- `middle` — по середине (значение по умолчанию);
- `top` — по верхнему краю.

Значение `baseline` задает привязку содержимого ячеек одного ряда к одной базовой линии.

Часто требуется запретить перенос слов на другую строку в ячейках с текстовым содержимым. Например, нежелательны разрывы между фамилией и инициалами, числом и указанием единицы измерения или названия месяца и т. п. По умолчанию перенос на другую строку производится в месте пробела. Чтобы избежать этого, вместо пробела следует использовать числовой или буквенный эквивалент символа пробела ` ` или ` `. Например, строку "250 руб." следует записать как `"250 руб."`. Другой способ запретить перенос слов заключается в добавлении в теги `<td>` и `<th>` атрибута `nowrap`, который не имеет значений.

Для задания параметров выравнивания группы столбцов таблицы можно использовать контейнерный тег `<colgroup>` и обычный тег `<col>`. Эти теги записываются после тега `<table>` и перед первым тегом `<tr>`.

Теги `<colgroup>` и `<col>` имеют следующие атрибуты:

- `align` — горизонтальное выравнивание `center` (по центру), `left` (по левому краю) и `right` (по правому краю);
- `span` — количество смежных столбцов, имеющих одинаковые параметры выравнивания; значение по умолчанию равно 1;
- `valign` — вертикальное выравнивание содержимого ячеек; принимает значения `baseline`, `bottom`, `middle` и `top`;
- `width` — ширина столбца.

Пример:

```
<table>
  <colgroup>
    <col align=left span=2>
    <col align=right>
  </colgroup>
  <tr>
  <! Данные таблицы >
  ...
</table>
```

Здесь содержимое ячеек первых двух столбцов выравнивается по левому краю, а третьего столбца — по правому краю.

Параметры таблицы, устанавливаемые тегами `<col>` и `<colgroup>`, могут быть изменены с помощью атрибутов выравнивания в основных табличных тегах, используемых после тега `<table>` (`<tr>`, `<td>` и `<th>`).

Цвет и фон

Цвет фона всей таблицы или отдельных ячеек устанавливается с помощью атрибута `bgcolor`. Значение этого атрибута задается шестнадцатеричным числом в соответствии с моделью RGB или именем цвета (допустимые имена предопределены). Например, `bgcolor=#0000ff` или `bgcolor=blue`.

Фон всей таблицы или отдельных ячеек можно заполнить графическим изображением. Для этого предназначен атрибут `background`, принимающий в качестве значения URL-адрес графического файла. Заполнение таблицы в целом или только некоторых ячеек происходит аналогично заполнению

фона всего документа с помощью одноименного атрибута в теге `<body>`. Изображение, используемое в качестве фона, не масштабируется в соответствии с размерами элемента (таблицы или ячейки), а размеры элемента автоматически не сообразуются с размерами изображения. Если изображение больше элемента, то будет видна лишь некоторая его часть, а если меньше, то элемент будет заполнен несколькими экземплярами изображения.

Если в одном и том же табличном теге используются одновременно два атрибута, `bgcolor` и `background`, то заполнение фона происходит графическим изображением.

Если в тегах таблицы не указаны ни атрибут `bgcolor`, ни атрибут `background`, то фон таблицы считается прозрачным. В этом случае фон таблицы определяется фоном документа, заданным одноименными атрибутами в теге `<body>`. Более того, если применяется наложение (с помощью стилей) таблицы на другие элементы документа, то последние будут видны сквозь таблицу, собственный фон которой не задан.

Ячейки таблицы задаются контейнерными тегами, и, следовательно, в них можно вставлять другие теги. В частности, в ячейках можно размещать изображения. При этом вставленное с помощью тега `<img>` изображение отображается одним своим полным экземпляром, а не заполняет ячейку подобно обоям, как при использовании атрибута `background`.

Фреймы

Окно просмотра браузера можно разбить на несколько прямоугольных областей, каждая из которых функционирует подобно окну браузера. Эти области называются *фреймами*. Они создаются с помощью тегов `<frameset>` и `<frame>`. Это так называемые обыкновенные фреймы. Кроме того, с помощью тега `<iframe>` можно создать так называемые плавающие фреймы. Во фреймы можно загружать такие же ресурсы, что и в главное окно браузера. Фреймы могут работать независимо, а также содержать ссылки друг на друга.

Теги *<frameset>*, *<frame>* и *<noframe>*

При использовании фреймов потребуется несколько HTML-файлов. Один из них называется *установочным*. В нем описывается расположение (раскладка) фреймов в окне браузера и назначаются исходные HTML-файлы для каждого из фреймов. Исходные HTML-файлы выводятся во фреймы и могут содержать тексты, графику, ссылки и проч.

Установочный HTML-файл, как и любой другой, начинается с тега `<html>` и заканчивается тегом `</html>`. Для разделения окна браузера на несколько фреймов предназначен контейнерный тег `<frameset>`. Если в установочном

файле используются контейнерные теги `<head>` и `<body>`, то тег `<frameset>` должен быть размещен после закрывающего тега `</head>`, но перед открывающим тегом `<body>`.

Тег `<frameset>` обладает следующими атрибутами:

- `border` — толщина границ (рамок) между фреймами в пикселах, задаваемая числом;
- `bordercolor` — цвет границ между фреймами;
- `frameborder` — указывает, следует ли показывать границы между фреймами:
  - `yes` или `1` — показывать границы;
  - `no` или `0` — не показывать границы;
- `framespacing` — расстояние в пикселах между фреймами;
- `cols` — строка, содержащая сведения о количестве и ширине фреймов при их горизонтальной раскладке;
- `rows` — строка, содержащая сведения о количестве и ширине фреймов при их вертикальной раскладке.

Два фрейма можно расположить рядом друг с другом *(горизонтальная раскладка)* или друг над другом *(вертикальная раскладка)*. Для задания способа расположения в теге `<frameset>` используется атрибут `cols` (если рядом) или `rows` (если друг над другом). Чтобы разделить окно на два фрейма, в качестве значения атрибута `cols` или `rows` указывают через запятую два числа. Эти числа определяют размеры фреймов. Для трех фреймов задаются три числа, для четырех — четыре числа и т. д. Если нужно указать, что фрейм занимает все оставшееся место, используется символ "звездочка" (`*`). Размеры фреймов измеряются в пикселах или процентах. В последнем случае около числа пишется символ процента (`%`). Значения атрибутов `cols` и `rows` следует заключать в кавычки.

Примеры:

- `<frameset cols="100, *">` — задает деление окна на два вертикальных фрейма (левый и правый), первый из которых имеет ширину 100 пикселов, а второй занимает все оставшееся место окна браузера;
- `<frameset rows="20%, 30%, *">` — задает разбиение на три горизонтальных фрейма (верхний, средний и нижний) высотой 20, 30 и 50% от высоты окна браузера.

Можно использовать одновременно и горизонтальное, и вертикальное разбиение окна на фреймы. Это делается вложением тегов `<frameset>` друг в друга, т. е. размещением фреймов внутри фреймов.

Задав раскладку фреймов, следует указать для каждого из них исходный HTML-файл, который выводится в данный фрейм. Для этой цели служит тег `<frame>`, имеющий множество атрибутов:

- `src` — в качестве значения принимает URL-адрес загружаемого во фрейм ресурса;
- `name` — имя фрейма;
- `scrolling` — определяет, будут ли присутствовать в окне фрейма полосы прокрутки:
  - `auto` — полоса прокрутки появляется только при необходимости;
  - `yes` — показывать полосы прокрутки;
  - `no` — не показывать полосы прокрутки;
- `noresize` — определяет возможность изменять размеры фреймов пользователем (если этот атрибут не задан, то пользователь не может изменять размеры):
  - `noresize` — запрещено;
  - `resize` — разрешено;
- `border` — ширина полосы прокрутки в пикселах;
- `bordercolor` — цвет границы между фреймами;
- `marginheight` — вертикальный отступ в пикселах (высота пустого поля между верхней границей фрейма и его содержимым);
- `marginwidth` — горизонтальный отступ в пикселах (ширина пустых полей между боковыми границами фрейма и его содержимым).

Поскольку существуют браузеры, не поддерживающие фреймы, то следует использовать контейнерный тег `<noframe>`. В этот тег можно вставить сообщение, которое будет появляться в окне, если браузер не поддерживает фреймы. Далее мы еще рассмотрим применение тега `<noframe>`.

Пусть, например, требуется разделить окно браузера на два вертикальных фрейма так, чтобы в первом из них появлялось содержимое файла первый.htm, а во втором фрейме — содержимое файла второй.htm. Назначим фреймам имена, соответственно, "`ЛЕВЫЙ`" и "`ПРАВЫЙ`". Разумеется, имена выбираются произвольно. Ширину первого фрейма определим в 40%, а для второго выделим все оставшееся место окна. В первом фрейме запретим появление полосы прокрутки. Ширину разделительной полосы зададим равной 10 пикселам. Тогда установочный HTML-файл будет содержать следующие теги (листинг 1.4).

Листинг 1.4. Разделение окна браузера на два вертикальных фрейма

```
<html>
  <head><title>Установочный файл</title><head>
  <frameset cols="40%,*">
    <frame src="первый.htm" name="ЛЕВЫЙ" scrolling=no border=10>
    <frame src="второй.htm" name="ПРАВЫЙ">
  </frameset>
  <noframe>
    Для просмотра необходим браузер, поддерживающий фреймы
  </noframe>
</html>
```

Во фреймах можно располагать ссылки, указывающие на фреймы. Щелчок на таких ссылках может изменить файл, выводимый в другом фрейме. Этот прием часто используется при создании так называемых *навигационных панелей*. Навигационная панель представляет собой фрейм, расположенный обычно вверху или по краю окна. В ней содержатся ссылки, при инициализации которых во втором фрейме, большем по размеру, выводятся различные документы. Именно для этого и был предусмотрен атрибут `name` в теге `<frame>`, а в теге ссылки `<a>` атрибут `target`.

Ссылки на фреймы имеют формат:

```
<a href="адрес_файла" target="имя_фрейма">указатель_ссылки</a>
```

При разработке фреймовых Web-страниц обычно ограничиваются двумя-тремя фреймами. Рассмотрим создание трехфреймовой страницы с одним верхним и двумя нижними фреймами с горизонтальной раскладкой. При такой композиции в верхнем фрейме обычно размещают название сайта и, возможно, главное меню; один нижний боковой (чаще всего — левый) фрейм используют в качестве навигационной панели с набором ссылок, а другой — для отображения документа, вызванного при инициализации ссылки. Чтобы организовать указанную раскладку фреймов, сначала создают два фрейма с вертикальной раскладкой (второй ниже первого), а затем нижний фрейм разбивают на два фрейма с горизонтальной раскладкой (левый и правый). Это делается путем вложения одного тега `<frameset>` в другой, как в примере из листинга 1.5.

Листинг 1.5. Разделение окна браузера на три фрейма

```
<html>
  <head><title>Установочный файл. 3 фрейма</title><head>
  <frameset rows="120,*" >
```

```
    <frame src="верхний.htm" name="TOP" scrolling=no >
    <frameset cols="115,*" >
      <frame src="левый.htm" name="LEFT">
      <frame src="правый.htm" name="RIGHT">
    </frameset>
  </frameset>
  <noframe>
    Для просмотра необходим браузер, поддерживающий фреймы
  </noframe>
</html>
```

Во многих случаях отображение границ (рамок) между фреймами не желательно, и тогда следует в теге `<frameset>` использовать атрибут `frameborder=no` (или `frameborder=0`).

Если в теге `<frameset>` указать `framespacing=`*число*, то между фреймами появятся пустые полосы указанной ширины в пикселах.

Тег *<iframe>*

Контейнерный тег `<iframe>` применяется для создания *плавающих фреймов*, в которые можно загрузить HTML-документы. В отличие от обычных фреймов (теги `<frameset>` и `<frame>`) для создания плавающих фреймов не требуется отдельный установочный HTML-файл. Тег `<iframe>` можно использовать непосредственно в обычном HTML-документе наряду с другими элементами. Внешне плавающий фрейм выглядит как прямоугольная область (с полосами прокрутки или без них), в которой отображен заданный HTML-документ. В простейшем случае тег `<iframe>` используется следующим образом:

```
<iframe src="URL-адрес"></iframe>
```

Атрибут `src` является обязательным. Кроме него существует целый ряд необязательных атрибутов.

Атрибуты тега `<iframe>`:

- ❒ `src` — URL-адрес ресурса;
- ❒ `align` — выравнивание; возможны следующие значения:
  - `absbottom` — выравнивает нижнюю границу фрейма по подстрочной линии символов окружающего текста;
  - `absmiddle` — выравнивает середину фрейма по центральной линии между `top` и `absbottom` окружающего текста;
  - `baseline` — выравнивает нижнюю границу фрейма по базовой линии окружающего текста;

- `bottom` — совпадает с `baseline` (только в Internet Explorer);
- `left` — выравнивает фрейм по левому краю элемента-контейнера;
- `middle` — выравнивает воображаемую центральную линию окружающего текста по воображаемой центральной линии фрейма;
- `right` — выравнивает фрейм по правому краю элемента-контейнера;
- `texttop` — выравнивает верхнюю границу фрейма по надстрочной линии символов окружающего текста;
- `top` — выравнивает верхнюю границу фрейма по верхней границе окружающего текста;

❒ `name` — имя фрейма;

❒ `height` — высота фрейма в пикселах или процентах;

❒ `weight` — ширина фрейма в пикселах или процентах;

❒ `scrolling` — определяет, будут ли присутствовать в окне фрейма полосы прокрутки:

- `auto` — полоса прокрутки появляется только при необходимости;
- `yes` — показывать полосы прокрутки;
- `no` — не показывать полосы прокрутки;

❒ `hspace` — ширина боковых пустых полей вокруг фрейма в пикселах;

❒ `vspace` — ширина верхнего и нижнего пустых полей вокруг фрейма в пикселах;

❒ `marginwidth` — ширина горизонтальных отступов содержимого фрейма от его боковых границ (действует для содержимого, не являющегося HTML-документом);

❒ `marginheight` — ширина вертикальных отступов содержимого фрейма от его верхней и нижней границ (действует для содержимого, не являющегося HTML-документом).

Большинство атрибутов тега `<iframe>` такие же, что и у тега вставки графического изображения `<img>`. Так, атрибут `src` позволяет задать URL-адрес ресурса, который следует загрузить в плавающий фрейм. Это может быть адрес сайта, конкретного HTML-документа, графического файла, документа MS Office и др. Использование атрибута `align=left` или `align=right` позволяет организовать обтекание фрейма справа или слева соответственно.

В следующем примере показано, как создать плавающий фрейм с загрузкой в него Web-страницы и организацией обтекания его текстом (листинг 1.6).

Листинг 1.6. Плавающий фрейм с Web-страницей

```
<html>
  <head><title>Плавающий фрейм</title><head>
  <body bgcolor=#e0e0e0>
    Пример плавающего фрейма, в который загружен сайт поисковой системы
    <b>Яndex</b>.
    <iframe src="http://www.yandex.ru" width=350 height=200
            hspace=20 vspace=10
            align=left></iframe>
    Обратите внимание, что текст обтекает фрейм справа
  </body>
</html>
```

В плавающие фреймы можно загружать не только HTML-документы (Web-страницы), но и другие ресурсы. Атрибуты `marginwidt` и `marginheight` действуют не для любого возможного содержимого фрейма. Например, они никак не влияют на отображение HTML-документов и некоторых других, но позволяют задать отступы для загружаемых графических изображений.

Атрибут `name` используется с той же целью, что и одноименный атрибут обыкновенных фреймов в теге `<frame>`. А именно в теге ссылки `<a>` в качестве значения атрибута `target` можно задать имя (значение атрибута `name`) плавающего фрейма, в котором следует отобразить документ.

Примечание

Хотя тег `<iframe>` является контейнерным, автору не известно, для чего это необходимо. Элементы, помещенные между `<iframe>` и `</iframe>`, не отображаются в Internet Explorer. С другой стороны, если не использовать закрывающий тег `</iframe>`, то все последующие элементы документа отображаться не будут.

Примечание

В настоящее время тег `<iframe>` поддерживается только MS Internet Explorer 3+ и не поддерживается Netscape Navigator. Браузер Opera поддерживает его, если он настроен на поведение, аналогичное Internet Explorer соответствующей версии.

Примечание

Применение плавающих фреймов особенно привлекательно при разработке Web-приложений с применением позиционирования посредством каскадных таблиц стилей.

1.2.6. Форматирование текста

Текст в HTML-документе может быть введен непосредственно, т. е. без использования каких бы то ни было тегов. При этом невидимые служебные символы перехода на другую строку, табуляции и длинные последовательности пробелов будут проигнорированы браузером при отображении текста. Например, вы можете ввести несколько строк текста, а браузер отобразит этот текст как одну строку; несколько пробелов, введенных вами для обозначения отступа, браузер заменит одним пробелом. Иначе говоря, форматирование текста средствами текстового редактора, в котором набирается HTML-код, не соответствует форматированию этого текста браузером. Чтобы управлять форматированием текста в окне браузера, следует использовать специальные теги. Так, для перевода строки предназначен тег `<br>`, для обозначения абзаца — `<p>`, для расположения текста в центре окна браузера — `<center>`, для запрета переноса слов — `<nobr>`, для установки режима "мягкого" переноса слов — `<wbr>`. Эти теги уже рассматривались в *разд. 1.2.2*. Напомним, что теги `<p>`, `<br>` и `<center>` влияют не только на текст, но и на отображение других элементов HTML-документа.

Широкие возможности для форматирования текстов дают таблицы стилей, которым посвящен *разд. 1.3*. А здесь мы рассмотрим только основные теги HTML, с помощью которых можно влиять на представление текстовой информации. В HTML имеется много тегов форматирования текста. Однако в большинстве случаев вполне можно обойтись лишь некоторыми из них. В первую очередь отметим теги `<h1>`, ..., `<h6>`, `<basefont>`, `<font>`, `<b>` и `<i>`.

Заголовки

Для задания размера шрифта можно применять теги заголовков или, как их еще называют, логических стилей. Таких тегов всего шесть, и они обычно используются для определения заголовков различного уровня. При переходе от заголовка первого уровня к заголовку шестого уровня постепенно уменьшается размер и толщина букв шрифта. Теги заголовков записываются как `<h1>`, `<h2>`, ..., `<h6>`. Каждый из них имеет соответствующий закрывающий тег. Таким образом, теги заголовков являются контейнерными. Заметим, что тег `<hn>` ($n = 1, 2, ..., 6$) определяет стиль текста сообразно настройкам браузера. При этом стиль `<h2>` всегда будет "меньше", чем `<h1>`, если, конечно, автор документа не переопределил его по своему усмотрению. Дело в том, что вы имеете возможность изменить установки по умолчанию с помощью таблиц стилей (тег `<style>` и атрибут `style`).

Текст, заключенный в тег заголовка, всегда начинается с новой строки. Текст, следующий за заголовком, также начинается с новой строки.

Теги заголовков имеют атрибуты:

- `align` — выравнивание; значения: `center`, `left`, `right` (по центру, левому и правому краю окна браузера);
- `title` — текст всплывающей подсказки, появляющейся при наведении указателя мыши на текст заголовка.

Физические стили

Кроме использования стандартных размеров и начертаний шрифтов, можно определять шрифты для каждого фрагмента текста с помощью специальных тегов. Самый простой способ — использование тегов так называемых физических стилей (табл. 1.3).

Таблица 1.3. *Теги физических стилей*

Стиль	Тег
Полужирный (Bold)	`<b>`
Курсив (Italic)	`<i>`
Подчеркнутый (Underscore)	`<u>`
Вычеркнутый (Strike Out)	`<s>`
Пишущая машинка (Typewriter)	`<tt>`

Для каждого тега физического стиля существует соответствующий закрывающий тег. Таким образом, теги физических стилей являются контейнерными.

К одному и тому же фрагменту текста можно применить несколько тегов физических стилей, вкладывая один из них в другой.

Внутри тега заголовка можно вставить тег физического стиля, чтобы модифицировать весь заголовок или только некоторую его часть. Например, чтобы выделить курсивом часть текста, определенного в качестве заголовка второго уровня, можно использовать конструкцию, приведенную в следующем примере:

```
<html>
  <head><title>Физические и логические стили</title></head>
  <h2>Это — <i>модифицированный</i> заголовок 2-го уровня</h2>
</html>
```

Тег *<basefont>*

С помощью тега `<basefont>` можно установить основной шрифт, который используется в документе по умолчанию. Этот тег имеет необязательные атрибуты:

- `color=`*цвет*;
- `face="`*имя_шрифта*`"`;
- `size=`*число*, где *число* = 1, 2, ..., 7 и определяет размер шрифта;
- `title="`*текст всплывающей подсказки*`"`.

Цвет задается именем или шестнадцатеричным числом. Размер определяется целым числом от 1 до 7; чем больше это число, тем больше размер шрифта. Если атрибут `size` не задан, то по умолчанию устанавливается значение 3. Обратите внимание, что значение атрибута `size` безразмерно: это относительный размер шрифта.

Тег *<font>*

С помощью контейнерного тега `<font>` можно настроить шрифт для содержащегося в нем текста. Тег `<font>` имеет такие же атрибуты, что и тег `<basefont>`. Особенностью атрибута `size` для тега `<font>` является то, что с его помощью можно установить не только абсолютный, но и относительный размер шрифта. В последнем случае значение атрибута `size` представляет собой целое число со знаком `+` или `-` для указания, на сколько следует увеличить или уменьшить размер шрифта относительно основного. Если значение атрибута `size` указано без знака, то предполагается абсолютный размер шрифта. Например, если размер основного шрифта задан равным 3, то тег `<font size=+2>` устанавливает размер текущего шрифта равным 5.

Для задания гарнитуры шрифта используется атрибут `face="`*имя_шрифта*`"`. Например, `<font face="Arial">`. Если этот тег не используется в HTML-документе, то браузер будет применять шрифт, установленный в настройках. Поэтому текст на экране пользователя может выглядеть совсем не так, как вы себе его представляли. Следует также иметь в виду, что если назначенный вами шрифт отсутствует на компьютере пользователя, то браузер будет изображать текст шрифтом, установленным по умолчанию. В значении атрибута `face` можно указать через запятую перечень шрифтов. В этом случае браузер будет использовать первый найденный шрифт. Например, можно записать тег:

```
<font face="Arial, Sans Serif, Courier">
```

Обычно в перечне задают похожие шрифты. Рекомендуется назначать наиболее популярные шрифты. При размещении на Web-странице текстовой ин-

формации лучше вообще не указывать название шрифта, полагаясь на стандартные настройки браузера. Но тогда при разработке страницы следует также использовать стандартные настройки браузера, чтобы синхронизировать свое восприятие текста с возможным восприятием пользователя. В конце концов, вы создаете страницу не для себя, а для читателей.

HTML-код, приведенный в листинге 1.7, демонстрирует управление шрифтом.

Листинг 1.7. Пример управления шрифтом

```
<html>
  <head><title>Управление шрифтом</title></head>
  <p>
  Шрифт Arial<br>
  <font face="ARIAL">АБВГДЕЖЗ 012345</font>
  <p>
  Шрифт Courier<br>
  <font face="COURIER">АБВГДЕЖЗ 012345</font>
  <p>
  Шрифт Courier, увеличенный на 3<br>
  <font face="COURIER" size=+3>АБВГДЕЖЗ 012345</font>
  <p>
  Шрифт SYMBOL размером 7<br>
  <font face="SYMBOL" size=7 color=red>АБВГДЕЖЗ 012345</font>
</html>
```

Индексы

В математических формулах, а также для ссылок на подстрочные замечания часто применяются индексы, которые отличаются от основного текста положением относительно строки (чуть выше или ниже), а также размером. Для этой цели служат контейнерные теги `<sup>` и `<sub>` для верхних и нижних индексов соответственно. Следующий код иллюстрирует применение верхних и нижних индексов (листинг 1.8).

Листинг 1.8. Использование верхних и нижних индексов

```
<html>
  <head><title>Индексы</title></head>
  <p>
  x<sup>y</sup>
```

```
  <p>
  x<sub>y</sub>
  <p>
  x<sub>1</sub>+ x<sub>2</sub>+ x<sub>3</sub>
  <p>
  (5+sin(x)<sup>2</sup>)<sup>x+z<sup>y<sub>2</sub></sup></sup>
  <p>
  Указание на подстрочное примечание <sup>*</sup>
</html>
```

Относительное изменение размера шрифта

Размер шрифта для текущего текста может быть изменен относительно размера основного шрифта. Напомню, что размер основного шрифта по умолчанию равен 3. Установка размера основного шрифта производится с помощью атрибута `size` тега `<basefont>`; этот атрибут принимает целочисленные значения от 1 до 7. Для относительного изменения размера шрифта фрагмента текста можно использовать тег `<font>` с атрибутом `size`, значение которого указано в виде числа со знаком `+` или `-` для увеличения или уменьшения размера шрифта соответственно. Об этом уже говорилось в *разд. "Тег <font>" ранее в этой главе*. Кроме того, можно использовать контейнерные теги `<big>` и `<small>`.

Контейнерный тег `<big>` увеличивает на единицу размер шрифта, а контейнерный тег `<small>` задает более мелкий шрифт относительно основного шрифта. В листинге 1.9 приведен пример использования тегов `<big>` и `<small>`.

Листинг 1.9. Использование тегов `<big>` и `<small>`

```
<html>
  <head><title>Относительное изменение размеров шрифта</title></head>
  Основной шрифт
  <p>
  <big>Увеличенный на единицу размер шрифта</big>
  <p>
  <big><big>Увеличенный на две единицы размер шрифта</big></big>
  <p>
  <small>Шрифт, более мелкий, чем основной
</html>
```

Для многократного увеличения относительного размера шрифта тег `<big>` следует применить несколько раз.

Текст с отступом

Для выделения фрагмента текста пустыми строками сверху и снизу, а также отступом от левого края окна браузера служит контейнерный тег `<blockquote>`.

Для увеличения отступа тег `<blockquote>` следует применить несколько раз. Например, для двойного отступа нужно использовать такую запись:

```
<blockquote><blockquote>Текст с двойным отсту-
пом</blockquote></blockquote>
```

Бегущая строка

Браузер Internet Explorer поддерживает контейнерный тег `<marquee>`, который позволяет создать так называемую *бегущую строку*, т. е. эффект прокручивания текста в заданном поле. Характеристики бегущей строки задаются следующими атрибутами:

- `width` — ширина поля бегущей строки в пикселах или процентах от ширины окна;
- `height` — высота поля бегущей строки (в пикселах);
- `hspace`, `vspace` — интервалы по горизонтали и вертикали между текстом строки и краями ее поля (в пикселах);
- `align` — положение текста бегущей строки в ее поле; возможные значения:
  - `top` (вверху);
  - `bottom` (внизу);
  - `middle` (посередине);
- `direction` — определяет направление движения; возможные значения:
  - `left` (справа налево);
  - `right` (слева направо);
- `behavior` — характер движения строки; возможные значения:
  - `scroll` — текст появляется от одного края и скрывается за другим;
  - `slide` — строка вытягивается из одного края поля и останавливается у другого края;
  - `alternate` — задает переменное направление движения от одного края к другому, а затем обратно;
- `loop` — количество повторений текста в бегущей строке, задаваемое числом; если необходимо "бесконечное" количество повторений, то следует задать аргумент в виде ключевого слова `infinity`. Атрибут `loop` не влияет

на поведение бегущей строки, если для атрибута `behavior` заданы значения `alternate` или `slide`;

- `scrollamount` — устанавливает длину в пикселах "прыжка" текста за один такт; при большом значении этого параметра текст движется рывками, а при малом — замедленно;
- `scrolldelay` — определяет величину паузы между тактами перемещения текста в миллисекундах;
- `bgcolor` — устанавливает цвет поля бегущей строки, задаваемый шестнадцатеричным числом или именем.

Соотношения между длиной текста, размером шрифта и скоростью перемещения, при которых бегущая строка выглядит приемлемо, подбираются опытным путем. В следующем примере бегущая строка "бесконечно" прокручивается на желтом поле шириной 75% от ширины окна браузера (листинг 1.10).

Листинг 1.10. Бегущая строка

```
<html>
  <head><title>Бегущая строка</title></head>
  <marquee height=50 width=75%
           hspace=5
           vspace=5
           align=top
           bgcolor=yellow
           direction=left
           loop=infinity
           behavior=scroll
           scrollamount=5
           scrolldelay=100>
    <h1>Это сообщение будет прокручиваться на экране справа налево</h1>
  </marquee>
</html>
```

В тег `<marquee>` в качестве прокручиваемого объекта можно вставлять не только текстовые элементы, но и, например, графические изображения.

В случае вставки изображений в тег `<marquee>` высота области определяется не атрибутом `height` этого тега, а высотой самого крупного изображения. Размеры изображения можно задать с помощью атрибутов `height` и `width` тега `<img>`.

Предварительно отформатированный текст

Иногда в HTML-документ необходимо включить фрагмент текста, предварительно отформатированный обычными средствами текстового редактора, т. е. с использованием пробелов, символов перевода строки, табуляции и т. п. Для этого предусмотрен контейнерный тег `<pre>`.

Теги форматирования текстов, содержащие тег `<pre>` и находящиеся внутри этого тега, влияют на отображение предварительно отформатированного текста.

Списки

Для представления информации в виде простых и иерархических (вложенных) списков в HTML имеются несколько тегов. Примером такого списка может быть оглавление книги или толковый словарь. Наиболее часто используются маркированный и нумерованный списки, задаваемые контейнерными тегами `<ul>` и `<ol>` соответственно. Элементам этих списков предшествуют теги `<li>`. Для создания списков типа словаря применяется тег `<dl>`, элементы которого задаются с помощью тегов `<dt>` и `<dl>`. Теги списков кроме атрибутов, перечисленных далее, имеют атрибуты `style` и `class` для форматирования содержимого с помощью таблиц стилей, атрибут `title`, задающий текст всплывающей подсказки, и атрибут `id`, чтобы иметь доступ к спискам и их элементам из сценария.

Маркированный список

Маркированный список еще называют *неупорядоченным*. Он задается контейнерным тегом `<ul>` (от англ. *unordered list*). Элементы списка (например, текстовые строки) располагаются внутри этого тега; каждому элементу списка предшествует тег `<li>` (от англ. *list item*). Браузер отображает элементы списка с новой строки в виде столбца с отступом и маркерами. Вид маркера определяется атрибутом `type`, который может принимать следующие значения:

- `disc` — закрашенный кружок (значение по умолчанию);
- `circle` — незакрашенный кружок;
- `square` — закрашенный квадратик.

Примечание

Вид маркеров, указанный выше, отображается браузером Internet Explorer. Однако не исключено, что другие браузеры могут сделать это иначе.

Атрибут `type` с этими же значениями можно использовать и в теге `<li>` внутри тега `<ul>`, чтобы определить вид маркера для данного элемента списка. Кроме того, тег `<li>` имеет необязательный атрибут `value`, принимающий

в качестве значения произвольную строку символов. Это значение может быть обработано в сценарии, который выполняется по некоторому событию (например, при щелчке на элементе списка).

В листинге 1.11 приведен пример маркированного списка.

Листинг 1.11. Маркированный список

```
<html>
  <head><title>Маркированный список</title></head>
  <b>Список планет </b>
  <ul type=square>
    <li>Меркурий
    <li>Венера
    <li>Земля
   </ul>
</html>
```

Вы можете в качестве маркеров использовать произвольные графические изображения (в том числе и анимационные gif-файлы), вставляемые посредством тега `<img>`, как в следующем примере (листинг 1.12).

Листинг 1.12. Маркированный список с маркерами-изображениями

```
<html>
  <head><title>Маркированный список</title></head>
  <b>Список планет</b>
  <ul>
    <img src=planeta.gif>Меркурий<br>
    <img src=planeta.gif>Венера<br>
    <img src=planeta.gif>Земля<br>
  </ul>
</html>
```

Иерархический маркированный список создается путем вложения контейнеров `<ul>` друг в друга.

При создании иерархического маркированного списка без указания типа маркера с помощью атрибута `type` браузер определяет их автоматически следующим образом: `disc` — для 1-го уровня, `circle` — для 2-го уровня и `square` — 3-го и всех последующих уровней.

Нумерованный список

Нумерованный список, называемый еще *упорядоченным*, задается с помощью контейнерного тега `<ol>` (от англ. *ordered list*). Построение этого списка аналогично организации маркированного списка, рассмотренного ранее. Особенность нумерованного списка заключается в способе пометки элементов, определяемом атрибутом `type`, который может принимать следующие значения:

- `1` — нумерация арабскими цифрами (значение по умолчанию);
- `A` — маркировка прописными латинскими буквами в алфавитном порядке;
- `a` — маркировка строчными латинскими буквами в алфавитном порядке;
- `I` — нумерация большими римскими цифрами;
- `i` — нумерация маленькими римскими цифрами.

Атрибут `type` можно использовать как в теге `<ol>`, так и в теге `<li>`.

С помощью атрибута `start` тега `<ol>` допустимо указать маркер первого элемента в списке. Значением этого атрибута может быть число. Так, например, `start=7` при `type=1` нумерация элементов начинается с 7; при `type=A` первый элемент получает маркер в виде буквы G; при `type=I` первый элемент получает маркер в виде VII. В листинге 1.13 приведен пример списка при трех различных способах маркировки.

Листинг 1.13. Нумерованный список с разными способами маркировки

```
<html>
  <head><title>Нумерованный список</title></head>
  <ol>
    <li>Меркурий
    <li>Венера
    <li>Земля
  </ol>
  <ol type=A>
    <li>Меркурий
    <li>Венера
    <li>Земля
  </ol>
  <ol type=I>
    <li>Меркурий
```

```
    <li>Венера
    <li>Земля
  </ol>
</html>
```

Иерархические нумерованные списки создаются аналогично маркированным иерархическим спискам. Однако их особенность заключается в том, что отличия в маркировке элементов различных уровней следует устанавливать самостоятельно с помощью атрибута `type`.

Список определений

Список определений содержит элементы, которые состоят из двух частей — определяемой и определяющей. Примером такого списка является толковый словарь. Каждый элемент словаря состоит из определяемого термина (понятия) и его описания.

Список определений задается контейнерным тегом `<dl>` (от англ. *definition list*). Элемент списка задается двумя контейнерными тегами: `<dt>` — для определяемого термина (от англ. *definition term*) и `<dd>` — для его описания (от англ. *definition description*). Браузер отображает определяемый термин (строку символов) в отдельной строке, а его описание — ниже и с отступом, как показано в следующем примере (листинг 1.14).

Листинг 1.14. Список определений

```
<html>
  <head><title>Список определений</title></head>
  <dl>
    <dt>HTML</dt>
    <dd>Язык разметки гипертекста (HyperText Markup Language).
      Документы, написанные на HTML, составляют основу Web-страниц.</dd>
    <dt>JavaScript<dt>
    <dd>Язык программирования сценариев.</dd>
  </dl>
</html>
```

При желании содержимое тегов `<dt>` и `<dd>` можно отформатировать, используя, например, теги заголовков или физических стилей. Кроме текста в определяемую и/или описательную части списка можно вставить графические изображения.

Иерархические списки определений создаются таким же способом, как и иерархические маркированные и нумерованные списки.

Специальные символы

При формировании HTML-документа может потребоваться ввести символы, которые воспринимаются браузером как служебные. Например, нельзя использовать символы < и > для обозначения знаков "меньше" и "больше", т. к. они интерпретируются как символы тегов. Кавычки (") и амперсанд (&) также применяются в языке HTML для служебных целей. Все символы, которые можно набрать при нажатых клавишах управления, отличных от <Shift>, могут не воспроизводиться при просмотре HTML-документа, поскольку они зарезервированы. Кроме того, несколько пробелов, введенных подряд, браузер воспримет как один символ (проблема создания отступов с помощью символа пробела). В этих случаях можно использовать буквенные или числовые эквиваленты символов. Числовые эквиваленты символов однозначно соответствуют их кодам ASCII.

Буквенный эквивалент символа имеет формат:

```
&буквенный_эквивалент;
```

Например, `<` обозначает символ < (от англ. *less then* — меньше чем).

Числовой эквивалент имеет формат:

```
&#код_ASCII;
```

Например, символ < можно вставить в документ как `<`. Символ пробела имеет буквенный эквивалент ` ` и числовой эквивалент ` `.

Таким образом, эквиваленты заключаются между символами `&` и `;`. При этом в случае цифрового эквивалента перед ASCII-кодом символа следует поставить символ `#`.

HTML-код, представленный в листинге 1.15, выводит таблицу эквивалентов для некоторых часто употребляемых символов.

Листинг 1.15. Таблица эквивалентов для некоторых символов

```
<html>
  <head><title>Специальные символы</title></head>
  <font face="courier" size=5>
  Симв. Код Буквенный эквивалент<br>
  <hr>
  &#34;      34 quote <br>
```

```
  &#38;      38 amp <br>
  &#60;      60 lt <br>
  &#62;      62 gt <br>
        160 nbsp <br>
  &#162;     162 cent <br>
  &#163;     163 pound <br>
  &#167;     167 sect <br>
  &#169;     169 copy <br>
  &#174;     174 reg <br>
  &#177;     177 plusmn <br>
  &#181;     181 micro <br>
  &#182;     182 para <br>
  &#188;     188 frac14 <br>
  &#189;     189 frac12 <br>
  &#190;     190 frac34 <br>
  &#198;     198 AElig <br>
  &#216;     216 Oslash <br>
  &#230;     230 aelig <br>
  &#247;     247 divide <br>
</html>
```

Другие теги разметки текстов

Некоторые теги ничего особенного не делают, кроме того, что выводят содержащийся в них текст моноширинным шрифтом и/или курсивом. До появления каскадных таблиц стилей (CSS) использование этих тегов способствовало поддержанию стилевого единства документов. Теперь они довольно редко применяются. Ниже приведен их список:

- `<address>` — служит для представления сведений об авторе, дате создания и обновления документа; текст внутри этого контейнера обычно показывается курсивом;
- `<kbd>` — обычно используется для отображения моноширинным шрифтом текста, вводимого пользователем;
- `<code>` — обычно применяется для отображения моноширинным шрифтом текста, содержащего программный код. Если внутри этого тега находится контейнер `<script>`, содержащий сценарий, то сценарий будет выполнен, но не отображен;
- `<samp>` — обычно используется для отображения моноширинным шрифтом текста, содержащего пример программного кода. Если внутри этого

тега находится контейнер `<script>`, содержащий сценарий, то этот сценарий будет выполнен, но не отображен;

- `<xmp>` — используется так же, как и теги `<code>` и `<samp>`;
- `<var>` — служит для отображения названий переменных мелким моноширинным шрифтом;
- `<cite>` — обычно используется для отображения текста (например, цитат) курсивом;
- `<em>` — отображает текст курсивом.

Вывод HTML-кода как текста

Иногда требуется показать на Web-странице некоторый HTML-код. Иначе говоря, необходимо, чтобы HTML-теги не интерпретировались (не выполнялись) браузером, а выводились просто как текст. Это можно сделать несколькими способами. Один из них основан на применении контейнерного тега `<plaintext>`. Предполагалось, что не интерпретируется, а лишь выводится на экран только тот фрагмент HTML-документа, который заключен между тегами `<plaintext>` и `</plaintext>`. Однако похоже, что браузер MS Internet Explorer не считает этот тег контейнерным, поскольку выводит как текст все, что следует за открывающим тегом `<plaintext>`.

Второй способ заключается в том, чтобы после открывающей угловой скобки вставить пробел. В этом случае браузер не будет интерпретировать такой тег, а сочтет его обыкновенным текстовым фрагментом документа. Заметим, что любое количество пробелов перед закрывающей угловой скобкой оставляет запись тега интерпретируемой.

Третий способ заключается в том, чтобы отображаемый как текст фрагмент HTML-кода заключить в контейнерный тег `<textarea>`, создающий прокручиваемую текстовую область. Подробно этот тег рассматривается в *разд. 1.2.7*.

1.2.7. Формы и элементы пользовательского интерфейса

Интерфейс является некоторой системой, организующей взаимодействие других разнородных систем. Пользовательский интерфейс, о чем рассказывается в данном разделе, обеспечивает взаимодействие пользователя с некоторой программной системой. Обычно пользовательский интерфейс представляется набором элементов, посредством которых пользователь программы может ввести какие-то данные, послать сообщение, создать событие и т. п., которые должны быть как-то обработаны соответствующей программой.

В пользовательском интерфейсе есть две стороны: элементы, воспринимающие действия пользователя, и программы, обрабатывающие эти действия. К элементам интерфейса относятся поля ввода данных, кнопки, раскрывающиеся списки, текстовые области и т. п. Пользователь может воздействовать на их состояние (содержимое) посредством клавиатуры и/или мыши. В большинстве случаев эти воздействия обрабатываются сценариями, написанными на скриптовых языках, например, JavaScript или VBScript. Однако простейшее действие, заключающееся лишь в том, чтобы передать на сервер введенные пользователем данные, может быть выполнено и средствами только HTML без каких бы то ни было сценариев.

Формы

Форма, содержащая элементы пользовательского интерфейса и предназначенная для отправки данных на сервер, задается посредством контейнерного тега `<form>`. Внутри этого тега могут располагаться теги, задающие элементы пользовательского интерфейса (`<input>`, `<select>` и `<textarea>`). Эти и другие элементы будут рассмотрены в следующем подразделе.

Благодаря специальным атрибутам тега `<form>` с помощью кнопки типа `submit` (`<input type=submit>`) можно обеспечить передачу на сервер данных, содержащихся в элементах формы.

Тег `<form>` имеет перечисленные ниже атрибуты.

- `action` — URL-адрес сервера или конкретного серверного приложения, обрабатывающего переданные ему данные формы. Для передачи данных этот атрибут обязателен: если он не указан или его значение пусто, данные формы не будут отправлены никаким способом.

  Примеры:

  - `<form action="/cgi-bin/">` — относительный URL-адрес; поскольку файл программы не задан, то предполагается первая программа в указанной папке;
  - `<form action="http://mysite.com/scripts/myprogram.pl">` — URL-адрес файла программы, написанной на языке Perl;
  - `<form action="www.anyserver.ru/mypage.asp">` — URL-адрес на asp-файл (Active Server Page);
  - `<form action="www.anyserver.ru/mypage.php">` — URL-адрес на php-файл (файл, содержащий сценарий на языке PHP);
  - `<form action="mailto:anyname@anymailserver.ru">` — для отправки данных по электронной почте по указанному адресу.

- `method` — метод передачи данных на сервер; возможны два значения: `get` и `post`.
- `enctype` — метод кодирования данных, передаваемых на сервер.
- `name` — имя формы.
- `target` — строка, содержащая имя окна или фрейма, в котором следует показать форму; возможны также и следующие значения: `_parent`, `_blank`, `_top`, `_self`.

При отправке данных формы браузер генерирует запрос на Web-сервер в соответствии с протоколом HTTP — основным протоколом передачи в WWW. HTTP-запрос состоит из трех частей:

- метод; URL-адрес запроса и, возможно, данные; версия протокола;
- заголовок запроса (служебная часть);
- тело запроса, содержащее передаваемые данные.

Метод запроса задается в теге `<form>` как значение атрибута `method`. Значение `get`, устанавливаемое по умолчанию, означает, что передаваемые данные будут присоединены к URL-адресу через разделитель в виде символа `?`. Метод `get` используют при передаче небольших объемов данных. Значение `post` означает, что данные будут находиться в теле запроса. Этот метод применяется при передаче больших объемов данных и, в частности, сообщений электронной почты.

Передаваемые данные в запросе представляются последовательностью пар вида:

*значение_атрибута*_name=*значение_атрибута*_value

Для каждого элемента формы генерируется своя пара *имя=значение*. При использовании метода `get` такие пары разделяются амперсандом (`&`). Например:

```
GET /cgi-bin/myprogram.pl?product=auto&model=vaz2110&color=red HTTP/1.1
```

Атрибут `enctype` задает метод кодирования данных. Точнее, значением этого атрибута является так называемый MIME-тип (Multipurpose Internet Mail Extensions, многоцелевые расширения электронной почты для Интернета). Возможны следующие значения:

- `application/x-www-form-urlencoded` (значение по умолчанию);
- `multipart/form-data`;
- `text/plain`.

Для передачи данных формы по электронной почте рекомендуется для атрибута `enctype` использовать значение `"text/plain"`.

Рассмотрим в качестве примера HTML-документ, который содержит форму для передачи сообщений, введенных в поля формы. Данные, введенные пользователем в поля формы, передаются серверному сценарию, расположенному в файле mypage.php (по расширению файла предполагается, что сценарий написан на языке PHP).

Эта форма (рис. 1.2) содержит:

- поле ввода адреса отправителя:

```
<input type="text" name="from" value="" size=40>
```

- текстовую область для ввода собственно сообщения:

```
<textarea name='text' cols=40 rows=10>Привет от Вашего
читателя</textarea>
```

- кнопку сброса введенных данных:

```
<input type="reset">
```

- кнопку инициализации передачи данных формы:

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

В листинге 1.16 приведен пример HTML-кода документа с описанной выше формой.

Листинг 1.16. Пример формы

```
<html>
  <head><title>Форма</title></head>
  <body bgcolor=#e0e0e0>
    <h2>Отправьте сообщение автору</h2>
    <form action="http://www.myserver.ru/mypage.php" method="post"
          enctype="text/plain">
      Ваш E-mail:
      <input type="text" name="from" value="" size=40>
      <p>
      Сообщение:<br>
      <textarea name='text' cols=40 rows=10>
       Привет от Вашего читателя
       </textarea>
      <p>
      <input type="reset">
      <input type="submit" name="Отправить" value="Отправить">
    </form>
  </body>
<html>
```

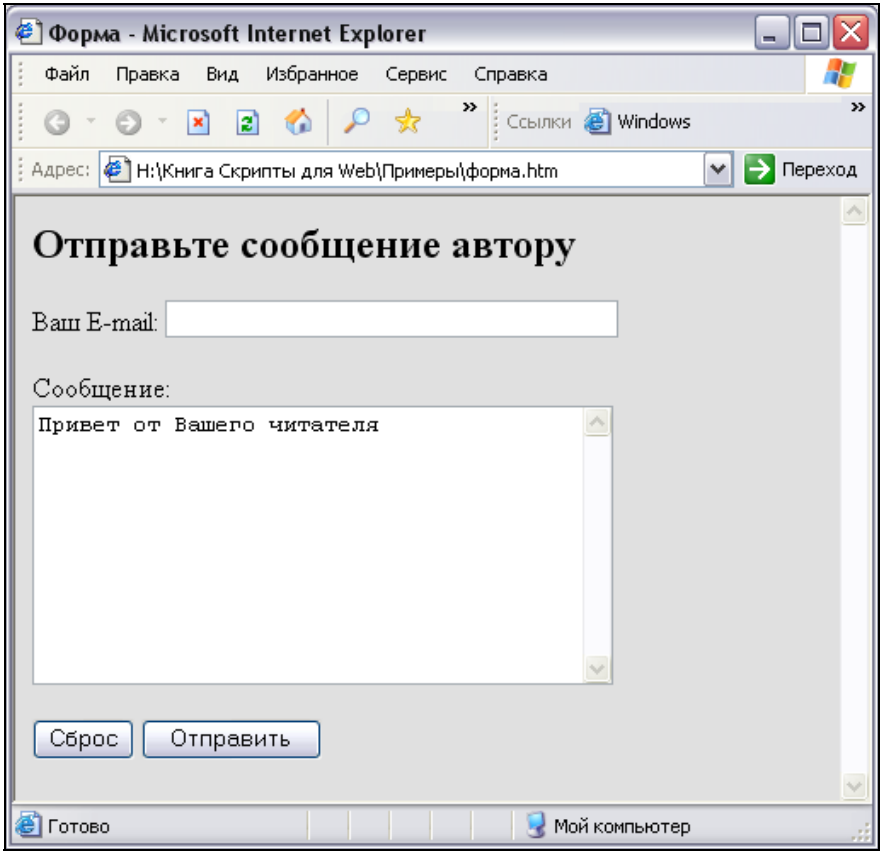


Рис. 1.2. Форма для отправки сообщений на сервер

Чтобы данные из формы можно было обработать серверным сценарием, необходимо в каждом теге элемента формы, содержащем передаваемые данные, использовать атрибут `name` (имя) с уникальным значением. В этом случае на сервер для каждого элемента формы будут переданы пары вида:

значение_атрибута`_name=`*данные_в_элементе*

Серверный сценарий, получив эти данные от клиента, сможет разобрать их как пары вида *имя_переменной=значение*, а затем обработать их каким-либо образом.

Данные, находящиеся в элементах формы, можно отправить не только серверному сценарию, но и просто по электронной почте. В этом случае необходимо атрибуту `action` тега `<form>` присвоить соответствующее значение:

`action==mailto:`*адрес*`?subject=`*тема*

Вот пример записи тега <form> для отправки данных элементов формы по электронной почте:

```
<form action="mailto:xxx@yyyy.ru?subject=Вопрос" method="post"
      enctype="text/plain">
```

Внимание

Форма, содержащая элементы с данными для отправки на сервер, должна иметь кнопку типа submit (<input type=submit>), а в теге <form> должен быть атрибут action, содержащий URL-адрес получателя. Если атрибут action не указан или его значение пусто, то отправка данных не произойдет даже при щелчке на кнопке типа submit.

Элементы пользовательского интерфейса

Элементы пользовательского интерфейса могут использоваться в HTML-документах как элементы формы (контейнера <form>), так и сами по себе. Их следует включать в форму, если только предполагается передавать содержащиеся в них данные на сервер для последующей обработки.

Тег *<input>*

С помощью тега <input> можно создавать элементы различных типов: поля ввода данных, кнопки и переключатели (флажки и радиокнопки). Тип элемента определяется атрибутом type. В зависимости от типа элемента могут использоваться те или иные дополнительные атрибуты. Некоторые типы элементов предусмотрены специально для передачи на сервер данных, ассоциированных с элементами других типов. Например, кнопка специального типа submit инициализирует передачу данных, если эта кнопка и элементы, содержащие данные, находятся в одном контейнерном теге формы <form> с соответствующими значениями атрибутов.

Атрибут type тега <input> может принимать следующие значения:

- text — текстовое поле ввода данных; дополнительные атрибуты:
  - maxlength — максимальное количество вводимых символов (по умолчанию — не ограничено);
  - size — ширина поля, выраженная в количестве одновременно видимых символов;
  - value — значение (содержимое поля);
- checkbox — переключатель типа "флажок"; дополнительные атрибуты:
  - value — значение, ассоциированное с флажком;
  - checked — если указан, то флажок является установленным;

- `radio` — переключатель типа "радиокнопка" (селекторный переключатель); дополнительные атрибуты:
  - `value` — значение, ассоциированное с радиокнопкой;
  - `checked` — если указан, то радиокнопка является установленной (выбранной);
- `file` — текстовое поле ввода данных с кнопкой **Обзор**, щелчок на которой открывает диалоговое окно **Выбор файла**; полное имя выбранного файла отображается в поле ввода. Дополнительные атрибуты `maxlength`, `size` и `value` такие же, что и для обычного поля ввода данных, задаваемого атрибутом `type="text"`;
- `password` — текстовое поле ввода данных, в котором все вводимые символы отображаются как символ "звездочка" (`*`). Дополнительные атрибуты `maxlength`, `size` и `value` те же, что и для обычного поля ввода данных, задаваемого атрибутом `type="text"`. Чаще всего такое поле используется для ввода пароля. Однако, хотя вводимые в поле типа `password` символы и маскируются символами `*`, передача их на сервер не шифруется;
- `button` — кнопка; дополнительный атрибут `value` принимает в качестве значения строку, отображаемую на кнопке. По умолчанию на кнопке нет надписи;
- `submit` — кнопка, щелчок на которой приводит к отправке данных, содержащихся в форме, на сервер; дополнительный атрибут `value` принимает в качестве значения строку, отображаемую на кнопке. По умолчанию на кнопке находится надпись **Передача запроса** или **Submit** в зависимости от браузера. Передача данных произойдет только лишь при определенных условиях *(см. разд. "Формы" ранее в этой главе)*;
- `image` — графическое изображение, играющее такую же роль, что и кнопка типа `submit`; адрес графического файла указывается как значение атрибута `src`;
- `reset` — кнопка **Сброс**, щелчок на которой отменяет все изменения в элементах, содержащихся в форме (восстанавливает исходные значения); дополнительный атрибут `value` принимает в качестве значения строку, отображаемую на кнопке. По умолчанию на кнопке находится надпись **Сброс** или **Reset** в зависимости от браузера;
- `hidden` — невидимый элемент; данные, сохраняемые как значение атрибута `value`, не могут быть изменены пользователем. Если этот элемент находится в форме, то его данные могут быть переданы на сервер.

На рис. 1.3 показано, как выглядят в окне браузера элементы пользовательского интерфейса, заданные тегом `<input>`.

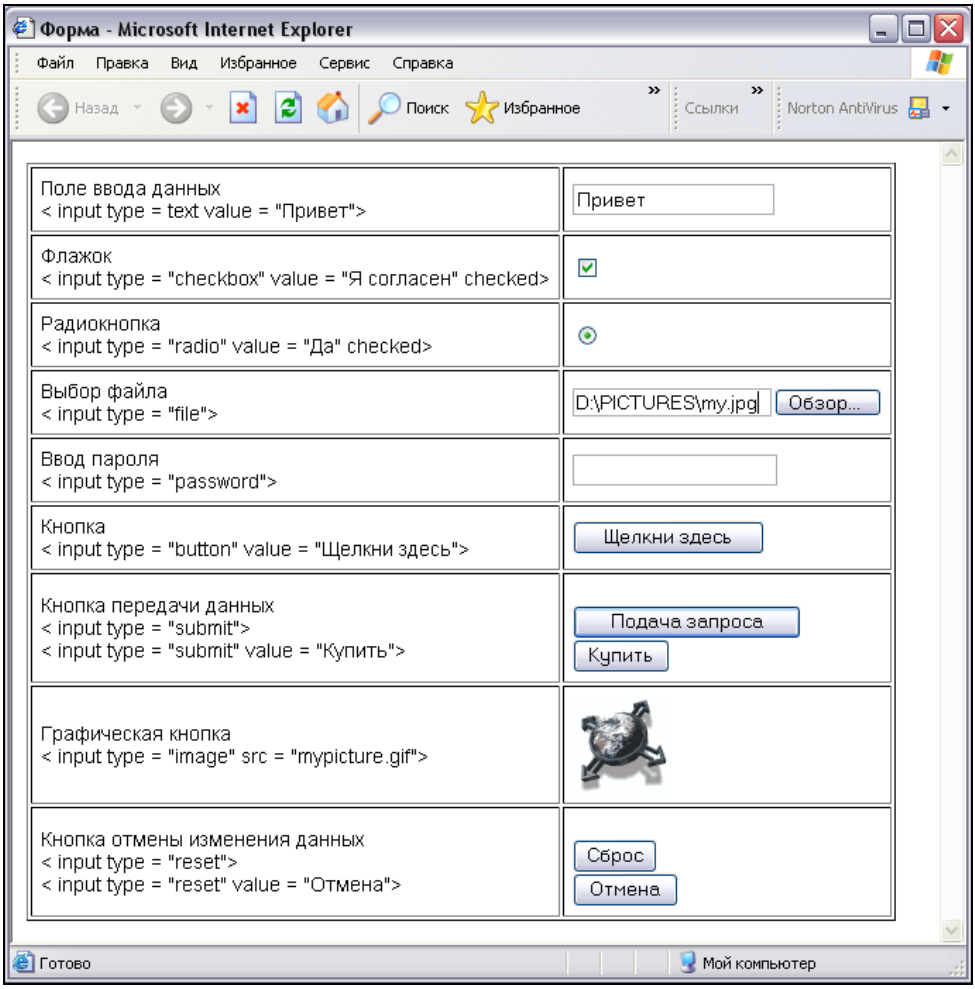


Рис. 1.3. Элементы, заданные тегом `<input>`

Кроме рассмотренных ранее, тег `<input>` имеет следующие атрибуты:

- `name` — имя (идентификатор) элемента. Если предполагается использовать элемент в составе формы для передачи его данных (значение атрибута `value`) на сервер, то применение атрибута `name` обязательно;
- `disabled` — делает элемент недоступным пользователю;
- `accesskey` — определяет клавишу быстрого доступа к элементу; значением атрибута является строка, содержащая букву или цифру. Если нажать указанную клавишу в сочетании с клавишей <Alt>, то элемент получит фокус. Например, `<input type="button" value="OK" accesskey="Y">`;

- `tabindex` — целое число, определяющее порядок перехода к элементу с помощью клавиши <Tab>; обычно указывается, когда в форме присутствует несколько элементов интерфейса, а порядок ввода данных в них важен.

В отличие от других рассмотренных выше элементов, радиокнопка не используется в одиночку. В группе из нескольких радиокнопок только одна из них может находиться в установленном состоянии. Щелчок на другой радиокнопке из той же группы устанавливает ее и одновременно сбрасывает ранее отмеченную кнопку. Чтобы организовать группу радиокнопок, достаточно сделать одинаковыми их имена (значения атрибута `name`). В одном и том же документе может быть несколько групп радиокнопок. Обычно (но не обязательно) радиокнопки из одной группы находятся рядом друг с другом.

Кнопка: тег *<button>*

Для создания кнопки можно использовать контейнерный тег `<button>`. Текст, изображение и другие видимые элементы, вставленные в этот тег, отображаются на кнопке, как показано в следующем примере (листинг 1.17).

Листинг 1.17. Примеры кнопок

```
<html>
  <head><title>Кнопка</title><head>
  <button>Щелкни здесь</button>
  <p>
  <button >
    <img src="auto.gif" width=80 height=50>
    <b>Хорошая машина</b>
  </button>
  <p>
  <button>
    <table border=1>
      <tr><th>Фамилия сотрудника</th><th>Зарплата</th></tr>
      <tr><td>Вася</td><td>100</td></tr>
      <tr><td>Маша</td><td>50</td></tr>
    </table>
  </button>
</html>
```

В этом примере на первой кнопке отображается надпись, на второй — графическое изображение и надпись, а на третьей — таблица.

Размер кнопки устанавливается автоматически в зависимости от размеров элементов, вставленных в тег `<button>`. Этот тег не имеет атрибутов, с помощью которых можно было бы задать размеры кнопки по своему усмотрению. Однако это можно сделать с помощью каскадных таблиц стилей.

Тег `<button>` имеет следующие атрибуты:

- `accesskey` — определяет клавишу быстрого доступа к кнопке; значением атрибута является строка, содержащая букву или цифру. Если нажать указанную клавишу в сочетании с клавишей <Alt>, то элемент получит фокус. Например, `<button accesskey="N">Щелкни здесь</button>`;
- `disabled` — делает кнопку недоступной пользователю.

Кнопка, заданная тегом `<button>`, может использоваться и в форме (тег `<form>`). Однако, поскольку тег `<button>` не имеет атрибутов `value` и `name`, его содержимое не может быть отправлено на сервер с помощью кнопки типа `submit`.

Раскрывающийся список: тег *<select>*

Тег `<select>` используется для создания раскрывающегося списка, из которого можно выбрать элемент. Это контейнерный тег, внутри которого используются теги `<option>`, предназначенные для задания элементов (опций) списка.

Тег `<select>` имеет следующие атрибуты:

- `align` — горизонтальное выравнивание списка; возможны следующие значения:
  - `absbottom` — выравнивание нижней границы списка по нижней границе текущей строки;
  - `absmiddle` — выравнивание середины списка по середине текущей строки;
  - `baseline` — выравнивание нижней границы списка по базовой линии текущей строки;
  - `bottom` — то же, что и `baseline`;
  - `top` — верхняя граница списка выравнивается по самому высокому элементу текущей строки;
  - `texttop` — верхняя граница списка выравнивается по самому высокому текстовому элементу текущей строки;
  - `left` — список располагается у левого края окна; текст и другие элементы обтекают его справа;

- `right` — список располагается у правого края окна; текст и другие элементы обтекают его слева;

- `accesskey` — определяет клавишу быстрого доступа к раскрывающемуся списку; значением атрибута является строка, содержащая букву или цифру. Если нажать указанную клавишу в сочетании с клавишей <Alt>, то элемент получит фокус. Например, `<select accesskey="S">`;
- `name` — имя (идентификатор) списка;
- `disabled` — делает список недоступным пользователю;
- `multiple` — обозначает возможность выбора из списка одновременно нескольких элементов;
- `size` — количество одновременно видимых элементов списка; по умолчанию — 1. Если это число больше 1, то список снабжается полосой прокрутки;
- `tabindex` — целое число, определяющее порядок перехода к элементу с помощью клавиши <Tab>; обычно используется, когда в форме присутствуют несколько элементов интерфейса, а порядок ввода данных в них важен.

Тег `<option>` имеет следующие атрибуты:

- `selected` — обозначает выбранный (выделенный) элемент списка;
- `value` — значение, ассоциированное с элементом списка.

Текстовое содержимое элемента списка указывается сразу после тега `<option>`.

В следующем примере приводятся три раскрывающихся списка (листинг 1.18).

Листинг 1.18. Раскрывающиеся списки

```
<html>
  <head><title>Раскрывающийся список</title><head>
  <select name="List1">
    <option value=1>Первый
    <option value=2>Второй
    <option value=3>Третий
    <option value=4>Четвертый
  </select>
  <p>
  <select name="List2" size=3>
    <option value="one">Один
```

```
    <option selected value="two">Два
    <option value="three">Три
    <option value="four">Четыре
  </select>
  <p>
  <b>Оцените этот сайт:</b>
  <select name="List3" size=4 align=top>
    <option value=3>Отлично
    <option selected value=2>Хорошо
    <option value=1>Так себе
    <option value=0>Плохо
  </select>
</html>
```

Теги форматирования текста, указанные до или после тега `<option>`, не действуют на вид элементов списка. Чтобы установить параметры шрифта для элементов списка, следует воспользоваться каскадными таблицами стилей для тега `<select>`.

Атрибут `value` предназначен для хранения значений, ассоциированных с элементами списка. Элемент списка на экране и ассоциированное с ним значение могут быть как одинаковыми, так и различными. Если раскрывающийся список находится в форме (тег `<form>`), то при отправке данных формы на сервер передается значение атрибута `value` выбранного элемента.

Примечание

Тег `<select>`, как и все другие, имеет атрибут `id`, значением которого является строка, содержащая идентификатор элемента (подобно имени переменной). Идентификатор используется в сценариях для доступа к объекту, соответствующему этому элементу документа. Атрибут `name` необходим для организации отправки данных (значения атрибута `value` выбранного элемента списка) на сервер. В сценарии можно получить доступ к элементу и по значению `name`, но иначе, чем по значению `id`. Поэтому, чтобы не запутаться, значения атрибутов `id` и `name` устанавливают одинаковыми, хотя это и не обязательно. Это же замечание касается тега `<form>` и других тегов, предназначенных для задания элементов формы (`<input>` и `<textarea>`).

Для создания иерархических списков внутри тега `<select>` можно использовать контейнерный тег `<optgroup>`, в который заключаются теги `<option>` элементов списка, объединяемых в одну группу. Атрибут `label` тега `<optgroup>` принимает в качестве значения текстовую строку, отображаемую в списке жирным курсивом. Это невыбираемый элемент списка, выполняющий

роль заголовка группы элементов. При этом выбираемые элементы группы, заданные посредством тега `<option>`, отображаются с левым отступом.

HTML-код, представленный в листинге 1.19, создает двухуровневый раскрывающийся список.

Листинг 1.19. Двухуровневый раскрывающийся список

```
<html>
  <head><title>Раскрывающийся список</title><head>
  <select name="mylist" size=9>
    <optgroup label="Фрукты">
      <option value=1>Яблоки
      <option value=2>Груши
    </optgroup>
    <optgroup label="Овощи">
      <option value=3>Капуста
      <option value=4 selected>Картофель
      <option value=5>Помидоры
    </optgroup>
    <option s value=6>Молоко
    <option value=7>Мясо
    <option value=8>Пиво
  </select>
</html>
```

Текстовая область: тег *<textarea>*

Контейнерный тег `<textarea>` позволяет создать элемент, называемый *текстовой областью*. Это прямоугольная область с полосами прокрутки. Внутри нее легко разместить многострочный текст, который можно сделать как редактируемым, так и доступным только для просмотра. Текстовую область обычно используют как большое поле ввода данных. Важной особенностью тега `<textarea>` является то, что все теги, заключенные в него, не выполняются браузером, а отображаются как обыкновенный текст.

Тег `<textarea>` имеет следующие атрибуты:

- `align` — горизонтальное выравнивание текстовой области; возможны следующие значения:
  - `absbottom` — выравнивание нижней границы области по нижней границе текущей строки;

- `absmiddle` — выравнивание середины области по середине текущей строки;
- `baseline` — выравнивание нижней границы области по базовой линии текущей строки;
- `bottom` — то же, что и `baseline`;
- `top` — верхняя граница области выравнивается по самому высокому элементу текущей строки;
- `texttop` — верхняя граница области выравнивается по самому высокому текстовому элементу текущей строки;
- `left` — текстовая область располагается у левого края окна; текст и другие элементы обтекают ее справа;
- `right` — область располагается у правого края окна; текст и другие элементы обтекают ее слева;

❒ `accesskey` — определяет клавишу быстрого доступа к текстовой области; значением атрибута является строка, содержащая букву или цифру. Если нажать указанную клавишу в сочетании с клавишей <Alt>, то элемент получит фокус. Например, `<textarea accesskey="T">`;

❒ `cols` — количество символьных позиций по горизонтали для отображения текста;

❒ `rows` — количество строк, одновременно видимых в окне текстовой области;

❒ `readonly` — устанавливает текстовую область в режим просмотра (только для чтения); если этот атрибут не использовать, то текстовая область будет работать как поле ввода и редактирования данных;

❒ `name` — имя (идентификатор) текстовой области;

❒ `disabled` — делает область недоступной пользователю;

❒ `tabindex` — целое число, определяющее порядок перехода к элементу с помощью клавиши <Tab>; обычно используется, когда в форме присутствуют несколько элементов интерфейса, а порядок ввода данных в них важен;

❒ `wrap` — определяет режим автоматического переноса символов на другую строку; возможные значения: `physical`, `virtual` и `off` (выключено). По умолчанию включен автоматический перенос посимвольно.

Вот пример текстовой области (листинг 1.20).

Листинг 1.20. Пример текстовой области

```
<html>
  <head><title>Текстовая область</title><head>
  <body bgcolor=#e0e0e0>
    <textarea cols= 15 rows=5>
Привет
    </textarea>
      <body>
</html>
```

Все теги, включенные в контейнер `<textarea>`, не интерпретируются браузером, а выводятся в текстовой области как обыкновенный текст.

Примечание

Тег `<textarea>`, как и все другие, имеет атрибут `id`, значением которого является строка, содержащая идентификатор элемента (подобно имени переменной). Идентификатор используется в сценариях для доступа к объекту, соответствующему этому элементу документа. Атрибут `name` необходим для организации отправки данных на сервер.

Теги *<fieldset>*, *<legend>* и *<label>*

Для создания выделенного рамкой блока элементов пользовательского интерфейса служит контейнерный тег `<fieldset>`. В нем может находиться контейнерный тег `<legend>`, содержимое которого отображается в верхней части блока (на рамке). Обычно это текстовый заголовок группы элементов, хотя можно использовать графические изображения и другие элементы. В следующем примере блок содержит три радиокнопки (листинг 1.21).

Листинг 1.21. Блок с радиокнопками

```
<html>
  <fieldset>
    <legend><b>Выберите продукт</b></legend>
    <input type=radio name=product value="auto">Автомобиль
      <img src="auto.gif"><br>
    <input type=radio name=product value="flowers" checked>Букет
      <img src="flowers.jpg"><br>
    <input type=radio name=product value="nothing">Ничего не надо
  </fieldset>
<html>
```

Размещение радиокнопок в одном блоке еще не делает их селекторными переключателями, среди которых только один может быть в установленном состоянии. Для этого, как уже отмечалось, требуется, чтобы они имели одинаковые значения атрибута `name`.

Тег `<fieldset>` не имеет атрибутов, с помощью которых можно было бы установить размеры блока. Высота блока зависит от количества расположенных в нем по вертикали элементов и их размеров. Однако прямоугольная рамка блока растягивается на всю ширину окна браузера. Установить размеры блока по своему усмотрению можно с помощью таблиц стилей.

Тег `<legend>` имеет атрибуты:

- `align` — горизонтальное выравнивание содержимого, которое располагается на верхней рамке блока; возможны значения:
  - `left` — по левому краю (значение по умолчанию);
  - `right` — по правому краю;
  - `center` — по центру;
- `accesskey` — определяет клавишу быстрого доступа к первому элементу пользовательского интерфейса, расположенному в контейнере `<legend>`. В случае, когда `<legend>` содержит только текст, быстрая клавиша не действует;
- `tabindex` — целое число, определяющее порядок перехода с помощью клавиши <Tab> к первому элементу пользовательского интерфейса, расположенному в контейнере `<legend>`. В случае, когда `<legend>` содержит только текст, быстрая клавиша не действует.

Примечание

Формально тег `<legend>` имеет еще атрибут `valign` вертикального выравнивания с возможными значениями `top` и `bottom`, а атрибут `align` принимает, кроме указанных выше, такие же значения. Однако они не действуют в Internet Explorer 6.0.

Обычно элементы пользовательского интерфейса сопровождаются поясняющими надписями (метками). Это можно сделать, расположив текст надписи до или после тега элемента или воспользовавшись таблицей для задания требуемого положения. Вместе с тем, с помощью контейнерного тега `<label>` можно определить автоматическое расположение надписи рядом с элементом. Например:

```
<label>
  Введите Ваше имя
```

```
  <input type="text">
</label>
```

Тег `<label>` обладает атрибутами `accesskey` и `for`. Атрибут `accesskey` такой же, как и для описанных выше элементов пользовательского интерфейса. Атрибут `for` позволяет указать, к какому элементу относится надпись: в качестве значения он принимает значение атрибута `id` элемента. Напомним, что все элементы обладают атрибутом `id` (идентификатор). Например:

```
<input type="radio" id="r1" name="product" value="auto">
<label for="r1">Автомобиль</label><br>
<input type="radio" id="r2" name=product value ="flowers" checked>
<label for="r2">Цветы</label>
```

Примечание

В принципе все, что позволяет сделать тег `<label>`, осуществимо с помощью тега `<div>`.

1.3. Каскадные таблицы стилей

Собственные средства HTML (теги и их атрибуты) выполняют две основные роли: поддержку структуры документа (состав и взаимосвязи элементов) и определение внешнего вида визуальных элементов. Идея как-то отделить параметры внешнего вида документа от массива дескрипторов, определяющих его структуру, воплотилась в технологии, называемой *каскадными таблицами стилей* (Cascading Style Sheets, CSS). Таблица стилей, подобно шаблону форматирования текстов, может быть разработана отдельно от конкретного HTML-документа, а затем применена к нему. Изменяя содержимое таблицы стилей, можно изменять внешний вид HTML-документов, не затрагивая их структуры и информационного содержания. Одна и та же таблица стилей может применяться к нескольким документам, и, наоборот, к одному и тому же документу может быть применено несколько таблиц стилей. В последнем случае браузер учитывает приоритеты таблиц и по определенным правилам разрешает возникающие конфликты, в результате чего таблицы выстраиваются неким каскадом. Отсюда и название — каскадные таблицы стилей.

Кроме технологичности стилизации HTML-документов, CSS обеспечивают еще две важные вещи: произвольное позиционирование элементов и создание визуальных эффектов, таких как полупрозрачность и трансформации графических изображений и текстов.

CSS содержит наборы стилевых параметров (правила форматирования), имеющие простую и отнюдь не табличную структуру. Например, если требу-

ется определить для всех заголовков 1-го уровня шрифт Arial 48 пунктов красного цвета, то соответствующее правило можно записать так:

```
H1 {font-family: Arial; font-size: 48pt; color: red}
```

В записи правила первый элемент, называемый *селектором*, представляет собой имя тега без угловых скобок, к которому относится последующий набор параметров, заключенный в фигурные скобки. Параметры в наборе отделяются друг от друга точкой с запятой (`;`) и представляют собой пары *`имя_параметра:значение`*. Имя параметра отделяется от значения двоеточием (`:`). Имя тега в записи правила может быть в любом регистре, а имена параметров и их значения записываются только так, как они определены в спецификации CSS.

1.3.1. Встраивание таблиц стилей в HTML-документ

Для применения каскадной таблицы стилей к HTML-документу необходимо ее связать с ним или встроить в него. Это можно сделать четырьмя способами.

- Вставка непосредственно в HTML-документ — правила таблицы стилей заключаются в контейнерный тег `<style>`, который обычно помещается в контейнер `<head>`.

  Пример:

  ```
  <style>
    h1 {color: blue; font-size: 24pt}
    b  {font-style: italic}
  </style>
  ```

- Вставка в элемент — набор параметров стиля в виде строки без фигурных скобок присваивается в качестве значения атрибута `style` тега того элемента, к которому применяется.

  Пример:

  ```
  <h1 style="color:blue; font-size:24pt">
  ```

- Импорт — вставка таблицы стилей из внешнего файла с помощью команды:

  `@import: url(`*`URL-адрес файла таблицы стилей`*`)`

  Файл таблицы стилей является текстовым файлом с расширением css. Оператор `@import` используется перед другими правилами таблицы стилей в контейнере `<style>` или в css-файле.

Пример:

```
<style>
  @import: url(http://www.myserver.ru/xyz/mystyle.css)
</style>
```

- Связывание с таблицей стилей в файле — использование ссылки на файл таблицы стилей с помощью тега `<link>`, помещаемого в контейнер `<head>`:

```
<link rel=stylesheet type=text/css
      href="URL-адрес файла таблицы стилей">
```

Пример:

```
<link rel=stylesheet type=text/css
      href="http://www.myserver.ru/xyz/mystyle.css">
```

Таблица стилей, вставленная с помощью тега `<style>`, действует на элементы только текущего HTML-документа, в котором этот тег находится. Если ту же таблицу необходимо применить и к другим документам, то ее код придется повторить в соответствующих HTML-кодах. При этом возрастает общий объем файлов сайта, а также трудозатраты в случае необходимости изменить его стиль. Чтобы избежать этого, используют импорт или связывание таблиц из внешних css-файлов. Когда требуется изменить параметры стилей для отдельных элементов (например, их координаты), вставляют атрибут `style`. Возможно также комбинирование всех способов встраивания таблиц.

В записях таблиц стилей можно использовать оператор комментария, чтобы вставить не отображаемые в окне браузера и не влияющие на форматирование элементов документа пояснения. Строки, заключенные между символами `/*` и `*/`, являются *комментарием*.

Старые браузеры могут не поддерживать CSS и тег `<style>`. Чтобы они не отображали таблицы стилей в виде текста, содержимое тега `<style>` заключают в тег комментария `<! --...-->`.

Пример:

```
<style>
  <!--
  h1 {color: blue; font-size: 24pt}    // заголовок 1-го уровня
  b  {font-style: italic}              // выделение жирным шрифтом
  -->
</style>
```

Тег `<style>` имеет следующие атрибуты:

- `type` — для каскадных таблиц стилей всегда имеет значение `text/css`;

- `media` — определяет тип устройства вывода. Браузеры обычно используют следующие значения: `screen` (экран), `print` (печать) и `all` (все). Вы можете создать стили отдельно для отображения документа на экране монитора и для вывода на печать.

1.3.2. Правила форматирования

Правило форматирования во всех вариантах вставки в HTML-документ, кроме использования атрибута `style`, содержит селектор и следующий за ним набор параметров, разделенных точкой с запятой (`;`) и заключенный в фигурные скобки (`{}`). Параметры задаются парой *имя_параметра:значение*. Короче говоря, правила имеют следующий синтаксис:

```
Селектор: {имя_параметра1:значение; имя_параметра2:значение; ...
           имя_параметраN:значение}
```

В случае использования атрибута `style` синтаксис несколько отличается:

```
style="имя_параметра1:значение; имя_параметра2:значение; ...
       имя_параметраN:значение"
```

В качестве селектора используется либо имя тега (без угловых скобок), либо значение его атрибута `id`, перед которым указывается символ `#`. Пусть, например, в HTML-документе имеется несколько заголовков 1-го уровня, задаваемых тегами `<h1>`. Тогда следующая таблица стилей изменит вид всех таких заголовков:

```
<style>
  h1 {color: red; font-size: 24pt}
</style>
```

Если в HTML-документе имеется заголовок 1-го уровня с явно указанным атрибутом `id` (например, `<h1 id=myh1>Добро пожаловать!</h1>`), то следующая таблица стилей повлияет только лишь на этот заголовок, оставив без изменения все остальные заголовки 1-го уровня:

```
<style>
  #myh1 {color: red; font-size: 24pt}
</style>
```

Когда к различным элементам (тегам) требуется применить одинаковые наборы параметров, тогда селекторы можно сгруппировать, перечислив их через запятую перед одним и тем же набором параметров, как в следующем примере:

```
h3, b {color:blue; font-style:italic; font-size:14pt; font-weight:bold}
```

Для одного и того же тега можно задать разновидности стилей, называемые *классами*. Для этого селектор формируется как имя тега, за которым через разделительную точку следует имя класса. Имя класса можно использовать в теге как значение атрибута `class`. В следующем примере показано, как используются классы таблицы стилей для заголовков первого уровня (листинг 1.22).

Листинг 1.22. Использование классов таблицы стилей

```
<html>
  <head><title>Классы стилей</title>
    <style>
      h1.курсив       {font-style: italic}
      h1.подчеркнутый {text-decoration: underline}
    </style>
  </head>
  <h1>Обычный заголовок</h1>
  <h1 class=курсив> Заголовок курсивом</h1>
  <h1 class= подчеркнутый>Заголовок подчеркнутый</h1>
</html>
```

Для гиперссылок, задаваемых тегом `<a>` с атрибутом `href`, в таблицах стилей можно использовать так называемые *псевдоклассы стилей*:

- `a: link` — неиспользованная ссылка;
- `a:visited` — использованная ссылка;
- `a:active` — активная ссылка;
- `a:hover` — ссылка, на которой находится указатель мыши.

Пример:

```
<style>
  a: link   {color: blue}
  a:visited {color: green}
  a:active  {color: #ffff00}
  a:hover   {color: red; font-weight: bold}
</style>
```

Как известно, одни элементы могут быть вложены в другие. Иногда требуется задать особенный стиль для элемента конкретного типа, если только он вложен в другой определенный тег. В противном случае для элементов этого типа должны выполняться установки по умолчанию или другие наборы сти-

левых параметров. Допустим, мы хотим, чтобы выделение текста жирным шрифтом (тег <b>) отображалось курсивом, если только оно применяется внутри абзаца (тег <p>). Во всех остальных случаях содержимое тега <b> должно отображаться жирным, но прямым шрифтом, как это принято по умолчанию. Использование просто правила `b {font-style: italic}` привело бы к повсеместному отображению содержимого тегов <b> курсивом. В подобных случаях используют так называемые *контекстные селекторы*, которые задают исключения из общих правил форматирования, определенных с помощью простых селекторов или по умолчанию.

Контекстный селектор состоит из имени тега-контейнера и имени вложенного в него тега, разделенных пробелом (в отличие от группировки тегов с одинаковыми стилями). Например:

```
p b {font-style: italic}
```

При создании таблиц стилей следует учитывать так называемый механизм наследования стилевых параметров вложенными элементами. Так элемент, вложенный в некоторый контейнер, наследует некоторые стилевые параметры, заданные для этого контейнера. Однако не все параметры могут наследоваться. Если необходимо заблокировать наследование некоторых параметров элементом, то для него их следует задать отдельно.

Тег <body> используется в качестве контейнера для всех других видимых элементов HTML-документа. Поэтому для него следует задавать стилевые параметры, общие для всех элементов, если только для них заданы свои собственные (иначе говоря, параметры по умолчанию).

Пример:

```
body {color:black; background:url(mytexture.gif);
      font-family:Arial; font-size:12pt}
```

1.3.3. Применение нескольких таблиц стилей

К одному и тому же HTML-документу можно применить несколько таблиц стилей. Например, в одной таблице могут быть определены правила, связанные со шрифтами, в другой — с позиционированием, а в третьей — с визуальными эффектами (фильтрами). Кроме того, как авторы документов, так и читатели способны влиять на их внешнее представление в окне браузера. При использовании нескольких таблиц стилей могут возникнуть противоречия, которые браузер разрешает по некоторым правилам, учитывая приоритеты (веса) таблиц.

По умолчанию приоритет таблиц стилей пользователя меньше, чем приоритет таблиц автора документа. Если между ними возникает конфликт, то при-

меняются таблицы автора. Самым низким приоритетом обладают стилевые параметры, принимаемые браузером по умолчанию.

Приоритет импортируемых с помощью оператора `@import` таблиц определяется порядком импортирования: каждый последующий оператор импортирует таблицу с более высоким приоритетом.

При использовании таблиц стилей на первых порах достаточно запомнить следующую последовательность приоритетов (от низшего к высшим):

1. Связанная таблица (тег `<link>`).
2. Импортируемая таблица (оператор `@import`).
3. Правило с именем тега в качестве селектора.
4. Правило с классом в качестве селектора (на это правило элемент ссылается с помощью значения атрибута `class`).
5. Правило со значением атрибута `id` тега элемента.
6. Правило, заданное как значение атрибута `style` тега элемента.

1.3.4. Единицы измерения

Некоторые стилевые параметры (например, размер шрифта, координаты позиционирования, размеры элементов и отступов) имеют числовые значения, для которых следует задавать единицы измерения. Единицы измерения могут быть относительными и абсолютными. Использование относительных единиц предпочтительней, поскольку не требуется знать параметров конкретной системы отображения (монитор, принтер и др.).

Относительные единицы измерения: `em`, `ex`, `en`, `px`, `%`. Первые три из них связаны с типографскими терминами и соотносятся с размерами других символов. В Internet Explorer, начиная с версии 4, `em` и `ex` заменены на `pt`, а `en` — на `px`. `px` обозначает пикселы — элементы экрана, размер которых зависит от настроек монитора и видеокарты компьютера пользователя. Проценты (`%`) задаются как числа (целые или с дробной частью), показывающие отношение к единице длины (обычно к размеру шрифта текущего элемента).

Абсолютные единицы измерения:

- `in` — дюйм (1 in = 2,54 cm);
- `cm` — сантиметр;
- `mm` — миллиметр;
- `pt` — пункт (1 pt = 1/72 in);
- `pc` — пика (1 pc = 12 pt = 1/6 in).

1.3.5. Шрифты

Шрифты различаются внешним видом, размером, стилем и "жирностью". Эти параметры можно задать с помощью каскадных таблиц стилей. Если указанный в таблице шрифт не установлен на компьютере пользователя, то можно предусмотреть его загрузку с сервера, на котором хранится документ. Ниже перечислены стилевые параметры для шрифтов.

- `font-family` — определяет список семейств шрифтов. Обычно задаются несколько похожих шрифтов в порядке предпочтения на тот случай, если в компьютере пользователя нет нужного шрифта. Названия шрифтов разделяются запятыми. Если название шрифта состоит из нескольких слов, то оно заключается в кавычки. Список шрифтов желательно завершить родовым именем шрифта: `serif`, `sans-serif`, `cursive`, `fantasy` или `monospace`. Например, для шрифта Times родовым является `serif`, для Helvetica — `sans-serif`, для Courier — `monospace`.

  Примеры:

  ```
  <style>
    h1 {font-family: "Times New Roman", serif}
  </style>
  <body style='font-family: "Times New Roman", serif'>
  ```

- `font-size` — размер шрифта. Значение может быть задано различными способами:
  - абсолютный размер, задаваемый с помощью ключевых слов: `xx-small`, `x-small`, `medium`, `large`, `x-large`, `xx-large`. Эти значения представляют индексы таблицы размеров шрифтов, поддерживаемой браузером. По умолчанию используется значение `medium` (средний);
  - относительный размер, задаваемый с помощью ключевых слов: `larger` (больше) и `smaller` (меньше). Эти значения интерпретируются относительно таблицы размеров шрифтов браузера и размера шрифта элемента-родителя. Например, если элемент-родитель имеет шрифт размером `medium`, то значение `larger` для элемента-потомка сделает размер шрифта равным `large` (большой). В CSS1 масштабирующий множитель равен 1,5, в CSS2 — 1,2. Размер шрифта относительно элемента-родителя можно задавать и в процентах;
  - размер, задаваемый в абсолютных единицах длины *(см. разд. 1.3.4)*.

- `font-weight` — жирность шрифта. Возможные значения задаются ключевыми словами `normal`, `bold` (жирный), `bolder` (жирнее), `lighter` (светлее) или одним из девяти целых чисел от 100 до 900. Ключевому слову `normal`

соответствует числовое значение 400; ключевое слово `bold` задает обычный жирный (полужирный) шрифт и соответствует числовому значению 700. Следует иметь в виду, что в текущем семействе может и не быть шрифта с заданной степенью жирности. В этом случае можно лишь гарантировать, что шрифт с большим значением параметра `font-weight` будет не светлее, чем шрифт с меньшим значением этого параметра.

- `font-style` — стиль шрифта. Возможные значения: `normal` (нормальный прямой), `italic` (курсив) и `oblique` (наклонный). По умолчанию применяется значение `normal`. Если уже имеется готовый шрифт, соответствующий заданному стилю, то он будет применен. В противном случае текущий шрифт будет изменен программным способом.
- `font-variant` — вариант стиля шрифта. Возможные значения: `normal` (принимается по умолчанию) и `small-caps`. Значение `normal` не влияет на отображение шрифта. Значение `small-caps` заменяет строчные буквы прописными, но делает их несколько меньшими по размеру, чем прописные буквы текущего шрифта.
- `font` — позволяет установить сразу несколько свойств шрифта в одном определении: `font-style`, `font-variant`, `font-weight`, `font-size`, `line-height`, `font-family`. Значения этих параметров указываются через пробел в том порядке, в котором они были перечислены. Допустимо не указывать первые три параметра, что соответствует значению `normal`, принятому для них по умолчанию. Если задается высота строки (`line-height`), то ее значение отделяется от размера шрифта (`font-size`) прямым слэшем (`/`). Если список семейств шрифтов (`font-family`) содержит более одного элемента, то последние отделяются друг от друга запятыми.

  Пример:

  ```
  <style>
    body {font: italic 10pt/12pt "Times", serif}
  </style>
  ```

- `@font-face` — позволяет указать семейство шрифтов и URL-адрес, по которому можно получить шрифт, если он отсутствует на компьютере пользователя. Данный оператор имеет два параметра, отделяемых друг от друга точкой с запятой:
  - `font-family` — название семейства шрифтов;
  - `src` — URL-адрес шрифта. При этом используется синтаксис:

    `src:url(`*URL-адрес*`)`

    Открывающаяся круглая скобка следует сразу же за ключевым словом `url`.

Пример:

```
<style>
  @font-face {font-family: MyFont;
              src:url(http://myserver.ru/MyFont.eot)}
</style>
```

1.3.6. Цвет и фон

Цвет текста определяется с помощью параметра `color`, а цвет фона элемента — с помощью параметра `background-color`. По умолчанию цвет фона элемента определен как прозрачный (transparent). Значения этих параметров задаются именем цвета или числовым представлением в системе RGB.

Числовое представление цвета допускает следующие варианты:

- шестнадцатеричное число (например, `#ffff00`);
- тройка целых десятичных чисел, каждое из которых представляет яркость одной из RGB-составляющих цвета в диапазоне от 0 до 255. При этом три числа передаются в качестве параметров функции `rgb()`. Например, `rgb(255,255,0)`;
- тройка вещественных чисел со знаками `%`, каждое из которых представляет яркость одной из RGB-составляющих цвета в диапазоне от 0 до 100%. При этом три числа передаются в качестве параметров функции `rgb()`. Например, `rgb(100%,100%,0%)`.

Пример:

```
<style>
  body {color: blue; background-color:#e0e0e0}  /* текст синий,
                                                   фон серый */
  h1   {color: #ff0000; background-color:white} /* заголовок 1-го уровня
                                                  красный на белом фоне */
  h2   {color: rgb(0, 255, 0)}   /* заголовок 2-го уровня
                                    зеленый на сером фоне */
  h3   {color: rgb(0, 255, 0)}   /* заголовок 3-го уровня
                                    зеленый на сером фоне (см. body) */
  h4   {color: rgb(100%, 50%, 0%); background-color: black}  /* заголовок
                                 4-го уровня оранжевый на черном фоне */
</style>
```

В качестве фона элемента (или всего документа, в частности) можно использовать графическое изображение из файла по указанному URL-адресу. При этом

можно задать способ заполнения фоновым изображением области элемента, позиционирование изображения, а также указать, должно ли оно перемещаться при прокрутке документа. Все это можно определить с помощью следующих параметров:

- `background-image` — принимает в качестве значения `url(`*`URL-адрес изображения`*`)` или `none` (отсутствие изображения).

  Пример:

  ```
  <style>
    body {background-image: url(/myimages/picture.jpg)}
  </style>
  ```

- `background-repeat` — определяет способ заполнения области элемента изображением, заданным параметром `background-image`. Возможны следующие значения:
  - `repeat` — заполнение по горизонтали и по вертикали (по умолчанию);
  - `repeat-x` — заполнение по горизонтали;
  - `repeat-y` — заполнение по вертикали;
  - `no-repeate` — изображение выводится в единственном экземпляре;
- `background-attachment` — определяет, будет ли фоновое изображение прокручиваться при пролистывании документа. Значением по умолчанию является `scroll` (будет прокручиваться). Для фиксации изображения используется значение `fixed`;
- `background-position` — определяет начальное положение фонового изображения с помощью двух значений (горизонтальной и вертикальной координат), разделенных запятой. Значения задаются в виде процентов, в абсолютных значениях длины или в виде ключевых слов: `top`, `center`, `bottom`, `left`, `right`. Значения по умолчанию: `left`, `top`;
- `background` — позволяет установить значения рассмотренных выше свойств `background-color`, `background-image`, `background-repeat` и `background-attachment`. Эти значения указываются через пробел. Значение `transparent` означает отсутствие фона.

  Пример:

  ```
  <style>
    body {background: blue url(/myimages/picture.jpg) repeat-x}
  </style>
  ```

Внимание

В Windows XP при использовании неклассического стиля оформления кнопок и окон для вставки графического изображения в качестве фона кнопки рекомендуется кроме параметра `background-image` использовать еще и параметр цвета фона `background-color`. В противном случае не удастся отобразить картинку как фон кнопки.

1.3.7. Размеры, поля, отступы и границы

Каждый видимый элемент HTML-документа занимает на экране некоторую прямоугольную область, размеры которой определяются параметрами `height` (высота) и `width` (ширина). Значения этих параметров задаются числом с указанием единиц измерения или ключевым словом `auto`. В последнем случае размер элемента выбирается автоматически так, чтобы по возможности отобразить его содержимое целиком. По умолчанию параметры `height` и `width` имеют значение `auto`.

Содержимое элемента HTML-документа может быть окружено рамками (прямоугольной границей). Пространство между содержимым и границей называется *отступом*. Границы, в свою очередь, окружены прямоугольными областями, называемыми *полями*. Иначе говоря, с точки зрения форматирования элемент можно представить в виде четырех вложенных прямоугольников (блоков): содержимого элемента, отступов от границ, собственно границ (рамок) и внешних полей (рис. 1.4).

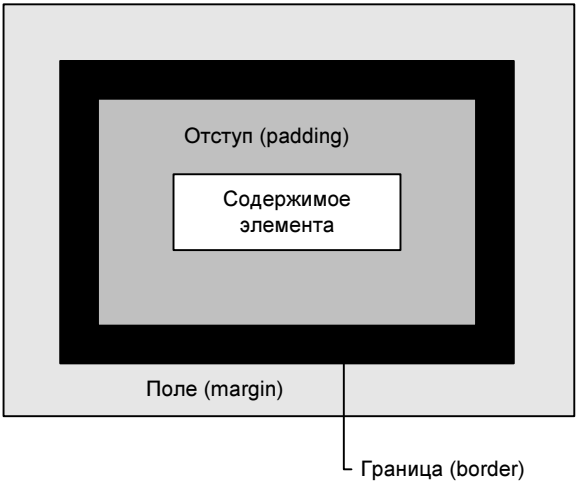


Рис. 1.4. Блоковая модель форматирования

Размеры отступов, границ и полей можно задавать по своему усмотрению. Однако по умолчанию они имеют значение 0, т. е. не видны и не занимают место. Параметры `height` и `with` определяют размеры содержимого элемента. Если рамка (граница) сделана видимой, то отступы задают положение содержимого (например, текста или изображения) относительно этой рамки. Внешние поля позволяют позиционно выделить содержимое элемента с рамкой от других элементов документа. Так например, обычно желательно, чтобы текст, обтекающий картинку, находился на некотором расстоянии от нее. Если рамку сделать невидимой, то положение относительно внешних элементов можно регулировать чем-либо одним: размерами или полей, или отступов.

Для полей и отступов можно задать размеры верхних (`top`), правых (`right`), левых (`left`) и нижних (`bottom`) их сторон с помощью следующих параметров:

- `margin-top`, `margin-right`, `margin-left`, `margin-bottom` — размеры полей;
- `padding-top`, `padding-right`, `padding-left`, `padding-bottom` — размеры отступов.

Значения этих параметров могут быть заданы числом с указанием единиц измерения или ключевым словом `auto`.

Кроме того, можно использовать параметры `margin` и `padding` (без уточнений типа `top`, `right` и т. п.). В качестве значений этих параметров можно перечислить через пробел размеры всех четырех сторон или только некоторых из них в следующем порядке: `top`, `right`, `left`, `bottom`. Если указано только одно значение, то оно будет задавать размер всех сторон. Если указаны два или три значения, то остальные будут равны размерам соответствующих противоположных сторон.

Размеры четырех сторон границы (рамки) можно задать с помощью следующих параметров: `border-top-width`, `border-right-width`, `border-left-width`, `border-bottom-width`.

Эти параметры принимают в качестве значений число с указанием единиц измерения (но не проценты) или ключевые слова: `thin` (тонкая), `medium` (средняя), `thick` (толстая). Значением по умолчанию является `medium`.

Кроме того, можно использовать параметр `border-width` (без уточнений типа `top`, `right` и т. п.) для задания размеров сразу всех или только некоторых сторон границы, подобно тому, как применяются описанные выше параметры `margin` и `padding`.

Границам, кроме размеров, можно назначить еще стиль и цвет. Задание стиля (типа) границы в явном виде необходимо, чтобы она была отображена (по умолчанию граница не видна). Если стиль границы не задан в явном виде, другие ее параметры не будут действовать.

Стиль четырех сторон границы задается следующими параметрами: `border-top-style`, `border-right-style`, `border-left-style`, `border-bottom-style`.

Эти параметры принимают в качестве значений ключевые слова:

- `none` — граница не отображается (по умолчанию), даже если указан размер границы (`border-width`);
- `solid` — сплошная линия;
- `double` — двойная линия (сумма толщины двух линий и промежутка между ними определяется параметром `border-width`);
- `grove` — вдавленный желобок;
- `ridge` — выпуклый валик;
- `inset` — весь блок элемента вдавлен;
- `outset` — весь блок элемента выпуклый.

Кроме того, можно использовать параметр `border-style` (без уточнений типа `top`, `right` и т. п.) для задания стиля сразу всех или только некоторых сторон границы, подобно тому, как применяются описанные выше параметры `margin` и `padding`.

Цвет для четырех сторон границы задается следующими параметрами: `border-top-color`, `border-right-color`, `border-left-color`, `border-bottom-color`.

Кроме того, можно использовать параметр `border-color` (без уточнений типа `top`, `right` и т. п.) для задания цвета или графического изображения сразу всех или только некоторых сторон границы, подобно тому, как применяются описанные выше параметры `margin` и `padding`. В следующем примере задается одинаковый красный цвет для всех сторон границы вокруг элемента, соответствующего тегу `<div>`:

```
<style>
  div {border-style:solid; border-color:red}
</style>
```

Здесь стиль границы задан не только для его определения ее внешнего вида, но и для того, чтобы граница была видна.

1.3.8. Текст

Для элементов, содержащих текст, кроме параметров шрифта *(см. разд. 1.3.5)* можно задать и другие параметры стиля.

- `letter-spacing` — интервал между буквами. Принимает значения: `normal` (по умолчанию) или число с указанием единиц измерения. В последнем случае к расстоянию между буквами, установленному по умолчанию, прибавляется заданное расстояние. При этом увеличивается и расстояние между словами.

  Пример:

```
<html>
  <head><title>Тексты</title>
    <style>
      #p1 {letter-spacing:10pt}
      #p2 {letter-spacing:15pt}
      #p3 {letter-spacing:20pt}
    </style>
  </head>
  Один два три
  <p id=p1>Один два три</p>
  <p id=p2>Один два три</p>
  <p id=p3>Один два три</p>
</html>
```

- `line-height` — высота текущей строки. Принимает значения: `normal`, число, число с указанием единиц измерения. Число без единиц измерения интерпретируется как коэффициент, на который следует умножить размер шрифта текущего элемента. Если указаны проценты (`%`), то имеется в виду соотношение с размером текущего шрифта; оно должно быть более 100%. Значение по умолчанию зависит от браузера.

  Пример:

```
<html>
  <head><title>Тексты</title>
    <style>
      #p1 {line-height:20pt}
      #p2 {line-height:1.4}
      #p3 {line-height:150%}
    </style>
  </head>
```

```
  Один два три
  <p id=p1>Один два три</p>
  <p id=p2>Один два три</p>
  <p id=p3>Один два три</p>
</html>
```

Параметры списка:

- `list-style-image` — определяет URL-адрес графического изображения, используемого в качестве маркера для каждого элемента списка. Принимает значения: `none` (по умолчанию) или URL-адрес;
- `list-style-position` — указывает положение маркеров элементов списка: внутри или вне "тела" списка. Принимает значения: `inside` (внутри) или `outside` (вне). Значением по умолчанию является `outside`;
- `list-style-type` — определяет вид маркера элементов списка. Принимает значения: `none`, `circle` (окружность), `disk` (затемненный круг), `square` (квадрат), `decimal` (десятичные номера), `lower-alpha` (строчные буквы), `upper-alpha` (прописные буквы), `lower-roman` (маленькие римские цифры), `upper-roman` (большие римские цифры). Значением по умолчанию является `disk`;
- `list-style` — позволяет задать значения сразу всех или некоторых параметров, описанных выше. Значения указываются через пробел в следующем порядке: *`вид_маркера положение URL-адрес`*.

❒ `text-align` — определяет выравнивание текста относительно содержащего его элемента (контейнера). Принимает значения: `left`, `right`, `center`, `justify`.

❒ `text-decoration` — определяет сочетание текста с горизонтальной линией. Принимает значения: `none` (по умолчанию, нет линии), `overline` (линия над текстом), `underline` (подчеркивание), `line-through` (зачеркивание).

Пример:

```
<html>
  <head><title>Тексты</title>
    <span style="text-decoration:line-through;
                font-size:24">пять</span>
  <h1>&#8730;<span style="text-decoration:overline">2</span></h1>
</html>
```

Данный код выводит в окно браузера зачеркнутое слово "пять" и корень квадратный из числа 2. Для отображения знака квадратного корня исполь-

зовался специальный символ `√` ("галочка") и линия над текстом (числом 2).

- `text-indent` — определяет отступ. Принимает в качестве значения число с указанием единиц измерения. По умолчанию отступ равен 0. Обычно используется в абзацах для задания красной строки.
- `text-transform` — преобразует текст в зависимости от значения:
  - `capitalize` — делает прописной (заглавной) первую букву каждого слова в элементе;
  - `uppercase` — делает все буквы в элементе прописными;
  - `lowercase` — делает все буквы в элементе строчными;
  - `none` — снимает все унаследованные установки; это значение принимается по умолчанию.
- `vertical-align` — определяет расположение по вертикали. Принимает следующие значения:
  - `baseline` — устанавливает выравнивание базовой линии элемента по базовой линии родительского элемента;
  - `middle` — устанавливает вертикальную среднюю линию элемента на уровне базовой линии родительского элемента с добавлением половины ширины строки последнего; это значение принято по умолчанию;
  - `super` — элемент отображается в виде верхнего индекса;
  - `sub` — элемент отображается в виде нижнего индекса;
  - `text-top` — выравнивает элемент по верху текста, набранного шрифтом родительского элемента;
  - `text-bottom` — выравнивает элемент по низу текста, набранного шрифтом родительского элемента;
  - `top` — выравнивает верх элемента по верху самого высокого элемента текущей строки;
  - `bottom` — выравнивает низ элемента по низу самого низкого элемента текущей строки.

1.3.9. Обтекание и видимость

Перечисленные ниже параметры позволяют для данного элемента указать, как будут располагаться относительно него другие элементы (т. е. как они будут обтекать данный элемент).

- `clear` — позволяет указать, что следующие элементы должны быть показаны ниже элемента, выровненного по левому или правому краю. По умолча-

нию текст обтекает такие элементы. Параметр принимает значения: `none`, `both`, `left` и `right`.

- `float` — определяет обтекание элемента другими слева или справа вместо помещения под ним. Принимает значения: `none`, `left` и `right`.

Нередко случается так, что содержимое элемента выходит за его границы. Например, графическое изображение или текст могут не поместиться полностью в содержащем его контейнере (`<div>`, `<span>` и др.). В таких случаях используется параметр `overflow`, который принимает следующие значения:

- `visible` — если содержимое и выйдет за пределы отведенного ему места, он все равно будет показан полностью; это значение принято по умолчанию;
- `hidden` — выступающие за границы элемента-контейнера части содержимого будут обрезаны;
- `auto` — добавляет полосы прокрутки, если содержимое выходит за границы элемента-контейнера;
- `scroll` — добавляет полосы прокрутки в любом случае.

Параметр `clip` позволяет показать некоторую прямоугольную часть содержимого элемента. Он принимает значения `auto` (по умолчанию) и `rect(top right bottom left)`. Функция `rect` принимает в качестве параметров верхнюю, правую, нижнюю и левую координаты, задающие прямоугольную область содержимого элемента, которую требуется показать. При этом координаты указываются через пробел.

Элементы, по умолчанию видимые в окне браузера, можно сделать невидимыми с помощью параметров `visibility` и `display`. Обычно они используются в сценариях для динамического управления видимостью элементов. Например, раскрывающееся меню можно сделать на основе установки с некоторой временной задержкой видимости его опций. Иногда в документе требуется использовать элемент, но не показывать его. Например, можно загрузить в браузер таблицу с данными, которые будут как-то использоваться сценарием, но пользователь не должен их видеть.

Параметр `visibility` принимает следующие значения:

- `visible` — элемент виден;
- `hidden` — элемент не виден (скрыт), однако занимает на странице место, т. е. элемент прозрачен;
- `inherit` — элемент виден, если его родительский элемент отображается.

Если элемент имеет параметр `visibility:hidden`, то он не виден, однако занимает место на странице.

Параметр `display` может принимать значение `none` (элемент не виден) или пустое значение (пустую строку), при котором элемент виден. В отличие от параметра `visibility:hidden`, параметр `display:none` делает элемент не только невидимым, но и не занимающим место. Иначе говоря, при `display:none` элемент исключается из потока отображения, и освободившееся место может быть занято другими элементами.

1.3.10. Позиционирование

Каскадные таблицы стилей позволяют произвольным образом позиционировать элементы HTML-документа. При этом можно нарушать расположение, связанное с естественным порядком упоминания элементов в HTML-коде и даже накладывать одни элементы на другие.

Позиционирование элементов обеспечивается с помощью следующих трех параметров:

- `left` — горизонтальная координата элемента (*x*-координата);
- `top` — вертикальная координата элемента (*y*-координата);
- `z-index` — третья координата для указания порядка, в котором элементы будут перекрывать друг друга; принимает в качестве значения целые числа. Элементы с большим *z*-индексом будут появляться над элементами с меньшим *z*-индексом. Таким образом, возникает возможность создавать слои документа.

Параметры `left` и `top` принимают в качестве значений число с указанием единиц измерения или ключевое слово `auto`. Если единицы измерения не указаны, то предполагаются пикселы (`px`).

Чтобы позиционировать элемент в какую-либо точку с координатами `left`, `top` и `z-index`, необходимо использовать параметр `position`, который задает способ позиционирования.

Параметр `position` может принимать следующие значения.

- `absolute` — абсолютное позиционирование. Элемент устанавливается относительно верхнего левого угла контейнера (объекта, содержащего данный элемент) в соответствии с заданными координатами `left` и `top`. Если элемент находится вне контейнера (точнее, внутри контейнера `<html>`) или в контейнере, заданном тегом `<body>`, то началом отсчета координат будет верхний левый угол страницы. Заметим, что положение элемента не зависит от положения его тега внутри HTML-документа.
- `relative` — относительное позиционирование. Элемент будет установлен в соответствии с тем, в каком месте исходного HTML-кода он находится,

т. е. в соответствии с естественным порядком. При этом следует помнить, что абсолютно позиционированные элементы изымаются из потока отображения по естественному порядку. Иначе говоря, элемент будет позиционироваться относительно предыдущего элемента, который не был позиционирован абсолютно. Значение `relative` установлено по умолчанию. Например, если элемент находится в трех строках от начала его выделения в тексте документа, то по умолчанию считается, что параметр позиционирования имеет значение `relative`, а свойства координат `left` и `top` — нулевые значения. Ненулевые значения свойств `left` и `top` сдвигают элемент относительно исходного положения. Однако чтобы указанные значения параметров `left` и `top` имели силу, необходимо указать `position:absolute` в явном виде.

- `static` — статическое позиционирование. Аналогично относительному (`relative`), за исключением того, что при данном способе позиционирования элемент нельзя сместить с помощью параметров `left` и `top`.

1.3.11. Графические фильтры

Интересные визуальные эффекты, связанные с графикой и текстом, можно получить с помощью *фильтров каскадных таблиц стилей*. Например, постепенное появление (исчезновение) рисунка, плавное преобразование одного изображения в другое, задание степени прозрачности и т. п.

Фильтр следует понимать как некий инструмент преобразования изображения, взятого из графического файла и вставленного в HTML-документ. Кроме того, некоторые фильтры можно применять и к текстам. Однако следует иметь в виду, что фильтры работают только в Internet Explorer, начиная с версии 4. Это, конечно, ограничивает их использование при создании публикаций, инвариантных относительно браузеров.

Существуют два вида фильтров: *статические* и *динамические*. Все они определяются в каскадной таблице стилей одинаково:

```
filter: имя_фильтра(параметр1, параметр2, ...)
```

Статические фильтры сначала преобразуют изображение или элемент с текстом, а затем результат преобразования отображается в браузере. Таким образом, пользователь видит только конечный результат преобразования.

Динамические фильтры выполняют преобразование в течение некоторого времени, которое можно указать в качестве параметра. При этом создается анимационный эффект: преобразование происходит постепенно с отображением всех его промежуточных этапов. В отличие от статических фильтров, применение динамических фильтров обязательно связано с использованием

несложных сценариев. В свою очередь, сценарии можно применять для изменения свойств как статических, так и динамических фильтров, создавая различные анимационные эффекты.

Фильтры применяются не ко всем HTML-элементам, а только к тем из них, которые определяют прямоугольный блок и не являются окнами. Ниже приводится список тегов, к содержимому которых можно применять фильтры:

- `<body>`;
- `<button>`;
- `<div>` — если заданы параметры размеров `width` и `height` или элемент абсолютно позиционирован;
- `<img>`;
- `<input>`;
- `<marquee>`;
- `<span>` — если заданы параметры размеров `width` и `height` или элемент абсолютно позиционирован;
- `<table>`;
- `<textarea>`;
- `<td>`;
- `<th>`;
- `<textarea>`.

Статические фильтры

К статическим фильтрам относятся следующие:

- `alpha` — определяет степень видимости объекта. Принимает параметры: `opacity`, `style`, `finishOpacity`, `startX`, `startY`, `finishX`, `finishY`, `enabled`;
- `blur` — создает эффект движения объекта. Принимает параметры: `add`, `direction`, `strength`, `enabled`;
- `chroma` — переводит определенный цвет изображения в прозрачный. Принимает параметры: `color`, `enabled`;
- `dropShadow` — отображает силуэт определенного цвета для выбранного объекта, создавая иллюзию, что объект находится над страницей, отбрасывая тень. Принимает параметры: `offX` , `offY`, `positive`, `enabled`;
- `fliph` — отражает объект относительно центральной вертикальной оси. Принимает параметр: `enabled`;

- `flipv` — отражает объект относительно центральной горизонтальной оси. Принимает параметр: `enabled`;
- `glow` — создает для объекта эффект сияния. Принимает параметры: `color`, `strength`, `enabled`;
- `gray` — отображает объект в градациях серого. Принимает параметр: `enabled`;
- `invert` — обращает цвета объекта. Принимает параметр: `enabled`;
- `mask` — делает из объекта прозрачную маску (отображает прозрачные (transparent) символы определенным цветом, а непрозрачные делает прозрачными). Принимает параметр: `color`, `enabled`;
- `shadow` — отрисовывает "боковые" грани объекта, создавая впечатление объемности. Принимает параметры: `color`, `direction`, `enabled`;
- `wave` — создает волнистое искажение объекта вдоль вертикальной оси. Принимает параметры: `add`, `strength`, `freq`, `phase`, `ligthStrength`, `enabled`;
- `xray` — изменяет глубину цвета объекта и отрисовывает черно-белым, делая его похожим на рентгеновский снимок. Принимает параметр: `enabled`.

В данном разделе будут рассмотрены лишь некоторые из фильтров.

Следующий пример иллюстрирует применение некоторых из перечисленных выше фильтров с параметрами, принятыми по умолчанию (листинг 1.23).

Листинг 1.23. Применение фильтров

```
<html>
  <head><title>Фильтры</title>
    <style>
      p {position:absolute; top:0; left:190; font-size:80}
    </style>
  </head>
  <img style="position:absolute" src="picture.jpg">
  <img style="position:absolute;top:140;left:10; filter:flipv"
       src="picture.jpg">
    <p style="top:140; filter:flipv">Фильтр flipv</p>
</html>
```

Каждый фильтр имеет параметр `enabled`, принимающий значения `true` (разрешено применение) и `false` (запрещено применение). Значением по умолчанию является `true`.

Фильтры `blur`, `glow` и `wave` имеют параметр `strength`, определяющий интенсивность (силу) применения фильтра в диапазоне целых чисел от 0 до 255.

Фильтры `chroma`, `dropShadow`, `glow`, `mask` и `shadow` имеют параметр `color`, принимающий в качестве значения имя или код цвета.

Фильтры `blur` и `shadow` имеют параметр `direction`, определяющий направление размытия изображения и падения тени соответственно. Этот параметр принимает значения 0 и кратные 45, означающие количество градусов, отсчитанное от вертикали по часовой стрелке.

Фильтр `blur` имеет параметр `add`, принимающий значения `true` и `false`. Этот параметр определяет, добавлять (`true`) или нет (`false`) исходное изображение к результату применения фильтра.

Чтобы фильтры `blur`, `glow`, `wave`, `alpha`, `dropShadow`, `shadow`, `xray` воздействовали на тексты, последние должны быть абсолютно позиционированными, т. е. иметь параметр `position:absolute`.

Рассмотрим более подробно некоторые из наиболее популярных фильтров.

Тень *(dropShadow, shadow)*

Фильтр `dropShadow` имеет параметры `offX` и `offY`, определяющие смещение тени в пикселах в горизонтальном и вертикальном направлениях соответственно. Параметр `positive` определяет, от каких пикселов следует отбрасывать тень: от непрозрачных (`true`) или прозрачных (`false`). Этот фильтр действует на графические изображения и дает видимые результаты, если они содержат прозрачные участки. Графические изображения с прозрачными областями можно сохранять, например, в gif-файлах. В следующем примере изображение представляет собой текст на прозрачном фоне, сохраненный в gif-файле. С помощью фильтра `dropShadow` к этому изображению был добавлен эффект тени (листинг 1.24).

Листинг 1.24. Добавление тени

```
<html>
  <head><title>Фильтр dropShadow</title>
    <style>
      #myimg {filter: dropShadow(offX=4, offY=4, color:#a0a0a0)}
    </style>
  </head>
  <img id="myimg" src="picture.gif">
</html>
```

Фильтр `dropShadow` можно применять и к абсолютно позиционированным текстам.

Фильтр `shadow` имеет параметры `direction` (направление) и `color` (цвет). Параметр `direction` принимает значения 0 и кратные 45, означающие количество градусов, отсчитанное от вертикали по часовой стрелке.

Волновое искажение *(wave)*

Фильтр `wave` имеет параметры: `add`, `strength`, `freq`, `phase`, `ligthStrength`. Параметр `add` определяет добавлять (`true`) или нет (`false`) исходное изображение к результату применения фильтра. Параметр `strength` задает амплитуду волны искажения. Параметр `freq` указывает количество максимумов волны искажения, а `phase` — фазу в диапазоне от 0 до 100. Параметр `ligthStrength` определяет степень освещения для имитации трехмерного эффекта для волны. Этот фильтр можно применять и к изображениям, и к абсолютно позиционированным текстам.

Размытие *(blur)*

Фильтр `blur` имеет параметры: `add`, `strength` и `direction`. Параметр `add` определяет, добавлять (`true`) или нет (`false`) исходное изображение к результату применения фильтра. Параметр `strength` задает силу размытия. Параметр `direction` указывает направление размытия изображения. Он принимает значения 0 и кратные 45, означающие количество градусов, отсчитанных от вертикали по часовой стрелке. Фильтр `blur` можно применять и к изображениям, и к абсолютно позиционированным текстам.

Прозрачность *(alpha)*

Прозрачность графического объекта в диапазоне целочисленных значений от 0 до 100 можно установить с помощью фильтра `alpha`. Значение 0 соответствует полной прозрачности изображения, т. е. оно становится невидимым. Значение 100 означает, наоборот, полную непрозрачность изображения. Поэтому правильнее говорить, что `alpha` является степенью непрозрачности. Сквозь прозрачные графические объекты видны лежащие ниже изображения. Степень их видимости определяется значением прозрачности изображения, лежащего выше. Кроме того, прозрачность имеет несколько вариантов градиентной формы. Например, можно сделать так, чтобы прозрачность постепенно увеличивалась от центра к краям изображения. Это позволяет просто и весьма эффективно согласовать вставляемое в документ изображение с фоном. Web-дизайнеры хорошо знают, что выполнение такой операции с помощью графического редактора требует определенных навыков и усилий.

Фильтр `alpha` имеет параметры:

- `opacity` — задает степень прозрачности (точнее, непрозрачности) как целое число в диапазоне от 0 до 100;
- `style` — определяет градиентную форму распределения прозрачности по изображению как целое число от 0 до 3. Если параметр `style` не указан или имеет значение 0, то градиент не применяется (степень прозрачности одинакова для любых участков изображения);
- `finishOpacity` — конечное значение степени прозрачности (от 0 до 100);
- `startX`, `startY`, `finishX`, `finishY` — координаты точек, определяющие прямую, на которой прозрачность равна значению параметра `finishOpacity`. При `style` > 0 прозрачность точек между границами объекта и этой прямой изменяется от значения `opacity` до `finishOpacity`. При `style` = 2 прозрачность изменяется от центра к эллипсу, вписанному в прямоугольник с диагональю, заданной координатами `startX`, `startY`, `finishX`, `finishY`. При этом вне эллипса прозрачность определяется значением `finishOpacity`.

  Пример:

```
<img src="mypicture.gif"
     style="filter:alpha(opacity=70, style=3)">
```

Выше мы рассмотрели синтаксис выражений, воспринимаемый браузером Internet Explorer, начиная с версии 4 . Для версии 5.5 и более поздних можно использовать другой синтаксис, рекомендованный Microsoft. Его особенность в том, что в каскадной таблице стилей задается ссылка на специальный компонент и имя фильтра:

```
# id_изображения {filter:
progid:DXImageTransform.Microsoft.имя_фильтра(параметры)}
```

Например:

```
# myimg {filter: progid:DXImageTransform.Microsoft.alpha(opacity=70,
style=3)}
```

Повороты

В браузере Internet Explorer 5.5 и более поздних версий возможно применение фильтра `basicimage`, имеющего множество параметров, с помощью которых графическое изображение можно повернуть на угол, кратный 90°, задать прозрачность, зеркально отразить, определить маску и др. В Internet Explorer, начиная с версии 4, все эти эффекты, за исключением поворотов, создаются отдельными фильтрами. Порядок их применения такой же, как и фильтра `alpha`, задающего прозрачность. Здесь мы рассмотрим создание эффекта поворота.

Параметр `rotation` фильтра `basicimage` принимает целочисленные значения: 0 (нет поворота), 1 (90°), 2 (180°), 3 (270°). В следующем примере графическое изображение поворачивается на 90° по часовой стрелке (листинг 1.25).

Листинг 1.25. Поворот изображения на 90° по часовой стрелке

```
<html>
  <head><title>Поворот</title>
    <style>
      #myimg {filter:
              progid:DXImageTransform.Microsoft.basicimage(rotation=1)}
    </style>
  </head>
  <img id="myimg" src="myimg.jpg">
</html>
```

Хотя применение статических фильтров не требует использования сценариев, наиболее интересные визуальные эффекты получаются за счет изменения параметров фильтра по событиям и/или с течением времени. А это можно сделать только с помощью сценариев.

Динамические фильтры

Динамические фильтры позволяют организовать постепенное появление или исчезновение изображения, трансформацию одного графического объекта в другой, а также имитирование освещения. Как уже отмечалось выше, для их применения необходимо использовать сценарии. Язык JavaScript и программирование сценариев подробно будут рассмотрены в следующих главах, однако простейшие сценарии, связанные с применением динамических фильтров, мы рассмотрим уже сейчас.

Трансформация

Суть трансформации графического объекта заключается в том, что сначала необходимо зафиксировать первое изображение, затем произвести замену этого изображения другим и/или изменить параметры того же самого изображения, а после этого осуществить собственно трансформацию. Все эти действия выполняются в сценарии. Фиксация и трансформация изображения производятся с помощью специальных методов (функций) фильтра — `apply()` и `play()` соответственно. Для остановки процесса преобразования предназначен метод `stop()`. Для трансформации объектов служат два фильт-

ра: `revealtrans` и `blendtrans`. По существу, второй из них является частным случаем первого.

Фильтр `revealtrans` используется для трансформации изображения — постепенного появления или перехода от одного изображения к другому. Кроме параметра `enabled` (которым обладают все фильтры), он имеет следующие параметры:

- `duration` — длительность преобразования в секундах (число с плавающей точкой);
- `transition` — тип преобразования (целое число от 0 до 23):
  - 0 — Box In (стягивающийся прямоугольник);
  - 1 — Box Out (расширяющийся прямоугольник);
  - 2 — Circle In (стягивающийся круг);
  - 3 — Circle Out (расширяющийся круг);
  - 4 — Wipe Up (стирание вверх);
  - 5 — Wipe Down (стирание вниз);
  - 6 — Wipe Right (стирание вправо);
  - 7 — Wipe Left (стирание влево);
  - 8 — Vertical Blinds (вертикальные жалюзи);
  - 9 — Horisontal Blinds (горизонтальные жалюзи);
  - 10 — Checkerboard Across (сужающиеся клетки шахматной доски);
  - 11 — Checkerboard Down (закрывающаяся шахматная доска);
  - 12 — Random Dissolve (случайный наплыв);
  - 13 — Split Vertical In (вертикальное деление внутрь);
  - 14 — Split Vertical Out (вертикальное деление наружу);
  - 15 — Split Horisontal In (горизонтальное деление внутрь);
  - 16 — Split Horisontal Out (горизонтальное деление наружу);
  - 17 — Strips Left Down (стирание влево вниз);
  - 18 — Strips Left Up (стирание влево вверх);
  - 19 — Strips Right Down (стирание вправо вниз);
  - 20 — Strips Right Up (стирание вправо вверх);
  - 21 — Random Bars Horisontal (случайные горизонтальные полосы);
  - 22 — Random Bars Vertical (случайные вертикальные полосы);
  - 23 — Random selection of (0—22) (случайный выбор из предыдущих вариантов).

Сначала рассмотрим применение фильтра `revealtrans` для создания эффекта появления изображения. В листинге 1.26 приведен пример, в котором изображение постепенно появляется.

Листинг 1.26. Постепенное появление изображения

```
<html>
  <head><title>Фильтр revealtrans</title>
  </head>
  <img id=myimg  src="mypicture.jpg"
       style="visibility=hidden;
              filter:revealtrans(duration=2, transition =12)">
  <script>
    // появление изображения
    myimg.filters("revealtrans").apply() /* фиксируем исходное */
                                            состояние изображения */
    myimg.style.visibility="visible"     /* делаем изображение видимым */
    myimg.filters("revealtrans").play()  /* выполняем преобразование */
  </script>
</html>
```

Здесь в таблице стилей для изображения устанавливается его невидимость (`visibility=hidden`) и фильтр с указанием его имени и параметров. В секции сценария с помощью метода `applay()` сначала фиксируется исходное невидимое состояние изображения, затем изображение делается видимым и, наконец, с помощью метода `play()` выполняется переход от невидимого состояния к видимому.

Доступ к методам фильтра `apply()`, `play()` и `stop()` осуществляется через объект `filters` с указанием идентификатора (значения атрибута `id`, в примере — `myimg`) соответствующего элемента. В приведенном выше коде это делается с помощью выражения:

```
myimg.filters("revealtrans").apply()
```

При этом допустима и такая запись:

```
myimg.filters.revealtrans.apply()
```

Следующий HTML-код (листинг 1.27) выполняет преобразование изображения после загрузки всех элементов документа (т. е. по событию `onload` для объекта `window`, который соответствует окну браузера). В отличие от предыдущего примера, исходное состояние изображения задается не в таблице стилей, а в сценарии.

Листинг 1.27. Преобразование изображения после загрузки всех элементов документа

```
<html>
  <head><title>Фильтр revealtrans</title>
  </head>
  <img id=myimg  src="mypicture.jpg"
       style="filter:revealtrans(duration=2, transition =12)">
  <script>
    function window.onload(){       // появление изображения
      document.all.myimg.style.visibility="hidden" /* делаем
                                                     изображение невидимым */
      myimg.filters("revealtrans").apply()  /* фиксируем исходное
                                                    состояние изображения */
      myimg.style.visibility="visible"   /* делаем изображение видимым */
      myimg.filters("revealtrans").play() /* выполняем преобразование */
    }
  </script>
</html>
```

Очевидно, чтобы добиться исчезновения изображения, необходимо поступить наоборот: сначала сделать его видимым, а затем невидимым.

Теперь рассмотрим более общий случай: трансформацию одного графического объекта в другой. В отличие от рассмотренного выше случая, эта задача сводится к установке различных начального и конечного изображений, а не различных состояний одного и того же изображения. Это делается путем присвоения нужных значений свойству `src` объекта, значением которого является URL-адрес файла с изображением, как показано в следующем примере (листинг 1.28).

Листинг 1.28. Трансформация одного графического объекта в другой

```
<html>
  <head><title>Фильтр revealtrans</title>
  </head>
  <img id=myimg  src="mypicture1.jpg"
       style="filter:revealtrans(duration=2, transition =12)">
  <script>
    function window.onload(){     // появление изображения
```

```
    myimg.filters("revealtrans").apply()/* фиксируем исходное
                                           изображение */
    myimg.src="mypicture2.jpg"          /* заменяем изображение */
    myimg.filters("revealtrans").play() /* выполняем преобразование */
  }
  </script>
</html>
```

Переход между различными изображениями или различными состояниями одного и того же изображения можно выполнять и с помощью фильтра `blendtrans`. Этот фильтр принимает единственный параметр `duration`, определяющий длительность преобразования в секундах, и обладает такими же методами. В отличие от `revealtrans`, он осуществляет преобразование единственным способом: одно изображение постепенно исчезает, а другое — появляется. Следующий HTML-код иллюстрирует применение фильтра `blendtrans` (листинг 1.29).

Листинг 1.29. Переход между различными изображениями

```
<html>
  <head><title>Фильтр blendtrans</title>
  </head>
  <img id=myimg  src="mypicture1.jpg"
       style="filter: blendtrans(duration=2)">
  <script>
    function window.onload(){    // появление изображения
      myimg.filters("blendtrans").apply() /* фиксируем исходное
                                             изображение */
      myimg.src="mypicture2.jpg"          /* заменяем изображение */
      myimg.filters("blendtrans").play()  /* выполняем преобразование */
    }
  </script>
</html>
```

Освещение

Создать эффект освещения графического объекта источником света можно с помощью фильтра `light`. Этот фильтр имеет единственный параметр `enabled`, принимающий значения `true` (разрешено применение) и `false` (запрещено применение). Значением по умолчанию является `true`. Управление фильтром осуществляется из сценариев с помощью нескольких методов,

применение которых выполняется из сценария. Методы фильтра задают тип источника освещения и параметры собственно освещения.

Методы фильтра `light` перечислены ниже.

- `addAmbient(r, g, b, s)` — добавляет источник рассеянного (ненаправленного) света, параллельные лучи которого направлены перпендикулярно плоскости страницы. Первые три параметра метода задают яркости соответственно красной, зеленой и синей составляющих света. Последний параметр определяет интенсивность источника света. Значения параметров — целые числа в диапазоне от 0 до 255.
- `addCone(x1, y1, z, x2, y2, r, g, b, s, a)` — добавляет источник направленного света. Принимает следующие параметры:
  - `x1`, `y1` — координаты источника света в пикселах;
  - `z` — номер слоя (`z-index`) источника света;
  - `x2`, `y2` — координаты точки, в которую направлен свет;
  - `r`, `g`, `b` — яркости соответственно красной, зеленой и синей составляющих света (0—255);
  - `s` — интенсивность источника света (0—255);
  - `a` — угол конуса света (0—360). Прямая, проходящая через точки источника и падения света, является биссектрисой этого угла.
- `addPoint(x1, y1, z, r, g, b, s)` — добавляет точечный источник света. Принимает параметры:
  - `x1`, `y1` — координаты источника света в пикселах;
  - `z` — номер слоя (`z-index`) источника света;
  - `r`, `g`, `b` — яркости соответственно красной, зеленой и синей составляющих света (0—255);
  - `s` — интенсивность источника света (0—255).

HTML-код из листинга 1.30 иллюстрирует применение фильтра с направленным источником света.

Листинг 1.30. Применение фильтра с направленным источником света

```
<html>
  <head>
    <title>Фильтр light</title>
  </head>
  <img id=myimg src="myimg.jpg"
```

```
      style="position:absolute;top:0;left:0;width:200;height:180;
             filter:light;">
  <script>
    function window.onload() {
      myimg.filters.light.addCone(10,10,0,100,100,255,255,255,255,25)
    }
  </script>
</html>
```

При использовании только направленного источника света часть изображения, находящаяся вне светового конуса, не видна. Чтобы сделать ее видимой, необходимо добавить источник рассеянного света.

1.4. Что и как делают клиентские сценарии?

Программы, работающие с объектами HTML-документа и браузера, выполняются браузером и называются *клиентскими сценариями*. Браузеры, вообще способные воспринимать и выполнять сценарии, имеют встроенный интерпретатор языка JavaScript, а Internet Explorer, кроме того, имеет и интерпретатор языка VBScript, близкого к Visual Basic.

Возможны несколько вариантов размещения программ на JavaScript, играющих роль сценариев для HTML-документов. Сценарии можно писать непосредственно в HTML-документе, а также в отдельных текстовых файлах, которые вызываются из HTML-документа. Проще всего размещать сценарий непосредственно в HTML-документе, чаще всего так и поступают. Размещение сценария в отдельном файле совсем не сложно, но не всегда оправдано из экономических соображений. Необходимо накопить достаточное количество программных заготовок, чтобы решиться использовать их в последующих проектах в качестве библиотек.

Стандартным является размещение сценария в HTML-документе в контейнерном теге `<SCRIPT>`, т. е. между тегами `<SCRIPT>` и `</SCRIPT>`. Встречая тег `<SCRIPT>`, браузер "понимает", что за ним начинается код сценария. Заключительный тег `</SCRIPT>` указывает браузеру, что код сценария закончился. Все, что находится вне этих тегов, браузер воспринимает как HTML-код. Контейнер `<SCRIPT>` может располагаться в любом месте HTML-документа, и даже не один раз. От его расположения иногда может зависеть функционирование всего HTML-документа. Описание тега `<SCRIPT>` приведено в *разд. 1.2.2*.

Старые браузеры, появившиеся раньше JavaScript, игнорируют теги `<SCRIPT>` и `</SCRIPT>`, а все, что находится между ними, интерпретируют как текстовое содержимое HTML-документа. Результат может быть самым неожиданным. Чтобы уменьшить вероятность отображения кода сценария в окне старого браузера, следует заключить его в дескрипторы комментария `<!--` и `-->`. Новые браузеры, поддерживающие сценарии, будут игнорировать эти символы, выполняя код сценария, а старые браузеры (не понимающие сценарии), наоборот, станут игнорировать код сценария. Вот как используются символы комментария предохранения сценария от старых браузеров:

```
<SCRIPT LANGUAGE="JavaScript" >
  <!--
    // код сценария
  //-->
</SCRIPT>
```

Обратите внимание на заключительные символы тега комментария, перед которыми стоят две косые черты. Без них JavaScript будет пытаться интерпретировать символы `-->` как заключительные символы комментария HTML, а с ними он просто их проигнорирует.

Внимание

В примерах, приводимых в настоящей книге, мы не будем указывать символы комментария для адаптации к старым браузерам из соображений экономии места. Однако помните, что современная культура оформления сценариев обязывает нас делать это в своих практических приложениях.

Если сценарий располагается в отдельном файле, то в нем, разумеется, теги `<SCRIPT>` и `</SCRIPT>` не пишутся. Как уже отмечалось, файлы со сценариями на JavaScript являются обычными текстовыми файлами. В принципе, они могут иметь любое расширение имени, но, как правило, используется расширение js. В отдельных файлах обычно размещают библиотеки функций (определения функций), а также сценарии, использующиеся в нескольких HTML-документах одного или нескольких сайтов. Сценарий, загруженный из внешнего файла, можно представить себе просто как его вставку в HTML-документ. Вот пример:

```
<HTML>
...
  <SCRIPT>
    function myfunc(){
      ...
    }
```

```
  </SCRIPT>
  <SCRIPT SRC="mylib.js"></SCRIPT>
  <SCRIPT SRC="myprogram.js"></SCRIPT>
...
</HTML>
```

Здесь в HTML-документе расположены три раздела (секции) сценария, два из которых загружаются из отдельных файлов. В первом разделе сценария дано определение некоторой функции.

Сценарий можно также писать как строку операторов, разделенных точкой с запятой, взятую в кавычки. В такой форме сценарии обычно используются в качестве обработчиков событий. Об этом более подробно будет рассказано в *разд. 3.2*.

Вызовы функций и их определения могут располагаться в произвольном порядке, только если они заключены в один и тот же контейнер `<SCRIPT>`. Если вы располагаете их в разных контейнерах `<SCRIPT>`, то необходимо, чтобы определение функции находилось выше ее вызова. Аналогично, если определения функций находятся в отдельном файле, то его необходимо загрузить в HTML-документ раньше вызовов этих функций. Далее приведены примеры правильного и неправильного расположения сценариев в HTML-документе.

Правильно

```
<HTML>
  ...
  <SCRIPT>
    function myfunc(){
      ...
    }
  </SCRIPT>
  ...
  <SCRIPT>
    ...
    myfunc()
    ...
  </SCRIPT>
  ...
</HTML>
```

Неправильно

```
<HTML>
  ...
  <SCRIPT>
    ...
    myfunc()
    ...
  </SCRIPT>
  ...
  <SCRIPT>
    function myfunc(){
      ...
    }
  </SCRIPT>
  ...
</HTML>
```

При попытке загрузить и выполнить неправильный вариант HTML-кода появится диалоговое окно с сообщением об ошибке "Ошибка: Предполагается

наличие объекта". Браузер интерпретирует теги HTML последовательно. Так, встретив выражение вызова функции `myfunc()` в одном контейнере `<SCRIPT>`, браузер еще не имеет в памяти определения этого объекта (функции), расположенного в другом контейнере `<SCRIPT>`. Если же определение функции и ее вызов находятся в одном и том же контейнере `<SCRIPT>`, то его содержимое сначала загружается в память, а затем анализируется. Если интерпретатор встречает вызов функции, то он ищет ее определение в памяти и при удачном результате выполняет код этой функции. Следующие два варианта являются правильными, потому что определение функции и ее вызов находятся в одной секции сценария (в одном контейнере `<SCRIPT>`):

```
<HTML>
  ...
  <SCRIPT>
    ...
    function myfunc(){
      ...
    }
    ...
    myfunc()
    ...
  </SCRIPT>
  ...
</HTML>
```

```
<HTML>
  ...
  <SCRIPT>
    ...
    myfunc()
    ...
    function myfunc(){
      ...
    }
    ...
  </SCRIPT>
  ...
</HTML>
```

Если необходимо, чтобы сценарий загрузился в браузер прежде, чем загрузятся элементы HTML-документа, то его следует расположить в верхней части HTML-кода. В этом случае сценарий обычно располагают в контейнере `<HEAD>` (в заголовке документа). Это самое лучшее место для расположения функций.

Если требуется, чтобы сценарий загрузился после загрузки всех элементов HTML-документа, то возможны следующие два варианта:

- расположить сценарий в конце HTML-документа;
- использовать атрибут-событие `onload` в контейнерном теге `<BODY>`, который задает основную часть HTML-документа. В последнем случае значением атрибута `onload` обычно является строка, содержащая вызов функции-обработчика события. Определение этой функции, как правило, содержится в контейнере `<SCRIPT>`, размещенном в контейнере заголовка HTML-документа `<HEAD>`. Обратите внимание, что при использо-

вании атрибута-события `onload` в теге `<BODY>` обработчик этого события выполняется по завершении загрузки всех элементов, определенных в контейнере `<BODY>`, а не в процессе их загрузки. Вот пример:

```
<HTML>
  <HEAD>
    <SCRIPT>
      function myfunc(){
        ...
      }
    </SCRIPT>
  </HEAD>
  ...
  <BODY onload="myfunc()">
    <! Теги тела документа >
  </BODY>
  ...
</HTML>
```

С помощью клиентских сценариев можно решить множество задач. Вот далеко не полный список наиболее типичных из них:

- обработка событий, инициированных пользователем (щелчок кнопкой мыши, нажатие клавиши и др.);
- обработка данных, введенных пользователем в форму, перед их отправкой на сервер;
- изменение параметров отдельных элементов документа и браузера (изменение значений атрибутов, стилевых параметров и даже тегов);
- запись и чтение cookie;
- управление процессами во времени.

1.5. Что и как делают серверные сценарии?

В HTML-документ кроме клиентских сценариев в обрамлении тегов `<script>` и `</script>` можно вставить сценарии, выполняющиеся не браузером, а сервером. Сценарий на языке PHP в HTML-документе обрамляется дескрипторами `<?php` и `?>`.

HTML-документы со вставками сценариев на языке PHP сохраняются в текстовых файлах с расширением php, например, mypage.php. Файлы с расширением php должны быть расположены в одной из папок Web-сервера.

При вызове из браузера php-файла последний сначала анализируется на сервере. Все обычные тексты и теги HTML анализатор пропускает, оставляя их без изменения. Если же он встречает фрагмент, заключенный в дескрипторы `<?php` и `?>`, то считает, что это код сценария, написанного на языке PHP. В этом случае к работе подключается интерпретатор PHP, который и выполняет сценарий. При этом сам код сценария из HTML-документа исключается. Так происходит со всеми вставками PHP-кода. Оставшийся в результате обработки HTML-документ пересылается запросившему его браузеру. Обратите внимание, что полученный браузером документ не содержит кодов серверных сценариев.

Сценарий на языке PHP, помимо прочего, посредством специальных операторов и функций может выводить в HTML-документ текстовые строки, как обычные, так и содержащие HTML-код. В этом случае происходит дополнение исходного HTML-документа фрагментами, сгенерированными серверными сценариями. Более того, исходный HTML-документ может и не содержать никакого HTML-кода. Вместо него может находиться только PHP-код, заключенный в дескрипторы `<?php` и `?>`. Что получит в этом случае браузер (и получит ли вообще что-нибудь), зависит от сценария. Сценарий может отработать "молча" (например, записать в файл или базу данных какие-то сведения), ничего не направляя в выходной документ, а может сгенерировать целую Web-страницу.

Итак, php-файл может содержать текст и/или HTML-код, и/или PHP-код. Как уже отмечалось, он должен быть расположен на Web-сервере, а не просто где-либо на жестком диске компьютера. Чтобы открыть php-файл в браузере, в адресную строку следует ввести URL-адрес. Так, если требуется отладить PHP-сценарий на локальном компьютере, то для этого необходимо иметь установленный на нем Web-сервер. Тогда для открытия в браузере файла, например, mypage.php, расположенного в домашнем каталоге Web-сервера, в адресную строку следует ввести **http://localhost/mypage.php**.

Многие задачи, которые трудно решать с помощью клиентских сценариев, довольно просто выполняются серверными сценариями. Вот далеко не полный список типичных задач, решаемых с помощью серверных сценариев:

- прием и обработка данных, введенных посетителями Web-страницы;
- сохранение данных в файлах и базах данных, расположенных на стороне сервера;
- передача данных по электронной почте;
- модификация графических изображений.

Защита страниц сайта паролем, счетчик посещений страниц, гостевая книга и форум, интернет-магазины и многое другое поддерживаются серверными сценариями.

Глава 2

Основы JavaScript

В данной главе мы рассмотрим важнейшие понятия и синтаксис языка JavaScript. Просматривать, создавать и редактировать скрипты (сценарии, программы) можно в специальных редакторах Web-приложений, таких как Macromedia Dreamweaver и Microsoft FrontPage. Однако здесь мы не рассматриваем эти средства. В качестве редактора программ выберите какой-нибудь простой текстовый редактор, например Блокнот Windows.

Чтобы программировать настоящие сценарии для Web-сайтов, необходимо знать среду — объектную модель браузера и загруженного в него документа. Впрочем, это важно для программирования и на любых других языках. Синтаксис современных языков программирования не сложен и может быть легко и быстро усвоен. Однако практическая польза программирования материализуется при синтезе знаний синтаксиса языка и уже готовой среды программирования — объектной модели, библиотек функций и классов. Сведения об основных объектах модели HTML-документа и браузера вы найдете в *приложении 1*. О способах их использования модели вы узнаете в *главе 3*. Сейчас же мы сконцентрируем внимание собственно на языке JavaScript — средстве доступа к упомянутым объектам. Этот язык очень прост, значительно проще, чем мир доступных для него объектов. Однако умение программировать на нем требует некоторых усилий. Приложите эти усилия, чтобы убедиться, что не боги горшки обжигают.

2.1. Подготовка к программированию

Чтобы создавать *клиентские сценарии* (т. е. сценарии, исполняемые Web-браузером, способным в принципе это делать), в вашей системе ничего не нужно устанавливать предварительно, коль скоро у вас уже установлен современный браузер. Однако следует иметь в виду, что результат выполнения

сценариев ощутимо зависит от конкретного типа браузера и его версии. Я, по мере своих возможностей, пытался в этой книге вас предупреждать о потенциальных неприятностях. Как бы то ни было, ручаюсь (если где-то не опечатался), что все должно работать в IE5.5+. Важные вопросы, связанные с кроссплатформенностью и инвариантностью Web-публикаций относительно браузеров и т. п., я отношу к теме разработки и дизайна (development & design) сайтов. Однако это другая тема, не рассматриваемая в данной книге. Здесь я рассматриваю лишь некоторые основные возможности языка JavaScript, применимые во всех или только в некоторых браузерах. Основные проблемы корректного воспроизведения некоторых примеров в браузерах Netscape Navigator, Opera и других обозревателях связаны, в первую очередь, с графическими фильтрами каскадных таблиц стилей и, во вторую очередь, с элементами ActiveX, которые естественны для Internet Explorer, но могут не найти полного соответствия среди плагинов (plug-ins) в других браузерах. В IE4+ если не все примеры, то многие из них работают. В IE5.5+ все примеры должны правильно воспроизводиться. Далее, многим элементам управления ActiveX, которые встроены в IE5+, соответствуют плагины для других браузеров. Некоторые из них либо поставляются вместе с дистрибутивом браузера, либо могут быть добавлены к браузеру через Интернет. В этой книге мы начинаем изучать язык JavaScript, используя интерпретатор, встроенный в браузер Internet Explorer.

Откройте текстовый редактор (например, Блокнот Windows) и создайте в нем заготовку файла, который вы будете потом редактировать, вводя экспериментальные выражения или даже целые программы. А именно введите в рабочее поле редактора следующий текст:

```
<HTML>
  <HEAD><TITLE>Примеры</TITLE></HEAD>
  <SCRIPT>

  </SCRIPT>
</HTML>
```

Здесь между тегами `<SCRIPT>` и `</SCRIPT>` оставлено пустое место для размещения кода сценария. По существу, это HTML-документ, определяющий пустую Web-страницу. Сохраните файл на диске, например, под именем примеры.htm. Расширение имени файла должно быть htm или html, поскольку мы хотим, чтобы он исполнялся в Web-браузере.

Внимание

При изучении данной главы записывайте в каждой строке не более одного выражения на языке JavaScript. Заканчивайте строку нажатием клавиши <Enter>, чтобы перейти к новой строке. Если требуется в одной строке написать не-

сколько выражений, то их следует отделять друг от друга точкой с запятой. Хорошим стилем считается, если любое выражение заканчивать точкой с запятой. Однако в данной книге автор не следует этой рекомендации.

В листинге 2.1 приведен пример программы.

Листинг 2.1. Пример программы с использованием сценария

```
<HTML>
  <HEAD><TITLE>Примеры</TITLE></HEAD>
  <SCRIPT>
    x=5
    y=x+4
    alert(y)
  </SCRIPT>
</HTML>
```

При выполнении учебных примеров обычно требуется вывести на экран окончательные или промежуточные данные. Для этого можно использовать метод `alert()`, указав в круглых скобках то, что требуется вывести на экран. В приведенном выше примере метод `alert(y)` выведет на экран диалоговое окно, в котором отобразит значение переменной `y` (в данном случае — число 9), как показано на рис. 2.1. Если не вставлять строку `alert(y)`, то программа выполняется без отображения результата.

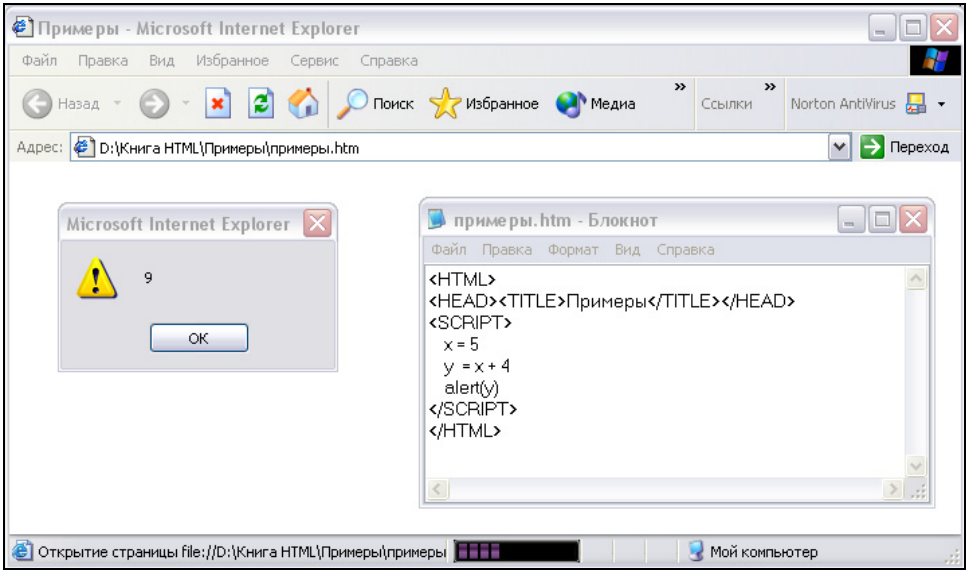


Рис. 2.1. HTML-код в текстовом редакторе и результат его выполнения браузером

Если вы хотите выводить результаты вычислений не в диалоговом окне, а непосредственно в окне Web-браузера, то вместо `alert(y)` напишите следующее выражение:

```
document.writeln(y)
```

Чтобы открыть файл (в нашем случае это примеры.htm) в Web-браузере и выполнить содержащуюся в нем программу, достаточно просто дважды щелкнуть левой кнопкой мыши на его имени в окне Проводника Windows. Если теперь потребуется открыть этот файл в окне текстового редактора, то щелкните правой кнопкой мыши где-нибудь на свободном пространстве в окне браузера и в контекстном меню выберите команду **Просмотр HTML-кода**. Можно также воспользоваться и командой меню браузера **Вид | Просмотр HTML-кода**. В результате откроется окно текстового редактора с текстом вашей программы. Внесите необходимые изменения и сохраните их на диске (**Файл | Сохранить**). Чтобы Web-браузер смог отобразить измененный файл, выполните команду меню браузера **Вид | Обновить**. Таким образом, переходя от текстового редактора к браузеру и обратно, можно разрабатывать и отлаживать свои программы.

Если код сценария содержит ошибки, то в строке состояния браузера Internet Explorer появится пиктограмма в виде треугольника с восклицательным знаком и будет открыто диалоговое окно с кратким описанием ошибки. При желании вывод сообщений об ошибках можно отключить в настройках браузера.

2.2. Ввод и вывод данных

Для ввода/вывода данных программ на JavaScript, выполняемых Web-браузером, предусмотрены несколько методов объектов `window` (окно) и `document` (документ), содержащих свойства Web-браузера и загруженного в него документа соответственно. Рассмотрим эти методы браузера подробнее.

2.2.1. Метод *alert*

Метод `alert` объекта `window` позволяет выводить диалоговое окно с заданным сообщением и кнопкой **OK**. Синтаксис соответствующего выражения имеет вид:

alert(*сообщение*)

Если ваше *сообщение* конкретно, т. е. представляет собой вполне определенный набор символов, то его необходимо заключить в двойные или одинарные кавычки. Например, `alert("Привет!")`. Вообще говоря, *сообщение* представ-

ляет собой данные любого типа: последовательность символов, заключенную в кавычки, число (в кавычках или без них), переменную или выражение.

Диалоговое окно, выведенное на экран методом `alert()`, можно закрыть, щелкнув на кнопке **OK**. До тех пор, пока вы не сделаете этого, переход к ранее открытым окнам невозможен. Окна, обладающие свойством останавливать все последующие действия пользователя и программ, называются *модальными*. Таким образом, окно, создаваемое посредством метода `alert()`, является модальным.

Внимание

Метод `alert()` можно использовать для вывода промежуточных и окончательных результатов программ при их отладке. При этом можно вывести результат вычисления какого-либо выражения и приостановить дальнейшее выполнение работы программы до тех пор, пока вы не щелкните на кнопке **OK**.

2.2.2. Метод *confirm*

Метод `confirm` объекта `window` позволяет вывести диалоговое окно с сообщением и двумя кнопками — **OK** и **Отмена** (Cancel). В отличие от метода `alert()`, этот метод возвращает логическую величину, значение которой зависит от того, на какой из двух кнопок щелкнул пользователь. Если он щелкнул на кнопке **OK**, то возвращается значение `true` (истина, да); если же он щелкнул на кнопке **Отмена** (Cancel), то возвращается значение `false` (ложь, нет). Возвращаемое значение можно обработать в программе и, следовательно, создать эффект интерактивности, т. е. диалогового взаимодействия программы с пользователем. Синтаксис применения метода `confirm()` имеет следующий вид:

`confirm(`*сообщение*`)`

Если ваше *сообщение* конкретно, т. е. представляет собой вполне определенный набор символов, то его необходимо заключить в кавычки, двойные или одинарные. Например,

```
confirm("Вы действительно хотите выйти из программы?")
```

Вообще говоря, *сообщение* представляет собой данные любого типа: последовательность символов, заключенную в кавычки, число (в кавычках или без них), переменную или выражение.

Диалоговое окно, выведенное на экран методом `confirm()`, можно убрать щелчком на любой из двух кнопок, **OK** или **Отмена**. До тех пор, пока вы не сделаете этого, переход к ранее открытым окнам не возможен. Окна, обладающие свойством останавливать все последующие действия пользователя

и программ, называются модальными. Окно, создаваемое посредством `confirm()`, также является модальным. Если пользователь щелкнет на кнопке **OK**, то метод вернет логическое значение `true` (истина, да), а если он щелкнет на кнопке **Отмена** (Cancel), то возвращается логическое значение `false`. Возвращаемое значение можно затем обработать в программе и, следовательно, создать эффект интерактивности, т. е. диалогового взаимодействия программы с пользователем.

2.2.3. Метод *prompt*

Метод `prompt()` объекта `window` позволяет вывести на экран диалоговое окно с сообщением, а также с текстовым полем, в которое пользователь может ввести данные. Кроме того, в окне предусмотрены две кнопки: **OK** и **Отмена** (Cancel). В отличие от методов `alert()` и `confirm()`, данный метод принимает два параметра: сообщение и значение, которое должно появиться в текстовом поле ввода данных по умолчанию. Если пользователь щелкнет на кнопке **OK**, то метод вернет содержимое поля ввода данных, а если он щелкнет на кнопке **Отмена**, то возвращается логическое значение `false` (ложь, нет). Возвращаемое значение можно затем обработать в программе и, следовательно, создать эффект интерактивности, т. е. диалогового взаимодействия программы с пользователем.

Синтаксис применения метода `prompt()` имеет следующий вид:

```
prompt(сообщение, значение_поля_ввода_данных)
```

Параметры метода `prompt()` не являются обязательными. Если вы их не укажете, то будет выведено окно без сообщения, а в поле ввода данных подставлено значение по умолчанию — `undefined` (не определено). Если вы не хотите, чтобы в поле ввода данных появлялось значение по умолчанию, то подставьте в качестве значения второго параметра пустую строку `""`. Например:

```
prompt("Введите Ваше имя, пожалуйста", "")
```

Диалоговое окно, выведенное на экран методом `prompt()`, можно закрыть щелчком на любой из двух кнопок, **OK** или **Отмена**. Как и в случае использования метода `confirm()`, переход к ранее открытым окнам не возможен. Таким образом, это диалоговое окно является модальным.

2.2.4. Метод *write*

Метод `write` принадлежит объекту `document`, который содержит свойства документа, загруженного в Web-браузер. Он позволяет вывести данные в текущий документ.

Метод `write()` принимает в качестве параметра строку, содержащую HTML-код и/или просто текст. Выполнение в сценарии выражения `document.write(`*`строка`*`)` приводит к дописыванию в текущий HTML-документ содержимого параметра *`строка`* и немедленной его интерпретации браузером. В результате документ и его объектная модель обновляются. При этом файл с исходным HTML-кодом остается без изменений. Если требуется полностью заменить текущий документ, то сначала применяют метод очистки документа `document.clear()`, а затем `document.write(`*`строка`*`)`. Однако при такой кардинальной трансформации текущего документа следует быть осторожным. Наиболее безопасный прием — сначала сгенерировать содержимое нового HTML-документа с помощью сценария в текущем документе, а затем отправить HTML-код в новое окно или в другой фрейм многофреймового документа.

Если быть более точным, то следует отметить, что метод `write()` может принимать произвольное количество строковых параметров:

```
document.write(строка1[, строка2 ...[, строкаN]])
```

Здесь квадратные скобки указывают лишь на необязательность заключенных в них параметров. Если указывается несколько параметров, то они разделяются запятыми.

Следующий код выводит в окно браузера обычную текстовую строку "Привет!" и графическое изображение из файла picture.jpg:

```
<HTML>
  Привет!
  <BR>
  <SCRIPT>
    Document.write("<IMG SRC=picture.jpg>")
  </SCRIPT>
</HTML>
```

Кроме метода `write()`, можно использовать для тех же целей и метод `writeln()`, имеющий такой же синтаксис. Его особенность в том, что он добавляет в конце каждой строки документа невидимый символ перехода на другую строку.

Методы `write()` и `writeln()` работают в браузерах IE3+ и NN2+.

2.3. Типы данных

Язык JavaScript относится к языкам с так называемыми *свободными типами данных*. Это означает, что применение операций к данным, для которых они не предназначены, не приведет к останову программы. Кроме того, имеется

возможность одной и той же переменной присваивать данные то одного, то другого типа. Этой возможностью следует пользоваться осторожно во избежание недоразумений.

Тип данных — некоторое ограничение на множество возможных значений. Если мы каким-либо образом выделяем (ограничиваем) некоторое множество значений, то тем самым мы определяем тип данных.

В JavaScript определены следующие основные (или, как еще говорят, примитивные) типы данных, приведенные в табл. 2.1.

Таблица 2.1. *Типы данных в JavaScript*

Тип данных	Примеры	Описание значений

Строковый или символьный (`String`)	`"Привет всем!"`
	`"д.т. 123-4567"`
	`'Сегодня 15.01.2006г.'`
		Последовательность символов, заключенная в кавычки, двойные или одинарные

Числовой (`Number`)	`3.14`
	`-567`
	`+2.5`
		Число, последовательность цифр, перед которой может быть указан знак числа (+ или –); перед положительными числами не обязательно ставить знак +; целая и дробная части чисел разделяются точкой. Число записывается без кавычек

Логический (булевый, `Boolean`)	`true` `false`	Этот тип данных имеет только два возможных значения: `true` (истина, да) или `false` (ложь, нет)
`Null`	`null`	Этот тип данных имеет одно значение — `null`, указывающее на отсутствие какого бы то ни было значения
`Undefined`	`undefined`	Этот тип данных имеет одно значение — `undefined`, указывающее, что переменной не присвоено никакое значение
Объект (`Object`)		Программный объект, определяемый своими свойствами. В частности, массив также является объектом
Функция (`Function`)		Определение функции — программного кода, выполнение которого может возвращать некоторое значение

На данном этапе следует обратить особое внимание на различия в представлении числовых и строковых (символьных) данных. Числа как данные числового типа всегда представляются без кавычек. Для их написания используются цифры и, при необходимости, знак числа и разделительная точка. Строковые данные заключаются в кавычки, двойные или одинарные. Например, запись `-345.12` представляет число, а запись `"-345.12"` — строку символов. Это данные различных типов, хотя мы и можем сказать, что их содержанием является одно и то же число.

Заметим, что строка `""`, не содержащая ни одного символа (даже пробела), называется *пустой*. При этом строка, содержащая хотя бы один пробел (например, `" "`), не пуста.

Данные логического типа могут иметь одно из двух значений: `true` или `false`. Эти значения записываются без кавычек. Значение `true` означает истину (да), а `false` — ложь (нет). Обычно эти значения получаются в результате вычисления выражений с использованием операций сравнения двух данных, а также логических операций (И, ИЛИ, НЕ). Например, результатом вычисления выражения `3<5` является, очевидно, значение `true`. Логический тип данных называют еще булевым (`Boolean`) в честь английского математика Джона Буля, придумавшего алгебру для логических величин.

Другие типы данных (`Null`, `Undefined`, `Object` и `Function`) будут рассмотрены в следующих разделах.

При создании программ на JavaScript за типами данных следит сам программист. Если он напутает с типами, то интерпретатор не зафиксирует ошибки, а попытается привести данные к некоторому типу, чтобы выполнить указанную операцию. Вам следует разобраться, к какому именно типу приводится смесь данных различного типа.

Например, если вы напишете выражение `25+"Вася"`, то результатом его вычисления будет строка символов `"25Вася"`. Таким образом, интерпретатор, столкнувшись с выражением сложения числа и строки символов, переводит число в строку, содержащую это число, а затем выполняет операцию сложения двух строк. Сложение двух строк в JavaScript дает в результате строку, получающуюся путем приписывания второй строки к концу первой.

Результатом вычисления выражения `2+3` будет число 5, а выражения `2+"3"` — строка `"23"`, состоящая из двух цифровых символов. Как нетрудно заметить, в случае применения операции сложения к числу и строке символов интерпретатор преобразует число в строку символов и возвращает результат вычисления выражения тоже в виде строки символов. Иначе говоря, в случае смысловой несогласованности типов данных интерпретатор использует некоторые правила их согласования, принятые по умолчанию. В результате могут

появиться трудно выявляемые ошибки. С другой стороны, эту особенность языка можно использовать, соблюдая известную осторожность, для написания изящных и компактных кодов.

При программировании часто бывает, что ранее полученные данные принадлежат одному типу, а имеющиеся средства дальнейшего их использования требуют, чтобы данные относились к другому типу. Например, некоторое число находится где-то внутри текста (данных строкового типа), а нам нужно использовать его в определенной арифметической операции (например, операции суммирования с другими числами). С помощью специальных строковых операций мы выделим из исходного текста нужное число, но полученные в результате данные будут относиться к строковому типу, хотя и состоять при этом из цифр и, возможно, разделительной точки. Чтобы применить к этим данным некую арифметическую операцию, придется сначала привести их к числовому типу. Для этого и служат специальные средства (функции, методы) преобразования типов данных. Их задача сводится к тому, чтобы изменить тип данных, не потеряв при этом их содержание (семантику). Впрочем, сохранить содержание данных — задача программиста, а специальные функции и методы преобразования типов поддерживают эту задачу в той или иной мере.

Для приведения данных к заданному типу в JavaScript предусмотрены встроенные функции конвертирования типов или приведения к заданному типу. Что такое функция, мы подробно рассмотрим ниже. А сейчас считайте, что это выражение, состоящее из имени функции с круглыми скобками, в которых можно указывать параметры. В результате вычисления функции получается некоторое значение, которое можно присвоить переменной или использовать в выражении.

Для преобразования некоторого выражения в число служит функция

`Number(`*`выражение`*`)`

результат вычисления которой — десятичное число или `NaN` (если не удалось преобразовать указанное *`выражение`* в число). Эта функция работает следующим образом:

- если *`выражение`* есть число, то возвращается это число;
- если *`выражение`* есть логическая величина, то возвращается 1 или 0 в зависимости от ее значения (если `true`, то 1; если `false`, то 0);
- если *`выражение`* есть строка, то функция пытается преобразовать ее в число как описанные далее функции `parseInt()` и `parseFloat()`;
- если *`выражение`* есть `udefined` (не определено), то результатом является `NaN` (не число).

Примеры:

```
Number(25.3)          // результат =25.3
Number("25")          // результат =25
Number("25 рублей")   // результат =NaN, т. е. не является числом
Number("1.2e+10")     // результат =12000000000
Namber(true)          // результат =1
Namber(false)         // результат =0
```

Для преобразования строк в числа в JavaScript предусмотрены встроенные функции `parseInt()` и `parseFloat()`.

Функция

`parseInt(`*строка*`, `*основание*`)`

преобразует указанную в параметре строку в целое число в системе счисления по заданному основанию (8, 10 или 16). Если основание не указано, то предполагается 10, т. е. десятичная система счисления.

Примеры:

```
parseInt("3.14")      // результат =3
parseInt("-7.875")    // результат =-7
parseInt("435")       // результат =435
parseInt("Вася")      // результат =NaN, т. е. не является числом
parseInt("15",8)      // результат =13
parseInt("0xFF",16)   // результат =255
```

Обратите внимание, что при преобразовании в целое число округление не происходит: дробная часть просто отбрасывается.

Функция

`parseFloat(`*строка*`)`

преобразует указанную строку в число с плавающей разделительной (десятичной) точкой.

Примеры:

```
parseFloat("3.14")    // результат =3.14
parseFloat("-7.875")  // результат =-7.875
parseFloat("123")     // результат =123
parseFloat("Вася")    // результат =NaN, т. е. не является числом
parseFloat("100 руб.")// результат =NaN, т. е. не является числом
parseFloat("17.5")    // результат =435
```

Задача преобразования чисел в строки возникает реже, чем обратное преобразование. Чтобы преобразовать число в строку, достаточно к пустой строке прибавить это число, т. е. воспользоваться оператором сложения `+`. Например, вычисление выражения `""+3.14` даст в результате строку `"3.14"`.

Для определения того, является ли значение выражения числом, служит встроенная функция

`isNaN(`*значение*`)`

Вычисление этой функции дает результат логического типа. Если указанное значение не является числом, функция возвращает `true`, иначе — `false`. Однако здесь понятие "число" не совпадает с понятием "значение числового типа". Функция `isNaN()` считает числом и данные числового типа, и строку, содержащую только число. Логические значения также идентифицируются как числа. При этом значению `true` соответствует 1, а значению `false` — 0. Таким образом, если `isNaN()` возвращает `false`, то это указывает, что значение параметра имеет числовой тип, либо является числом, преобразованным в строковый тип, либо является логическим (`true` или `false`).

Примеры:

```
isNaN(1234)            // результат false (т. е. это число)
isNaN("1234")          /* результат false (т. е. это число,
                          хотя и в виде строки) */
isNaN("25 рублей")     // результат true (т. е. это не число)
isNaN(true)            // результат false
isNaN(false)           // результат false
isNaN("Саша")          // результат true (т. е. это не число)
```

При создании строковых данных применяются кавычки. Правильное использование кавычек является очень важным и довольно часто вызывает проблемы у новичков. Здесь основным является правило: кавычки, обрамляющие строковые данные, должны быть одного вида и использоваться парами (подобно скобкам). Интерпретатор, встретив в тексте программы кавычки, будет искать еще кавычки такого же вида, считая все находящееся между ними строковыми данными. Внутри строки, заключенной в кавычки одного вида, можно использовать кавычки другого вида (иначе интерпретатор либо выдаст сообщение об ошибке, либо неправильно воспримет данные). Например, следующие выражения правильны:

```
'Акционерное общество "Рога и копыта"'
"Акционерное общество 'Рога и копыта'"
```

Кроме рассмотренных ранее, в JavaScript имеются и другие функции приведения данных к заданному типу.

- `String(`*`выражение`*`)` — приводит данные, получающиеся в результате вычисления выражения, которое указано в качестве параметра, к строковому типу. Преобразование происходит по следующим правилам:
  - если значение выражения имеет строковый тип, то возвращается это же значение;
  - если значение выражения имеет числовой тип, то возвращается строка, содержащая это число;
  - если значение выражения имеет логический тип, то возвращается строка `"true"` или `"false"` в зависимости от значения выражения;
  - если значение выражения не определено (`undefined`), то возвращается пустая строка `""`.
- `Boolean(`*`выражение`*`)` — приводит данные, получающиеся в результате вычисления выражения, которое указано в качестве параметра, к логическому типу. Преобразование происходит по следующим правилам:
  - если значение выражения имеет логический тип, то возвращается это же значение;
  - если значение выражения имеет числовой тип, то возвращается `true`, если число не равно 0, и `false` — в противном случае;
  - если значение выражения имеет строковый тип, то возвращается `true`, если строка не пуста, и `false` — в противном случае; напомним, что строка, содержащая только пробелы, не пуста.
- `Array(`*`элемент0`*`[, `*`элемент1`*`, ...[, `*`элементN`*`]])` — создает массив из элементов, указанных в качестве параметров.

Для проверки, относится ли значение к типу `String`, `Number`, `Boolean`, `Object`, `Function` или `Undefined`, служит оператор `typeof`. Значение, возвращаемое этим оператором, является строковым. Оно содержит одно из перечисленных выше названий типа. Единственный операнд пишется справа от ключевого слова `typeof`.

Примеры:

```
var x
n=3.14
N=new Number(3.14)
s="Привет всем!"
a =new Array(1, 2, 3, 4, 5)
function f(){}
```

```
typeof x   // "undefined"
typeof n   // "number"
typeof N   // "object"
typeof s   // "string"
typeof a   // "object"
typeof f   // "function"
```

Возможна также запись оператора `typeof` как функции, например, `typeof(x)`.

2.4. Переменные и оператор присваивания

2.4.1. Имена переменных

Переменная является контейнером для хранения данных. Данные, сохраняемые в переменной, называются *значениями* этой переменной. Переменная имеет имя — последовательность букв, цифр, символов подчеркивания (_) и доллара ($) без пробелов и знаков препинания, начинающаяся обязательно с буквы, символа подчеркивания или доллара. Иначе говоря, имя переменной не должно начинаться с цифры.

Примеры правильных имен переменных: `myName`, `my_adress`, `_x`, `tel123_4567`, `$var`.

Примеры неправильных имен переменных: `512group`, `my adress`, `tel:412 3456`.

Желательно, чтобы имя отражало содержание переменной. Если имя состоит из нескольких слов, то между ними можно вводить символ подчеркивания или писать их слитно, начиная каждое слово с прописной буквы. Вот примеры: `my_first_adress`, `myFirstAdress`. Иногда в качестве первого символа имени используют букву, указывающую на тип данных (значений) этой переменной: `c` — строковый (`Character`), `n` — числовой (`Number`), `b` — логический (`Boolean`), `o` — объект (`Object`), `a` — массив (`Array`). Например, `cAdress`, `nCena`, `aMonth`.

Внимание

JavaScript является регистрозависимым языком. Это означает, что изменение регистра символов (с прописных на строчные и наоборот) в имени переменной приводит к другой переменной. Например, `variable`, `Variable` и `vaRiabLe` — различные переменные. Кроме того, регистрозависимыми являются и ключевые слова, такие как `if`, `function`, `Array` и др.

Совет

Выработайте для себя правила образования имен, которые не должны противоречить указанным выше требованиям. Если им следовать, то уменьшится вероятность возникновения ошибок в ваших программах.

При выборе имен переменных нельзя использовать ключевые слова, т. е. слова, зарезервированные для конструкций языка. Например, нельзя выбирать слова `var`, `if`, `else`, `const`, `true`, `false`, `function`, `super`, `switch` и ряд других. Список зарезервированных ключевых слов таков:

abstract	else	int	super
boolean	extends	interface	switch
break	false	long	synchronized
byte	final	native	this
case	finally	new	throw
catch	float	null	throws
char	for	package	transient
class	function	private	true
const	goto	protected	try
continue	if	public	typeof
default	Implements	reset	var
delete	import	return	void
do	in	short	while
double	instanceof	static	with

2.4.2. Создание переменных

Создать переменную в программе можно несколькими способами.

- С помощью оператора присваивания значений в формате:

  имя_переменной=значение

  Оператор присваивания обозначается символом равенства =.

  Пример: `x="Вася"`.

- С помощью ключевого слова `var` (от англ. *variable* — переменная) в формате:

  `var` *имя_переменной*

  В этом случае созданная переменная не имеет никакого значения, но может его получить в дальнейшем с помощью оператора присваивания.

Пример:

```
var x
x="Вася"
```

- С помощью ключевого слова `var` и оператора присваивания в формате:

  `var` *`имя_переменной=значение`*

  Пример: `var x="Вася"`

Строка кода программы с ключевым словом `var` задает так называемую *инициализацию переменной* и для каждой переменной используется один раз. Тип переменной определяется типом значения, которое она имеет. В отличие от многих других языков программирования, при инициализации переменной не нужно описывать ее тип. Переменная может иметь значения различных типов и неоднократно их изменять. Например, вы можете написать следующий код программы:

```
var x="Вася"
  // некоторый код
x="Петр"
  // некоторый код
x=12.5
```

Одно ключевое слово `var` можно использовать для инициализации сразу нескольких переменных, как с оператором присваивания, так и без него. При этом переменные и выражения с операторами присваивания разделяются запятыми, например:

```
var name="Вася", address, x=3.14
```

Если переменная в данный момент имеет значение числового типа, то говорят, что это числовая переменная. Аналогично можно говорить о строковых, логических (булевых), неопределенных (в случае типа `Null` и `Undefined`) переменных.

Выше мы использовали оператор присваивания значения переменной, обозначаемый символом равенства `=`. Не следует путать этот оператор с отношением равенства и соответствующей операцией сравнения. Выражение с оператором `=` интерпретатор вычисляет следующим образом: переменной слева от него присваивается значение, расположенное справа от него.

Если `x` и `y` — две переменные, то выражение `x=y` интерпретируется так: переменной `x` присваивается значение переменной `y`. Иначе говоря, переменной можно присвоить значение другой переменной, указав справа от оператора `=` ее имя. Таким образом, к значениям можно получать доступ опосредованно — через имена переменных. В следующих разделах мы узнаем, что справа от

оператора присваивания можно писать не только значения и имена переменных, но и выражения. В *разд. 2.5.3* будут рассмотрены дополнительные операторы присваивания, такие как `+=`, `-=` и т. п.

2.4.3. Область действия переменных

Переменные, которые созданы в программе с помощью оператора присваивания с использованием ключевого слова `var` или без него, являются *глобальными*. Это означает, что такие переменные доступны всюду в этой же программе, а также в вызываемых программах из других файлов. Эти переменные также доступны внутри кода функций *(см. разд. 2.6.2)*.

Кроме глобальных существуют и *локальные* переменные. Они могут быть созданы внутри кода функций. Вы можете определить переменные с одинаковыми названиями и во внешней программе, и в коде функции. В этом случае переменные, определенные в коде функции с помощью ключевого слова `var`, будут локальными: изменения их значений внутри функции никак не отразятся на переменных с такими же именами, определенных во внешней программе. При этом значения локальных переменных не доступны из внешней программы.

Мы еще неоднократно будем говорить об *области действия переменных*. Здесь лишь отметим, что в программировании это понятие играет очень важную роль. Нередко область действия называют областью видимости. Переменная может быть видна или не видна внутри программной единицы (функции, подпрограммы). Область видимости, доступности или действия переменной — эквивалентные термины. Кроме них еще используют понятие *времени жизни переменной*. Время жизни переменных в JavaScript определяется интервалом времени от загрузки до выгрузки программы из памяти компьютера. Так, если программа (сценарий) записана в HTML-коде Web-страницы, то после ее выгрузки вся программа (сценарий) вместе с определенными в ней переменными прекращает активное существование.

2.5. Операторы

Операторы предназначены для составления выражений. Оператор применяется к одному или двум данным, которые в этом случае называются *операндами*. Например, оператор сложения применяется к двум операндам, а оператор логического отрицания — к одному операнду. *Элементарное выражение*, состоящее из операндов и оператора, вычисляется интерпретатором и, следо-

вательно, имеет некоторое значение. В этом случае говорят, что оператор возвращает значение. Например, оператор сложения, примененный к числам 3 и 5, возвращает значение 8. Оператор имеет тип, совпадающий с типом возвращаемого им значения. Поскольку элементарное выражение с оператором и операндами возвращает значение, это выражение можно присвоить переменной.

Один оператор мы уже рассмотрели в предыдущем разделе. Это оператор присваивания `=`. Мы часто будем его использовать. Однако следует заметить, что существуют еще пять разновидностей оператора присваивания, сочетающих в себе действия обычного оператора `=` и операторов сложения, вычитания, умножения, деления и деления по модулю. Эти дополнительные операторы присваивания мы рассмотрим в *разд. 2.5.3*.

2.5.1. Комментарии

Операторы комментария позволяют выделить фрагмент программы, который не выполняется интерпретатором, а служит лишь для пояснений содержания программы.

В JavaScript допустимы два вида оператора комментария:

- `//` — одна строка символов, расположенная справа от этого оператора, считается комментарием;
- `/*...*/` — все, что заключено между `/*` и `*/`, считается комментарием; с помощью этого оператора можно выделить несколько строк в качестве комментария.

Примеры:

```
x=y +15   // Это однострочный комментарий
// Это однострочный комментарий
z="Добро пожаловать!"  /* Однострочный комментарий справа от выражения */
/* Это однострочный комментарий слева от выражения */ x=3 + 4
var xyz    /* Это двухстрочный
              комментарий */
```

2.5.2. Арифметические операторы

Арифметические операторы, такие как сложение, умножение и т. д., в JavaScript могут применяться к данным любых типов. Что из этого получается, мы рассмотрим немного позже. А сейчас просто перечислим их (табл. 2.2).

Таблица 2.2. *Арифметические операторы*

Оператор	Название	Пример
+	Сложение	X+Y
-	Вычитание	X-Y
*	Умножение	X*Y
/	Деление	X/Y
%	Деление по модулю	X%Y
++	Увеличение на 1	X++ ++X
--	Уменьшение на 1	X-- --X

Арифметические операторы лучше всего понять на примере чисел. Первые четыре оператора не требуют особых разъяснений. Оператор деления по модулю возвращает остаток от деления первого числа на второе. Например, `8%3` возвращает 2. При этом результат будет отрицательным числом, если хотя бы один из операндов отрицателен. Операторы `++` и `--` сочетают в себе действия операторов соответственно сложения и вычитания, а также присваивания. Выражение `X++` эквивалентно выражению `X=X+1`, однако оно возвращает не новое, а прежнее значение переменной `X`. Таким образом, оператор `X++` сначала возвращает текущее значение `X`, а затем увеличивает его на 1. В противоположность этому оператор `++X` сначала увеличивает значение переменной `$X` на 1, а затем возвращает новое значение. Аналогично, оператор `X--` сначала возвращает текущее значение `X`, а затем уменьшает его 1. Оператор `--X` делает это в противоположном порядке.

Операторы `++` и `--` называют соответственно *инкрементом* и *декрементом*.

Символ `-` используется не только как бинарный (для двух операндов) оператор вычитания, но и как унарный (для одного операнда) оператор отрицания для указания того, что число является отрицательным.

Примеры:

```
y=5     // значение переменной y равно 5
x=y+3   // значение переменной x равно 8 (5+3)
x++     // значение переменной x стало равным 9 (8+1)
x--     // значение переменной x стало равным 8 (9-1)
y++     // значение переменной y равно 6 (5+1)
5-3     // значение равно 2
-3      // отрицательное число
```

Однако формально ничто не мешает нам применить эти операторы к данным других типов. В случае строковых данных оператор сложения дает в результате строку, полученную путем приписывания справа второй строки к первой. Например, выражение `"Саша"+"Маша"` возвращает в результате `"СашаМаша"`. Поэтому для строк оператор сложения называется *оператором склейки* или *конкатенации*. Применение остальных арифметических операторов к строкам дает в результате `NaN` — значение, не являющееся числом. В случае, когда оператор сложения применяется к строке и числу, интерпретатор переводит число в соответствующую строку и далее действует как оператор склейки двух строк.

Примеры:

```
x="Саша"        // значение переменной x равно "Саша"
y="Маша"        // значение переменной y равно "Маша"
z=x+" "+y       // значение переменной z равно "Саша Маша"
z=x+5           // значение переменной z равно "Саша5"
n="200"+6       // значение переменной n равно "2006", а не 206 или "206"
```

В случае логических данных интерпретатор переводит логические значения операндов в числовые (`true` в 1, `false` в 0), выполняет вычисление и возвращает числовой результат. То же самое происходит, когда один оператор логический, а другой — числовой.

Примеры:

```
true + true      // возвращает 2
true + false     // возвращает 1
true * true      // возвращает 1
true/false       /* возвращает Infinity (неопределенность, т. к. на
                    ноль делить нельзя) */
true + 5         // возвращает 6
false + 5        // возвращает 5
true * 5         // возвращает 5
true/5           // возвращает 0.2
```

Если один операнд строкового типа, а другой — логического, то в случае сложения интерпретатор переводит оба операнда в строковый тип и возвращает строку символов, а в случае других арифметических операторов он переводит оба операнда в числовой тип.

Примеры:

```
"Вася" + true     // возвращает "Васятrue"
"5" + true        // возвращает "5true"
```

```
"Вася" * true      /* возвращает NaN (т. е. значение,
                      не являющееся числом) */
"5" * true         // возвращает 5
"5" * false        // возвращает 0
"5"/true           // возвращает 5
```

На первый взгляд, применение арифметических операторов к разнотипным данным может показаться довольно запутанным. Поэтому советую новичкам, особенно на первых порах, руководствоваться следующими простыми правилами:

- используйте арифметические операторы с данными одного и того же типа;
- оператор сложения применительно к строкам действует как оператор их склейки (приписывания, конкатенации);
- в случае применения арифметических операторов к логическим данным интерпретатор рассматривает значения `true` и `false` как числа соответственно 1 и 0 и возвращает результат в числовом виде.

Кроме этих правил нужно помнить, что в JavaScript имеются средства для преобразования типов данных, т. е. для преобразования данных одного типа в данные другого типа *(см. разд. 2.3)*.

2.5.3. Дополнительные операторы присваивания

Кроме обычного оператора присваивания = имеются еще пять дополнительных операторов, сочетающих в себе действия обычного оператора присваивания и арифметических операторов (табл. 2.3).

Таблица 2.3. *Операторы присваивания*

Оператор	Пример	Эквивалентное выражение
+=	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

Поскольку действия арифметических операторов и обычного оператора присваивания уже известны, то дополнительные операторы присваивания мы не будем здесь подробно рассматривать. Обратим лишь внимание на экономный способ записи выражений, предоставляемый этими операторами.

Пример:

```
x="Вася"
x+=", привет"   // "Вася, привет"
x=5; x*=7       // 35
```

2.5.4. Операторы сравнения

В программах часто приходится проверять выполнение каких-либо условий. В зависимости от результата процесс дальнейших вычислений может пойти по тому или другому пути. Проверяемые условия формируются на основе операторов сравнения, таких как "равно", "меньше", "больше" и т. п. (табл. 2.4). Результатом вычисления элементарного выражения, содержащего оператор сравнения и операнды (сравниваемые данные), является логическое значение, т. е. `true` или `false`. Так, если условие выполняется (верно, справедливо), то возвращается `true`. В противном случае возвращается `false`.

Таблица 2.4. *Операторы сравнения*

Оператор	Название	Пример
==	Равно	X==Y
!=	Не равно	X!=Y
<>	Не равно	X<>Y
>	Больше, чем	X>Y
>=	Больше или равно (не меньше)	X>=Y
<	Меньше, чем	X<Y
<=	Меньше или равно (не больше)	X<=Y

Внимание

Оператор "равно" записывается с помощью двух символов == без пробелов между ними.

Сравнивать можно числа, строки и логические значения. Сравнение чисел происходит по правилам арифметики, а строк — путем сравнения ASCII-кодов символов, начиная с левого конца строк. Логические значения сравниваются так же, как и числа 1 и 0 (`true` соответствует 1, а `false` — 0). Как видим, с числами и логическими данными все довольно просто. А вот результат сравнения строк не всегда очевиден.

Примеры:

```
"abcd" == "abc"          // возвращает false
"abc" == "abcd"          // возвращает false
"abcd" == "abcd"         // возвращает true
"abcd" == " abcd"        /* возвращает false
                            (1-й символ 2-го операнда — пробел) */
"abcd" > " abcd"         // возвращает true
"abc" < "abcd"           // возвращает true
"235ab" < "abcdxyz"      // возвращает true
"235xyz" < "abc"         // возвращает true
```

Мы не приводим здесь таблицу кодов ASCII для всех символов. Отметим лишь, что некоторые из них упорядочены по возрастанию ASCII-кодов следующим образом: сначала идет пробел, затем в порядке возрастания цифры от 0 до 9, символ подчеркивания, латинские и кириллические буквы по алфавиту (сначала прописные, а затем строчные).

Заметим, что операторы сравнения могут быть применены и к разнотипным данным. Если сравниваются строка и число, то интерпретатор приводит операнды к числовому типу. То же самое происходит при сравнении логических данных и числа. Если сравниваются логические данные и строка, то дело обстоит несколько сложнее. В этом случае результат не зависит от содержимого строки. Если она хранит число (но не цифры с буквами), только пробелы или является пустой, то операнды приводятся к числовому типу. При этом пустая строка (`""`) и строка, содержащая только пробелы, преобразуется в число 0. В остальных случаях все операторы сравнения кроме `!=` будут возвращать `false` (а оператор `!=` — противоположный результат, т. е. `true`).

Примеры:

```
false < "57"    // возвращает true
false < "-2.5"  // возвращает false
true >= "0.5"   // возвращает true

false < "57ab"  // возвращает false
false > "57ab"  // возвращает false
true == "true"  // возвращает false
true != "true"  // возвращает true

true <= "true"  // возвращает false
true >= "true"  // возвращает false
```

2.5.5. Логические операторы

Логические данные, обычно получаемые с помощью элементарных выражений, содержащих операторы сравнения, можно объединять в более сложные выражения. Для этого используются логические (булевы) операторы — логические союзы И и ИЛИ, а также оператор отрицания НЕ (табл. 2.5). Например, нам может потребоваться сформировать сложное логическое условие вида: "возраст не более 30 и опыт работы больше 10, или юрист". В этом примере есть и операторы сравнения, и логические операторы. Выражения с логическими операторами возвращают значение `true` или `false`.

Таблица 2.5. *Логические операторы*

Оператор	Название	Пример
`!`	Отрицание (НЕ)	`!X`
`&&`	И	`X&&Y`
`||`	ИЛИ	`X||Y`

Оператор отрицания `!` применяется к одному операнду, изменяя его значение на противоположное: если `X` имеет значение `true`, то `!X` возвращает значение `false`, и наоборот, если `X` имеет значение `false`, то `!X` возвращает значение `true`.

В табл. 2.6 указано, какие значения возвращают операторы И, ИЛИ при различных логических значениях двух операндов.

Таблица 2.6. *Примеры использования операторов И, ИЛИ*

X	**Y**	**X&&Y**	**X||Y**
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Операторы `&&` и `||` еще называют *логическим умножением* и *логическим сложением* соответственно. Если вспомнить, что значению `true` можно сопоставить 1, а значению `false` — 0, то нетрудно понять, как вычисляются значения элементарных выражений с логическими операторами. Нужно только учесть, что в алгебре логики 1 + 1 = 1 (а не 2). Аналогично, оператору отрицания соответствует вычитание из единицы числового эквивалента

логического значения операнда. В математике логические операции И и ИЛИ называют соответственно *конъюнкцией* и *дизъюнкцией*.

Совет

Чтобы не запутаться, не применяйте логические операторы к нелогическим данным и особенно к разнотипным данным.

Примеры:

```
x=false || 2*2==4        // значение переменной x равно true
x=5<2 || "abcd" <= "xy"  // значение переменной x равно true
y=!x                     // значение переменной y равно false
z=x && y                 // значение переменной z равно false
z=x || y                 // значение переменной z равно true
```

Сложные логические выражения, состоящие из нескольких более простых, соединенных операторами И и ИЛИ, выполняются по так называемому *принципу короткой обработки*. Дело в том, что значение всего выражения можно определить, вычислив лишь одно или несколько более простых выражений, не вычисляя остальные. Например, выражение `x&&y` вычисляется слева направо; если значение `x` оказалось равным `false`, то значение `y` не вычисляется, поскольку и так известно, что значение всего выражения равно `false`. Аналогично, если в выражении `x||y` значение `x` равно `true`, то значение `y` не вычисляется, поскольку уже ясно, что все выражение равно `true`. Данное обстоятельство требует особого внимания при тестировании сложных логических выражений. Так, если какая-нибудь составная часть выражения содержит ошибку, то она может остаться невыявленной, поскольку эта часть выражения просто не выполнялась при тестировании.

2.5.6. Операторы условного перехода

Управление вычислительным процессом производится с помощью операторов условного перехода `if`, `switch` и `?:`.

Оператор *if*

Оператор условного перехода `if` позволяет реализовать структуру условного выражения:

если ..., то ..., иначе ...

Синтаксис оператора `if` перехода следующий:

```
if (условие)
  {код, который выполняется, если условие истинно}
```

```
else
  {код, который выполняется, если условие ложно}
```

В фигурных скобках располагается блок кода — несколько выражений. Если в блоке используется не более одного выражения, то фигурные скобки можно не писать. Часть этой конструкции, определяемая ключевым словом `else` (иначе), необязательна. В этом случае остается только часть, определенная ключевым словом `if` (если):

```
if (условие)
  {код, который работает, если условие истинно}
```

Конструкция оператора условного перехода допускает вложение других операторов условного перехода. Условие обычно представляет собой выражение логического типа, т. е. выражение, значение которого есть `true` или `false`. Обычно это элементарные выражения с операторами сравнения.

Примеры:

- выводится диалоговое окно с тем или иным сообщением в зависимости от значения переменной `age` (возраст):

```
if (age<18)
  {alert("Вы слишком молоды для просмотра этого сайта")}
else
  {alert("Подтвердите свое решение заглянуть на этот сайт")}
```

- выводится диалоговое окно с сообщением, если только значение переменной `age` меньше 18:

```
if (age<18)
  {alert("Вы слишком молоды для просмотра этого сайта")}
```

Делать ли отступы при написании операторов, где располагать фигурные скобки — дело вкуса. Следует руководствоваться наглядностью и ясностью структуры, при которой легко проверить правильность расстановки скобок. Кроме описанного выше способа, часто встречается еще и такой:

```
if (условие) {
  код, который работает, если условие выполнено
} else {
    код, который работает, если условие не выполнено
}
```

Возможно, этот способ записи лучше отражает структуру оператора условного перехода.

Более сложная структура оператора условного перехода получается при вложении других операторов `if`:

```
if (условие1) {
  код, который работает, если условие1 выполнено
} else { if (условие2){
    код, который работает, если условие2 выполнено
  }else{
    код, который работает, если условие2 не выполнено
  }
}
```

Ранее уже отмечалось, что условие в операторе `if` обычно является логическим выражением. Однако это может быть также и строковое, и числовое выражение. В случае строкового выражения условие считается выполненным, если его значением является непустая строка. Напомним, что пустая строка `""` не содержит ни одного символа, в том числе и пробела (строка, содержащая хотя бы один пробел, не пуста). В случае числового выражения условие считается выполненным, если его значением является число, отличное от нуля. Во многих случаях эта многозначность типа условия оказывается очень удобной.

Допустим, что переменная `x` содержит данные, которые ввел пользователь, и нам требуется проверить, что он действительно что-то ввел. В следующем примере мы проверяем, что значение переменной `x` не пусто (не 0, не пустая строка `""` и не `null`). Если `x` пусто, то выводится соответствующее сообщение.

```
if (!x){
  alert("Вы ничего не ввели")
}
```

Заметим, что поскольку в этом примере блок кода содержит всего лишь одно выражение, то фигурные скобки можно опустить, да и всю конструкцию оператора условного перехода можно записать в одной строке:

```
if (!x) alert("Вы ничего не ввели")
```

Оператор *switch*

Для организации проверки большого количества условий вполне достаточно использовать рассмотренный выше оператор `if`. Однако в случае нескольких условий более удобным и наглядным оказывается оператор `switch` (переключатель). Он особенно удобен, если требуется проверить несколько условий, которые не являются взаимоисключающими.

Синтаксис оператора `switch` выглядит следующим образом:

```
switch (выражение) {
  case вариант1:
    код
    [break]
  case вариант2:
    код
    [break]
  ...
  [default:
    код]
}
```

Параметр *выражение* оператора `switch` может принимать строковые, числовые и логические значения. Разумеется, в случае логического выражения возможны только два варианта. Ключевые слова (операторы) `break` и `default` не являются обязательными, что отражено с помощью прямоугольных скобок. Здесь они являются элементами описания синтаксиса, и при написании операторов их указывать не нужно. Если оператор `break` указан, то проверка остальных условий не производится. Если указан оператор `default`, то следующий за ним код выполняется, если значение выражения *выражение* не соответствует ни одному из вариантов. Если все варианты возможных значений предусмотрены в операторе `switch`, то оператор `default` можно не использовать.

Оператор `switch` работает следующим образом. Сначала вычисляется *выражение*, указанное в круглых скобках сразу за ключевым словом `switch`. Полученное значение сравнивается с тем, которое указано в первом варианте. Если они не совпадают, то код этого варианта не выполняется и происходит переход к следующему варианту. Если же значения совпали, то выполняется код, соответствующий этому варианту. При этом если не указан оператор `break`, то выполняются коды и остальных вариантов, пока не встретится оператор `break`. Это же правило действует и для остальных вариантов.

Пример:

```
x=2
switch (x) {
  case 1:
    alert(1)
  case 2:
    alert(2)
```

```
  case 3:
    alert(3)
}
```

В приведенном выше примере сработают 2-й и 3-й варианты. Если мы хотим, чтобы сработал только один какой-нибудь вариант (только тот, который соответствует значению выражения), то нужно использовать оператор `break`.

Пример:

```
x=2
switch (x) {
  case 1:
    alert(1);
    break
  case 2:
    alert(2);
    break
  case 3:
    alert(3);
    break
}
```

В этом примере сработает только 2-й вариант.

Пример. Допустим, переменная `xlang` содержит название языка, который выбрал пользователь и ввел в поле формы. Тогда возможен такой вариант применения оператора `switch`:

```
switch (xlang) {
  case "английский":
    window.open("engl.htm");
    break
  case "французский":
    window.open("french.htm");
    break
  case "немецкий":
    window.open("german.htm");
    break
  default:
    alert("У нас нет документа на таком языке")
}
```

Здесь выражение `window.open()` открывает новое окно браузера и загружает в него указанный в скобках HTML-документ. Заметим, что эту же задачу можно решить и с помощью вложенных операторов `if`:

```
if (xlang == "английский")
  window.open("engl.htm")
else {
  if (xlang == "французский")
    window.open("french.htm");
  else {
    if (xlang == "немецкий")
      window.open("german.htm")
    else
      alert("У нас нет документа на таком языке");
  }
}
```

Для тренировки перепишем приведенный выше код в другом виде:

```
if (xlang == "английский") window.open("engl.htm")
else { if (xlang == "французский")
  window.open("french.htm")
  else { if (xlang == "немецкий")
    window.open("german.htm")
    else alert("У нас нет документа на таком языке")
  }
}
```

Оператор условия *?:*

Этот оператор является сокращенной формой оператора условного перехода `if...else...`. Его так и называют: *оператор условия*. Обычно он применяется вместе с оператором присваивания одного из двух возможных значений, в зависимости от значения условного выражения. Синтаксис оператора условия следующий:

условие ? *выражение1* : *выражение2*

С оператором присваивания оператор условия имеет такой вид:

переменная=условие ? *выражение1* : *выражение2*

Оператор условия возвращает значение выражения *выражение1*, если *условие* истинно, в противном случае — значение выражения *выражение2*. Поэтому он может использоваться в составных выражениях.

Пример:

```
d=new Date()   // текущая дата
x=d.getDate()
typedate=(x%2 == 0)&&(x > 1) ? "четное" : "нечетное"
```

Здесь использовался объект даты и его метод `getDate()` для получения числа текущей даты *(см. разд. 2.11)*.

2.5.7. Операторы цикла

Оператор цикла обеспечивает многократное выполнение блока программного кода до тех пор, пока не выполнится некоторое условие. В JavaScript предусмотрены три оператора цикла: `for`, `while` и `do-while`. При создании программ вполне можно обойтись каким-нибудь одним из них, например, `for` или `while`. Однако возникают ситуации, в которых один из операторов более удобен или естественен, чем другой.

Оператор *for*

Оператор `for` (для) также называют *оператором со счетчиком циклов*, хотя в нем совсем не обязательно использовать счетчик. Синтаксис этого оператора следующий:

```
for ([начальное_выражение]; [условие]; [выражение_обновления])
{
  код
}
```

Здесь квадратные скобки лишь указывают на то, что заключенные в них параметры не являются обязательными. Как и в случае оператора условного перехода, возможна и такая запись:

```
for ([начальное_выражение]; [условие]; [выражение_обновления]) {
  код
}
```

Все, что находится в круглых скобках справа от ключевого слова `for`, называется *заголовком* оператора цикла, а содержимое фигурных скобок — его *телом*.

В заголовке оператора цикла начальное выражение обрабатывается только один раз в начале выполнения оператора. Второй параметр представляет собой условие продолжения работы оператора цикла. Он аналогичен условию в операторе `if`. Третий параметр содержит выражение, которое выполняется после выполнения всех выражений кода, заключенного в фигурные скобки.

Оператор цикла работает следующим образом. Сначала выполняется *начальное_выражение*. Затем проверяется *условие*. Если оно ложно, то оператор цикла прекращает работу (при этом *код* не выполняется). В противном случае выполняется *код*, расположенный в теле оператора `for`, т. е. между фигурными скобками. После этого выполняется *выражение_обновления* (третий параметр оператора `for`). Так заканчивается первый цикл или, как еще говорят, первая *итерация* цикла. Далее, снова проверяется *условие*, и все повторяется описанным выше способом.

Обычно в качестве начального выражения используют оператор присваивания значения переменной. Например, `i=0` или `var i=0`. Имя переменной и присваиваемое значение могут быть любыми. Эту переменную называют *счетчиком* цикла. В этом случае условие, как правило, представляет собой элементарное выражение сравнения переменной счетчика цикла с некоторым числом, например, `i<=nMax`. Выражение обновления в таком случае просто изменяет значение счетчика, например, `i=i+1` или, короче, `i++`.

В следующем примере оператор цикла просто изменяет значение своего счетчика, выполняя 15 итераций:

```
for (i=1; i <= 15; i++) {}
```

Немного модифицируем этот код, чтобы вычислить сумму всех целых положительных чисел от 1 до 15:

```
var s=1
for (i=1; i <= 15; i++) {
  s=s + i
}
```

Заметим, что счетчик цикла может быть не только возрастающим, но и убывающим.

- Пример: `x` в степени `y`.

  Допустим, требуется вычислить `x` в степени `y`, где `x`, `y` — целые положительные числа. Алгоритм решения этой задачи прост: надо вычислить выражение `x*x*x*...*x`, в котором сомножитель `x` встречается `y` раз. Очевидно, что если `y` не превышает 10, то можно воспользоваться оператором умножения: выражение будет не очень длинным. А как быть в том случае, когда `y` равно нескольким десяткам или сотням? Ясно, что в общем случае следует применить оператор цикла:

  ```
  /* Вычисляем x в степени y */
  var z=x     // z хранит результат
  for (i=2; i <= y; i++) {
  ```

```
    z=z*x
  }
```

В этом примере результат сохраняется в переменной z. Ее начальное значение равно x. Если y равно 1, то оператор цикла не будет выполняться, поскольку не выполняется условие 2<=1 (обратите внимание на начальное значение счетчика цикла) и, следовательно, мы получим верный результат: x в степени 1 равно x. Если y равно 2, то оператор цикла выполнит одну итерацию, вычислив выражение z=z*x при z, равном x (т. е. z=x*x). При y, равном 3, оператор цикла сделает две итерации. На второй, последней, итерации выражение z=z*x вычисляется при текущем значении z, равном x*x, и, следовательно, становится равным x*x*x (т. е. x в степени 3).

- Пример: n!.

  Рассмотрим довольно традиционный при изучении операторов цикла пример вычисления факториала числа. Факториал числа n в математике обозначают как n!. Для n, равного 0 и 1, n! равен 1. В остальных случаях n! равен 2*3*4*...*n. Поскольку возможны два варианта исходных данных, нам потребуется использовать оператор условного перехода. Код, решающий эту задачу, выглядит следующим образом:

```
/* Вычисляем n! */
var z=1     // z хранит результат n!
if (n >1) {
  for (i=2; i <= n; i++) {
    z=z*i
  }
}
```

Для принудительного (т. е. не по условию) выхода из цикла используется оператор break (прерывание). Если вычислительный процесс встречает этот оператор в теле оператора цикла, то он сразу же завершается без выполнения последующих выражений кода в теле и даже выражения обновления. Обычно оператор break применяется при проверке некоторого дополнительного условия, выполнение которого требует завершения цикла, несмотря на то, что условие в заголовке цикла еще не выполнено. Типовая структура оператора цикла с использованием break имеет следующий вид:

```
for ([начальное_выражение]; [условие1]; [выражение_обновления])
{
  код
  if (условие2){
    код
  break
```

```
  }
  код
}
```

Для управления вычислениями в операторе цикла можно также использовать оператор `continue` (продолжение). Так же, как и `break`, этот оператор применяется в теле оператора цикла вместе с оператором условного перехода. Однако в отличие от `break`, оператор `continue` прекращает выполнение последующего кода, выполняет выражение обновления и возвращает вычислительный процесс в начало оператора цикла, где производится проверка условия, указанного в заголовке. Типовая структура оператора цикла с использованием `continue` имеет следующий вид:

```
for ([начальное_выражение]; [условие1]; [выражение_обновления] )
{
  код
  if (условие2){
    код
    continue
  }
  код
}
```

Оператор *while*

Оператор цикла `while` (до тех пор, пока) имеет структуру, более простую, чем оператор `for`, и работает несколько иначе. Синтаксис этого оператора следующий:

```
while (условие)
{
  код
}
```

При выполнении этого оператора сначала производится проверка условия, указанного в заголовке, т. е. в круглых скобках справа от ключевого слова `while`. Если оно истинно, то выполняется код в теле оператора цикла, заключенного в фигурные скобки. В противном случае код не выполняется. При выполнении кода (завершении первой итерации) вычислительный процесс возвращается к заголовку, где снова проверяется условие, и т. д.

Если сравнивать оператор `while` с оператором `for`, то особенность первого заключается в том, что выражение обновления записывается в теле оператора, а не в заголовке. Часто забывают указать это выражение, и в результате цикл не завершается (программа "зависает").

Рассмотрим несколько примеров задач, которые мы уже решали ранее с использованием оператора цикла `for`.

- Пример: `x` в степени `y`.

```
/* Вычисляем x в степени y */
var z=x     // z хранит результат
i=2
while (i=2) {
  z=z*x
  i++
}
```

- Пример: `n!`.

```
/* Вычисляем n! */
var z=1     // z хранит результат n!
if (n >1) {
  i=2
  while ( i <= n ) {
    z=z*i
    i++
  }
}
```

Во всех приведенных выше примерах использовался счетчик цикла. Однако он инициировался заранее, до оператора цикла. Обновление значения этого счетчика производилось в теле оператора цикла.

Для управления вычислительным процессом в операторе `while`, так же как и в операторе `for`, можно применять операторы прерывания `break` и продолжения `continue`.

Внимание

Если в `while` вы используете счетчики циклов, то будьте осторожны, применяя `break` и `continue`.

Оператор *do-while*

Оператор `do-while` (делай до тех пор, пока) представляет собой конструкцию из двух операторов, используемых совместно. Синтаксис этой конструкции следующий:

```
do {
  код
}
while (условие)
```

В отличие от оператора `while` в операторе `do-while` код выполняется хотя бы один раз, независимо от условия. Условие проверяется после выполнения кода. Если оно истинно, то снова выполняется код в теле оператора `do`. В противном случае работа оператора `do-while` завершается. В операторе `while` условие проверяется в первую очередь, до выполнения кода в теле. Если при первом обращении к оператору `while` условие ложно, то код не будет выполнен никогда.

Рассмотрим примеры задач, которые мы уже решали ранее с использованием операторов цикла `for` и `while`.

- Пример: `x` в степени `y`.

```
/* Вычисляем x в степени y */
var z=x     // z хранит результат
i=2
do{
  z=z*x
  i++
}
while (i <= y)
```

- Пример: `n!`.

```
/* Вычисляем n! */
var z=1     // z хранит результат n!
if (n >1) {
  i=2
  do {
    z=z*i
    i++
  }
  while ( i <= n )
}
```

2.5.8. Выражения с операторами

Просто данные (конечные значения) являются выражениями языка. Например, число `25`, одиноко стоящее в строке текста программного кода, является выражением. Последовательность символов, заключенная в кавычки (например, `"Добро пожаловать!"`), — тоже выражение. Имя переменной — еще один вариант выражения.

Запись, содержащая имя переменной, за которой следуют символы оператора присваивания и некоторое значение (например, `x="Привет всем!"`), является выражением JavaScript. Запись, состоящая из операндов и оператора (например, `x+5`), также является выражением.

Все перечисленные выше варианты выражения назовем элементарными выражениями. Тогда записи, содержащие операторы и операнды в виде элементарных выражений, являются выражениями. Например, пусть имеются два элементарных выражения: `x+5` и `y-3`. Тогда запись `x+5 + y-3`, объединяющая их оператором сложения, является выражением. Это касается не только арифметических операторов, но и всех других. Аналогично выражения, не являющиеся элементарными, можно объединять с помощью операторов в более сложные выражения.

Итак, мы можем писать сложные выражения, в которых операторы различных типов могут встречаться несколько раз. Выражение с последовательностью из нескольких операторов вычисляется слева направо, но с учетом так называемого *приоритета операторов*.

Внимание

Среди арифметических операторов наибольшим приоритетом обладают операторы умножения и деления (в том числе и деления по модулю). Затем следуют сложение и вычитание. Среди логических операторов наибольшим приоритетом обладает отрицание, затем следует логическое И, а наименьшим — логическое ИЛИ. Операторы сравнения по приоритету выше логических операторов. Последовательность операторов с одинаковым приоритетом выполняется слева направо.

Примеры:

```
2 + 3 * 5                  // результат равен 17, а не 25
2 < 3 || 3 < 1             // результат равен true
2 < 3 || 3 < 1 && false    // результат равен true
!2 < 3 || 3 < 1 && false   // результат равен false
```

Процессом вычисления выражений можно управлять с помощью круглых скобок. Каждой открывающей скобке соответствует своя закрывающая скобка, так что выражения, заключенные в круглые скобки, могут входить в состав других выражений, которые в свою очередь также могут быть заключены в скобки. Таким образом, с помощью круглых скобок можно создать иерархическую структуру выражения, состоящего из других выражений. Первыми выполняются выражения, имеющие наибольшую глубину вложенности. Далее выполняются выражения, находящиеся на меньшей глубине в иерархии, и т. д. Выражения с одинаковой глубиной вложенности выполняются в порядке, принятом по умолчанию, т. е. слева направо.

Внимание

В выражении с круглыми скобками количество открывающих скобок должно быть равно количеству закрывающих, а общее количество всех круглых скобок должно быть четным числом. Если это не выполняется, то интерпретатор выдает сообщение о синтаксической ошибке. Ошибки такого рода часто встречаются при программировании.

Примеры:

```
  2 + 3 * 5           // результат равен 17
 (2 + 3) * 5          // результат равен 25
((2 + 3) + 4*5)/2     // результат равен 12.5
 (2 + 3) + 4*5/2      // результат равен 15
  2 + 3 + 4*5/2       // результат равен 15
 (2 + 3 + 4)*5/2      // результат равен 22.5
  2 + (3 + 4*5)/2     // результат равен 13.5
 (2 + (3 + 4)*5)/2    // результат равен 18.5
```

Операторы условного перехода цикла также представляют собой выражения языка JavaScript. Кроме обычных способов их записи, рассмотренных выше, принципиально возможна запись в одну строку. При этом необходимо, чтобы выражения в блоках кода разделялись точкой с запятой.

Пример:

```
if (!x){x="Вы ничего не ввели"; alert(x)}else alert("Все в порядке")
```

Операторы условного перехода и цикла возвращают значения подобно другим операторам, а именно значение последнего выполненного выражения.

2.6. Функции

Функция представляет собой подпрограмму, которую можно вызвать для выполнения, обратившись к ней по имени. Взаимодействие функции с внешней программой, из которой она была вызвана, происходит путем передачи функции параметров и приема от нее результата вычислений. Впрочем, функция в JavaScript может и не требовать параметров, а также ничего не возвращать.

В JavaScript есть *встроенные функции*, которые можно использовать в программах, но код которых нельзя редактировать или посмотреть. Все, что мы можем узнать о них, — это описание их действия, параметров и возвращаемого значения.

Кроме использования встроенных функций, вы можете создать собственные, так называемые *пользовательские функции*. Часто используемые фрагменты программного кода целесообразно оформлять в виде функций. Такой фрагмент кода заключается в фигурные скобки, а перед ним пишется ключевое слово `function`, за которым следуют круглые скобки, обрамляющие список параметров. Более подробно пользовательские функции будут рассмотрены в *разд. 2.6.2*.

Чтобы вызвать функцию в программе, следует написать выражение в следующем формате:

имя_функции(*параметры*)

Если требуются параметры, то они указываются в круглых скобках через запятую. Функция может и не иметь параметров. В этом случае в круглых скобках ничего не указывается.

2.6.1. Встроенные функции

В JavaScript имеются несколько встроенных функций, которые перечислены далее. Некоторые из них мы уже рассматривали ранее.

- `parseInt(`*строка*`|`*число*`, `*основание*`)` — преобразует указанную строку или число в целое число. Второй параметр указывает основание системы счисления (2, 8, 10 или 16), в которой представлено число в строке. Если основание не задано, то предполагается 10, т. е. десятичная система счисления.

  Примеры:

```
parseInt("2.5")            // результат=2
parseInt("-17.875")        // результат=-17
parseInt("1952")           // результат=1952
parseInt("150 руб.")       // результат=150
parseInt("цена 150 руб.")  // результат=NaN
parseInt("")                // результат=NaN

parseInt("ff", 16)         // результат=255
parseInt("0xff", 16)       // результат=255
parseInt("0xF0F0FA", 16)   // результат=15790330
parseInt("12", 8)          // результат=10
parseInt("1101", 2)        // результат=13
parseInt(1101, 2)          // результат=13
```

Для правильного преобразования шестнадцатеричного числа, последнее должно быть представлено как данные строкового типа.

- `parseFloat(`*`строка|число, основание`*`)` — преобразует указанную строку или число в число с плавающей разделительной (десятичной) точкой. Второй параметр указывает основание системы счисления (8, 10 или 16), в которой представлено число в строке. Если основание не задано, то предполагается 10, т. е. десятичная система счисления.

  Примеры:

```
parseFloat("2.5")          // результат=2.5
parseFloat("-17.875")      // результат=-17.875
parseFloat ("1952")        // результат=1952
pareseFloat("1.50px")      // результат=1.5
pareseFloat("двести")      // результат=NaN
parseFloat(0xF0F0FA, 16)   // результат=15790330
```

  Для правильного преобразования шестнадцатеричного числа, последнее не должно быть представлено как данные строкового типа, в отличие от функции `parseInt()`.

- `isNaN(`*`значение`*`)` — возвращает `true`, если указанное в параметре значение не является числом, иначе — `false`.

  Примеры:

```
isNaN(123)                 // результат false
isNaN("123")               // результат false
isNaN("тел. 1234567")      // результат true
isNaN("35 px")             // результат true
isNaN(true)                // результат false
isNaN(false)               // результат false
isNaN("Вася")              // результат true
```

- `eval(`*`строка`*`)` — вычисляет выражение в указанной строке. Выражение должно быть написано на языке JavaScript (не содержит тегов HTML).

  Примеры:

```
x=5          // значение x равно 5
y="x"        // значение y равно "x"
z=eval(y)    // значение z равно 5
```

  В данном примере переменная `y` содержит имя переменной `x`, а функция `eval(y)` возвращает значение переменной `x`. Так в JavaScript реализуется понятие "переменная".

Примечание

В PHP подобное можно сделать с помощью двойного символа `$` *(см. разд. 4.4.3)*.

- ```
  y=5                          // значение y равно 5
  var x="if(y<10) {y=y+2}"   // значение x равно строке символов
  eval(x)                      // значение y равно 7
  ```

  Как видите, в переменной можно сохранить выражение с оператором, которое затем выполняется с помощью функции `eval()`.

- Web-страница, содержащая текстовые области для ввода и отображения результатов выполнения выражений JavaScript. Это простой редактор кода на языке JavaScript. Код, введенный в верхнюю текстовую область, выполняется с помощью функции `eval()`, а результат отображается в нижней текстовой области. Соответствующий HTML-код со сценарием приведен в листинге 2.2.

Листинг 2.2. Редактор JavaScript

```
<HTML>
  <HEAD><TITLE>Редактор JavaScript</TITLE></HEAD>
  <TEXTAREA ID="mycode" ROWS=10 COLS=60></TEXTAREA>
  <TEXTAREA ID="myrezult" ROWS=3 COLS=60></TEXTAREA>
  <P>
  <BUTTON onclick="document.all.myrezult.value=eval(mycode.value)">
    Выполнить
  </BUTTON>
  <BUTTON onclick="document.all.mycode.value='';
                   document.all.myrezult.value=''">
    Очистить
  </ BUTTON >
</HTML>
```

Здесь сценарии записаны в виде символьных строк в качестве значений атрибутов `onclik`, определяющих событие "Щелчок кнопкой мыши" для кнопок, которые заданы тегами `&lt;BUTTON&gt;`. На рис. 2.2 показан вид приведенного выше HTML-документа в браузере.
```

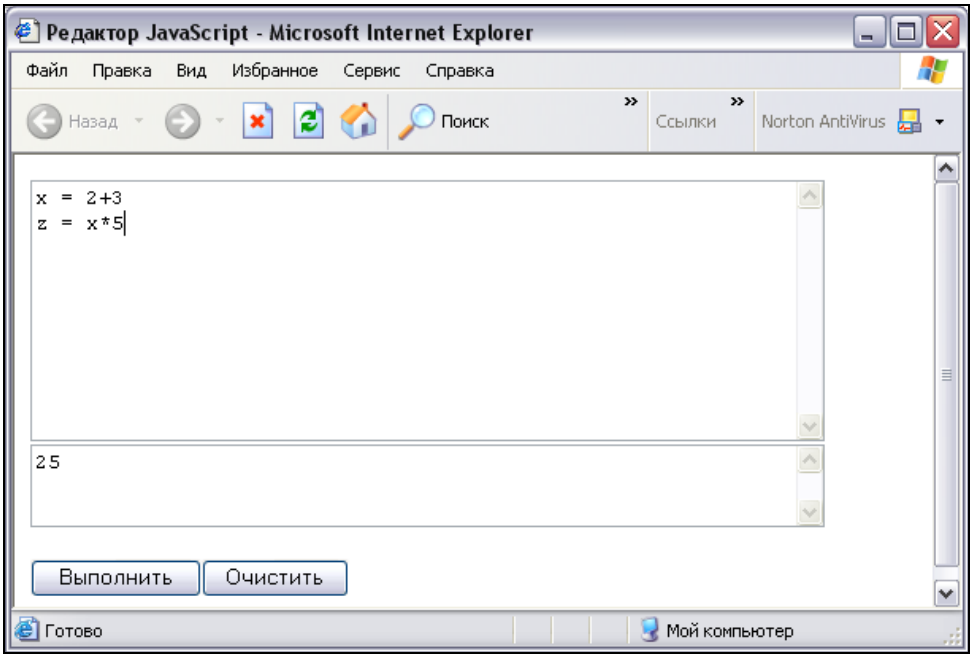


Рис. 2.2. Редактор кодов JavaScript, созданный с помощью функции `eval()`

- `escape(`*строка*`)` — возвращает строку в виде `%`*XX*, где *XX* — код указанного символа; такую строку еще называют escape-последовательностью.

  При взаимодействии браузеров и серверов протоколы передачи данных позволяют передавать не все символы в их естественном виде. Для передачи остальных символов используются их коды, перед которыми указывается символ `%`. Например, пробел представляется в escape-последовательности как `%20`. Так, например, при передаче данных формы HTML-документа методом `GET` в адресной строке браузера можно видеть escape-последовательность, начинающуюся после адреса за символом `?` *(см. разд. 1.2.7)*.

  Примеры:

```
escape("How do you do")     /* значение равно "How%20do%20you%20do" */
escape("Привет")            /* значение равно
                               %u041F%u0440%u0438%u0432%u0435%u0442 */
```

- `unescape(`*строка*`)` — осуществляет обратное преобразование.
- `typeOf(`*объект*`)` — возвращает тип указанного объекта в виде символьной строки; например, `"boolean"`, `"function"` и т. п.

2.6.2. Пользовательские функции

Пользовательские функции — это функции, которые вы можете создать сами, по своему усмотрению, для решения собственных задач. Функция задается своим определением (описанием), которое начинается ключевым словом `function`. Точнее, описание функции имеет следующий синтаксис:

```
function имя_функции(параметры){
  код
}
```

Имя функции выбирается так же, как и имя переменной. Недопустимо использовать в качестве имени ключевые слова языка JavaScript. За именем функции обязательно стоит пара круглых скобок. Программный код (тело) функции заключается в фигурные скобки. Они определяют группу выражений, которые относятся к коду именно этой функции. Если функция принимает параметры, то список их имен (идентификаторов) указывается в круглых скобках около имени функции. Имена параметров выбираются согласно тем же требованиям, что и имена обычных переменных. Если параметров несколько, то в списке они разделяются запятыми. Если параметры для данной функции не предусмотрены, то в круглых скобках около имени функции ничего не пишут.

Когда создается определение функции, список ее параметров (если он необходим) содержит просто формальные идентификаторы (имена) этих параметров, понимаемые как переменные. В определении функции в списке параметров, заключенном в круглые скобки сразу же за именем функции после ключевого слова `function`, нельзя использовать конкретные значения и выражения. В этом смысле определение функции задает код, оперирующий формальными параметрами, которые конкретизируются лишь при вызове функции из внешней программы.

Если требуется, чтобы функция возвращала некоторое значение, то в ее теле используется оператор возврата `return` с указанием справа от него того, что следует возвратить. В качестве возвращаемой величины может выступать любое выражение: простое значение, имя переменной или вычисляемое выражение. Оператор `return` может встречаться в коде функции несколько раз. Впрочем, возвращаемую величину, а также сам оператор `return` можно и не указывать. В этом случае функция ничего не будет возвращать.

Пример. Допустим, требуется определить функцию для вычисления площади прямоугольника (для этого необходимо умножить ширину на высоту). Назовем эту функцию `Srectangle`. Тогда ее определение будет выглядеть следующим образом:

```
function Srectangle(width, height){
  S=width * height
```

```
  return S
}
```

Здесь сначала вычисляется площадь путем умножения значений параметров `width` и `height`, полученное значение присваивается переменной `S`, а затем оператор `return` возвращает значение этой переменной.

Определение функции `Srectangle` можно сделать более экономным, минуя присвоение значения переменной `S`:

```
function Srectangle(width, height){
  return width * height
}
```

Определение функции, описанное выше, будучи помещенным в программу, усваивается интерпретатором, но сама функция (ее код или тело) не выполняется. Чтобы выполнить функцию, определение которой задано, необходимо написать в программе выражение вызова этой функции. Оно имеет следующий синтаксис:

имя_функции(*параметры*)

Имя функции должно полностью (вплоть до регистра) совпадать с именем ранее определенной функции. Параметры, если они заданы в определении функции, в вызове функции представляются конкретными значениями, переменными или выражениями.

Внимание

Не путайте определение функции с ее вызовом, хотя и то и другое может находиться в одной и той же программе.

В JavaScript можно не поддерживать равенство между количествами параметров в определении функции и в ее вызове. Если в функции определены, например, три параметра, а в вызове указаны только два, то последнему параметру будет автоматически присвоено значение `null`. Наоборот, лишние параметры в вызове функции будут просто проигнорированы.

Ранее мы рассмотрели пример определения функции `Srectangle()`, вычисляющей площадь прямоугольника, если заданы его ширина и высота. Чтобы вычислить значение площади прямоугольника при конкретных значениях параметров, в программе необходимо написать выражение вызова функции `Srectangle()`, указав в качестве ее параметров конкретные значения, либо переменные с конкретными значениями, либо выражения, вычисляющие значения. Далее приведены несколько возможных вариантов вызова функции `Srectangle`.

Примеры:

```
Srectangle(3, 5)    /* возвращает площадь прямоугольника размером  3*5 */
Srectangle(3, 4 + 2) /* возвращает площадь прямоугольника размером 3*6 */

height=8
Srectangle(3, height) /* возвращает площадь прямоугольника
                         размером 3*8 */

width=4
height=5
Srectangle(widt, height + 2) /* возвращает площадь прямоугольника
                                размером 4*7 */

Srectangle(2) /* возвращает 0 (второй параметр не задан) */
```

Внутри тела функции можно создавать переменные либо просто с помощью оператора присваивания, либо с помощью ключевого слова `var`. При этом возникает вопрос об области действия переменных.

Если вы используете просто оператор присваивания, то возможны две ситуации.

- Переменная в операторе присваивания встречается первый раз в вашей программе именно в теле функции. В этом случае она действует в пределах кода (тела) данной функции, но после выполнения функции эта переменная продолжает существовать и во внешней программе. Таким образом, эта переменная является *глобальной*.
- Переменная в операторе присваивания уже определена во внешней программе. В этом случае она действует также внутри функции и продолжает существовать после завершения ее выполнения, поскольку она была создана во внешней программе как глобальная переменная.

Если в теле функции вы используете ключевое слово `var` для инициализации переменной, то эта переменная будет действовать только в пределах кода функции, независимо от того, была ли она определена во внешней программе или нет. При этом если переменная создана с ключевым словом `var` впервые в теле функции, то она является *локальной*, т. е. недоступной из внешней программы. Таким образом, инициализация переменной с помощью выражения с ключевым словом `var` создает локальную переменную.

Если в теле функции вы используете переменную, определенную только во внешней программе, то изменение ее значения в теле функции сохранится даже после завершения выполнения этой функции, поскольку это глобальная переменная.

Если во внешней программе вы определили некоторые переменные, имена которых совпадают с формальными параметрами в определении функции, а затем указываете их в качестве параметров вызова этой функции, то произойдет следующее. Функция воспримет значения параметров, определенных во внешней программе, однако любые изменения их значений в теле функции останутся локальными, т. е. после завершения работы кода функции значения указанных переменных останутся прежними, как до вызова функции.

Рассмотрим типичные ситуации, связанные с областью действия переменных, на следующих примерах.

- Переменная `S` определена только в теле функции, но продолжает существовать во внешней программе:

```
function Srectangle(width, height){
  S=width * height
  return S
}
z=Srectangle(2, 3)  /* значения и z, и S равны 6;
                       S — глобальная переменная */
```

- Переменная `S` определена во внешней программе, но используется в теле функции:

```
function Srectangle(width, height){
  S=width * height
  return S
}
S=2
z=Srectangle(2, 3)  /* значение z равно 6, а значение S осталось
                       равным 2, т. е. значению глобальной переменной,
                       определенной во внешней программе */
```

- То же, что и в предыдущем примере, но с использованием ключевого слова `var`:

```
function Srectangle(width, height){
  var S=width * height
  return S
}
var S =2
z=Srectangle(2, 3)  /* значение z равно 6, а значение S осталось
                       равным 2, т. е. значению глобальной переменной,
                       определенной во внешней программе */
```

- Исключительно локальные переменные:

```
function Srectangle(width, height){
  var S=width * height
  var x =17
  return S
}
z=Srectangle(2,3)  /* значение z равно 6;
                      переменная S во внешней программе не определена;
                      переменная x во внешней программе не определена */
```

- Имена параметров в определении функции совпадают с именами параметров во внешней программе и с параметрами в вызове функции:

```
function Srectangle(width, height){
  var s=width * height
  width=width + 10    // изменяем значение параметра width
  return s
}
width=2
height=3
z=Srectangle(width, height)  /* значение z равно 6;
                                значение переменной width равно 2,
                                т. е. осталось без изменений */
```

Программа может содержать и определение функции, и выражения ее вызова. При этом порядок их следования в программе не важен: вы можете сначала написать определение функции, а затем где-нибудь в программе привести ее вызов или, наоборот, написать сначала вызов функции, а ее определение разместить где-нибудь в конце программы или даже в отдельном файле с расширением js. В приведенных выше примерах мы записывали определение функции до ее вызова, однако программисты чаще поступают наоборот.

- Пример: функция вычисления факториала `n!` с помощью оператора цикла.

  В *разд. 2.5.7*, посвященном операторам цикла, мы рассматривали программный код вычисления факториала числа `n`. Напомним, что факториал целого положительного числа `n` в математике обозначают как `n!` и вычисляют по формуле: `1*2*3*...*n`, если `n<=1`. Для `n=0` и `n=1`: `n!=1`. В этом примере программу вычисления `n!` мы оформим в виде функции:

```
function factorial(n){
  if(n <= 1) return 1
  rezult=2     // result — переменная для результата
```

```
  for (i=3; i <=n; i++) {
    rezult=rezult*i
  }
  return result
}
```

Для вычисления факториала конкретного числа, например 12, следует записать выражение `factorial(12)`.

Если в нашей программе определена описанная выше функция `factorial()`, то в этой же программе возможен такой фрагмент, содержащий вызов этой функции:

```
var m=10
x=factorial(m)   // значение x равно 3628800
```

Определение функции может содержать вызов этой же функции — так называемое *рекурсивное определение функции*. В качестве примера рекурсивной функции в учебной литературе обычно приводят функцию вычисления факториала.

- Пример: функция вычисления факториала `n!` с помощью рекурсии.

```
function factorial(n){
  if(n <= 1){
    return 1
  }
  return n*factorial(n-1) // вызов функции factorial()
}
```

Отметим, что определение этой функции можно записать более компактно, опустив пару фигурных скобок, поскольку заключенный в них блок кода содержит лишь одно выражение:

```
function factorial(n){
  if(n <= 1) return 1
  return n*factorial(n-1) // вызов функции factorial()
}
```

Совет

Чтобы избежать ошибок, старайтесь не использовать в коде функций имена переменных, которые были инициализированы во внешней программе. По возможности стремитесь к тому, чтобы все, что делается внутри вашей функции, было локальным, формально не зависимым от внешнего окружения.

Определение функции может содержать в себе определения других функций, однако такие вложенные функции доступны только из кода функции, содержащей их определения.

Определение функции создает так называемый экземпляр объекта `Function`, обладающий полезными свойствами и методами. В частности, он имеет свойство `arguments`, содержащее информацию о параметрах, которые действительно были переданы функции при ее вызове. В некоторых случаях, прежде чем выполнить выражения тела функции, требуется проанализировать значения параметров. И тогда понадобится использовать свойство `arguments`.

2.6.3. Выражения с функциями

В *разд. 2.5.8* мы рассматривали выражения с операторами. К множеству элементарных выражений JavaScript можно добавить еще и вызовы функций. Вызов функции может быть правой частью оператора присваивания, а также составной частью любого выражения с операторами.

Примеры.

- Пусть `Srectangle(width, height)` — функция, возвращающая значение площади прямоугольника со сторонами `width` и `height`. Тогда для вычисления площади прямоугольного треугольника с катетами `a` и `b` можно использовать следующее выражение:

  ```
  S3=0.5*Srectangle(a, b)
  ```

  В этом примере вызов функции является операндом выражения с арифметическим оператором умножения. При этом значение выражения присваивается переменной `S3`.

- Вызов функции может использоваться в логических выражениях:

  ```
  if (Srectangle(a, b) > Srectangle(c, d))
    alert("Первый прямоугольник больше второго")
  ```

- В качестве параметра функции может указываться вызов другой функции:

  ```
  var x="25px"
  var y=12
  var S=Srectangle(parseInt(x), y)
  ```

В выражениях с вызовами функций последние имеют наивысший приоритет.

2.7. Строки

Обработка символьных строк производится методами встроенного объекта `String` (строка). Он полезен, когда требуется, например, найти позицию вхождения одной строки в другую, вырезать из строки некоторую ее часть, разбить строку на отдельные элементы и создать из них массив и т. д.

С помощью объекта `String` можно создать строку как строковый объект. Однако в подавляющем большинстве случаев для этого достаточно использовать обычную переменную и оператор присваивания строкового значения. При этом интерпретатор все равно создает экземпляр (копию) строкового объекта, свойства и методы которого доступны из программного кода.

2.7.1. Создание строкового объекта

Для создания строкового объекта используется выражение следующего вида:

имя_переменной=new String("*строковое_значение*")

Здесь *имя_переменной* играет роль ссылки на строковый объект. Например, выражение

```
mystring=new String("Привет!")
```

создает строковый объект `mystring` со значением `"Привет!"`.

Однако можно создать строковый объект и с помощью обычного оператора присваивания:

имя_переменной="*строковое_значение*"

или

var *имя_переменной*="*строковое_значение*"

Доступ к свойствам и методам строкового объекта обеспечивается такими выражениями:

строка.*свойство*

String.*свойство*

строка.*метод*([*параметры*])

String.*метод*([*параметры*])

Некоторые методы могут и не иметь параметров, что указано с помощью квадратных скобок. Здесь *строка* может быть ссылкой на строковый объект, строковой переменной, выражением, возвращающим строку, а также просто строковым значением.

Когда используется ключевое слово `String` в качестве имени объекта, это означает, что нас интересуют свойства и методы статического строкового объекта, т. е. общие свойства и методы, не связанные, вообще говоря, с конкретными свойствами и методами конкретного строкового объекта (экземпляра объекта `String`).

Пример. Три различных способа использования свойства `length` строкового объекта, значением которого является длина строки (количество символов в строке).

```
mystring="Привет!"
mystring.length   // значение равно 7

"Привет!".length  // значение равно 7

function fstring(){return "abcde"}// функция, возвращающая строку "abcde"
fstring().length                  // значение равно 5
```

Методы строкового объекта используются для синтаксической обработки и форматирования строк. Эти две группы методов мы рассмотрим отдельно.

2.7.2. Свойства объекта *String*

Свойства объекта `String` описаны далее.

- `length` — длина или, иными словами, количество символов (включая пробелы) в строке; целое число.

  Пример:

  ```
  "Иван".length           // значение равно 4
  "Привет\nвсем".length   /* значение равно 11 (\n — один символ
                             перевода строки) */
  x=""                 // пустая строка
  x.length             // значение равно 0 (пустая строка имеет длину 0)
  ```

- `prototype` — свойство (прототип), позволяющее добавить новые свойства и методы ко всем создаваемым строковым объектам, если уже существующих вам окажется не достаточно.

  Пример. Создадим новый метод для всех строковых объектов. Содержание этого метода определяется пользовательской функцией.

  ```
  function myFunc() {     // функция для нового метода
    return "Вадим Дунаев"
  }
  ```

```
// Добавление нового метода myName к прототипу:
String.prototype.myName=myFunc
mystring = "Автор этой книги — "
mystring = mystring + mystring.myName()
// значение mystring равно "Автор этой книги — Вадим Дунаев"
```

2.7.3. Методы объекта *String* обработки строк

Перечислим методы объекта `String` для обработки строк.

- `charAt(`*индекс*`)` — возвращает символ, занимающий в строке указанную позицию. Синтаксис:

  строка`.charAt(`*индекс*`)`

  Возвращает односимвольную или пустую строку.

  Параметр *индекс* является числом, индекс первого символа равен 0.

  Примеры:

  ```
  "Привет".charAt(2)        // значение равно "и"
  "Привет".charAt(15)       // значение равно ""
  mystring="Привет"
  mystring.charAt(mystring.length-1) /* значение последнего символа
                                        равно "т" */
  ```

- `charCodeAt([`*индекс*`])` — преобразует символ в указанной позиции строки в его числовой эквивалент (код). Синтаксис:

  строка`.charCodeAt([`*индекс*`])`

  Возвращает число. IE4+ и NN6 поддерживают систему кодов Unicode, NN4 — ISO-Latin1.

  Примеры:

  ```
  "abc".charCodeAt()        // значение равно 97
  "abc".charCodeAt(1)       // значение равно 98
  "abc".charCodeAt(25)      // значение равно NaN
  "".charCodeAt(25)         // значение равно NaN
  "я".charCodeAt(0)         // значение равно 1103
  ```

- `fromCharCode(`*номер1*`[, `*номер2*`[, ... `*номерN*`]])` — возвращает строку символов, числовые коды которой указаны в качестве параметров. Синтаксис:

  `String.fromCharCode(`*номер1*`[, `*номер2*`[, ... `*номерN*`]])`

Возвращает строку. IE4+ и NN6 поддерживают систему кодов Unicode, NN4 — ISO-Latin1.

Пример:

```
String. fromcharCode(97,98,1102) // значение равно "абю"
```

- `concat(`*`строка`*`)` — конкатенация (склейка) строк. Синтаксис:

  `строка1``.concat(`*`строка2`*`)`

  Возвращает строку. Этот метод действует так же, как и оператор сложения `+` для строк: к строке *`строка1`* приписывается справа *`строка2`*.

  Примеры:

```
"Иван".concat("Иванов")   // значение равно "ИванИванов"
x="Федор" + " "           // значение x равно "Федор "
x.concat("Иванов")        // значение равно "Федор Иванов"
```

- `indexOf(`*`строка_поиска`*`[, `*`индекс`*`])` — производит поиск строки, указанной параметром, и возвращает индекс ее первого вхождения. Синтаксис:

  `строка``.indexOf(`*`строка_поиска`*`[, `*`индекс`*`])`

  Метод производит поиск позиции первого вхождения *`строка_поиска`* в строку *`строка`*. Возвращаемое число (индекс вхождения) отсчитывается от 0. Если поиск не удачен, то возвращается –1. Поиск в пустой строке всегда возвращает –1. Поиск пустой строки всегда возвращает 0.

  Второй параметр, не являющийся обязательным, указывает индекс, с которого следует начать поиск.

  Этот метод хорошо использовать вместе с методом выделения подстроки `substr()` *(см. далее)*, когда сперва требуется определить позиции начала и конца выделяемой подстроки. Рассматриваемый здесь метод подходит для определения начальной позиции.

  Примеры:

```
x ="Во первых строках своего письма"
x.indexOf("первых")           //значение равно 3
x.indexOf("первых строках")   //значение равно 3
x.indexOf("вторых строках")   //значение равно -1
x.indexOf("В")                //значение равно 0
x.indexOf("в")                //значение равно 6
x.indexOf("в", 7)             //значение равно 19
x.indexOf(" ")                //значение равно 2
x.indexOf(" ", 5)             //значение равно 9
x.indexOf("")                 //значение равно 0
```

- `lastIndexOf(`*`строка_поиска`*`[, `*`индекс`*`])` — производит поиск строки, указанной параметром, и возвращает индекс ее первого вхождения; при этом поиск начинается с конца исходной строки, но возвращаемый индекс отсчитывается от ее начала, т. е. от 0. Синтаксис:

  `строка``.lastIndexOf(`*`строка_поиска`*`[, `*`индекс`*`])`

  Возвращает число.

  Метод аналогичен рассмотренному выше `indexOf()` и отличается лишь направлением поиска.

  Примеры:

  ```
  x ="Во первых строках своего письма"
  x.lastIndexOf("первых")    // значение равно 3
  x.lastIndexOf("a")         // значение равно 30
  x.indexOf("a")             // значение равно 15
  ```

- `localeCompare(`*`строка`*`)` — позволяет сравнивать строки в кодировке Unicode, т. е. с учетом используемого браузером языка общения с пользователем. Синтаксис:

  `строка1``.localeCompare(`*`строка2`*`)`

  Возвращает число. Совместимость: IE5.5+, NN6+.

  Если сравниваемые строки одинаковы, метод возвращает 0. Если *`строка1`* меньше, чем *`строка2`*, то возвращается отрицательное число, в противном случае — положительное. Сравнение строк происходит путем сравнения сумм кодов их символов. Абсолютное значение возвращаемого числа зависит от браузера. Так, IE5.5 и IE6.0 возвращают 1 или –1, а NN6 — разность сумм кодов в кодировке Unicode.

- `slice(`*`индекс1`*`[, `*`индекс2`*`])` — возвращает подстроку исходной строки, начальный и конечный индексы которой указываются параметрами, за исключением последнего символа. Синтаксис:

  `строка``.slice(`*`индекс1`*`[, `*`индекс2`*`])`

  Данный метод не изменяет исходную строку.

  Если второй параметр не указан, то возвращается подстрока с начальной позицией *`индекс1`* и до конца строки. Отсчет позиций начинается с начала строки. Первый символ строки имеет индекс 0. Если второй параметр указан, то возвращается подстрока исходной строки, начиная с позиции *`индекс1`* и до позиции *`индекс2`*, исключая последний символ. Если второй параметр отрицателен, то отсчет конечного индекса производится от конца строки. В этом заключается основное отличие метода `slice()` от `substr()`. Сравните также этот метод с методом `substring()`.

Примеры:

```
x="Во первых строках своего письма"
x.slice(3,8)    // значение равно "первы"
x.slice(3,-2)   // значение равно "первых строках своего пись"

/* Анализ адреса электронной почты */
x="mumu@gerasim.ru"
i=x.indexOf("@")                    // значение равно 4
_name=x.slice(0, i)                 // значение равно "mumu"
_domen=x.slice(i+1, x.indexOf("."))  /* значение равно "gerasim" */
_domen=x.slice(i+1, -3)             // значение равно "gerasim"
```

- `split(`*разделитель*`[, `*ограничитель*`])` — возвращает массив элементов, полученных из исходной строки. Синтаксис:

  строка`.split(`*разделитель*`[, `*ограничитель*`])`

  Первый параметр является строкой символов, используемой в качестве разделителя строки на элементы. Второй необязательный параметр — число, указывающее, сколько элементов строки, полученной при разделении, следует включить в возвращаемый массив. Если разделитель — пустая строка, то возвращается массив символов строки.

  Примеры:

```
x="Привет всем"
x.split(" ")    /* значение — массив из элементов: "Привет", "всем" */
x.split("е")    /* значение — массив из элементов: "Прив", "т вс",
                                                    "м" */
x.split("е",2)  /* значение — массив из элементов: "Прив", "т вс" */
```

- `substr(`*индекс*`[, `*длина*`])` — возвращает подстроку исходной строки, начальный индекс и длина которой указываются параметрами. Синтаксис:

  строка`.substr(`*индекс*`[, `*длина*`])`

  Данный метод не изменяет исходную строку.

  Если второй параметр не указан, то возвращается подстрока с начальной позицией *индекс1* и до конца строки. Отсчет позиций выполняется с начала строки. Первый символ строки имеет индекс 0. Если второй параметр указан, то возвращается подстрока исходной строки, начиная с позиции *индекс1* и с общим количеством символов, равным *длина*. Сравните этот метод с методами `slice()` и `substring()`.

Примеры:

```
x="Привет всем"
x.substr(7,4)          // значение равно "всем"

/* Анализ адреса электронной почты */
x="mumu@gerasim.ru"
i=x.indexOf("@")       // значение равно 4
_name=x.substr(0, i)   // значение равно "mumu"
_domen=x.substr(i+1)   // значение равно "gerasim.ru"
```

- `substring(`*индекс1*`, `*индекс2*`)` — возвращает подстроку исходной строки, начальный и конечный индексы которой указываются параметрами. Синтаксис:

  строка`.substring(`*индекс1* `, `*индекс2*`)`

  Данный метод не изменяет исходную строку.

  Порядок индексов не важен: наименьший из них считается начальным. Отсчет позиций выполняется с начала строки. Первый символ строки имеет индекс 0. Символ, соответствующий конечному индексу, не включается в возвращаемую строку. Сравните этот метод с методами `substr()` и `slice()`.

  Примеры:

```
x="Привет всем"
x.substring(0,6)          // значение равно "Привет"
x.substring(7, x.length)  // значение равно "всем"
x.substring(7, 250)       // значение равно "всем"
x.substring(250, 7)       // значение равно "всем"
```

- `toLocaleLowerCase()`, `toLowerCase()` — переводят строку в нижний регистр. Синтаксис:

  строка`.toLocaleLowerCase()`

  строка`.toLowerCase()`

  Возвращают строку. Первый метод работает в IE5.5+, NN6, учитывает определенные языковые системы.

  Приведение строк к одному и тому же регистру требуется, например, при сравнении содержимого строк без учета регистра. Кроме того, многие серверы чувствительны к регистру, в котором определены имена файлов и папки (обычно требуется, чтобы они были определены в нижнем регистре).

Примеры:

```
x="ЗдраВствуйТе"
x.toLocaleLowerCase()              // значение равно "здравствуйте"
x.toLowerCase()                    // значение равно "здравствуйте"
y="Здравствуйте"
x == y                             // значение равно false
x.toLowerCase() == y.toLowerCase() // значение равно true
```

- `toLocaleUpperCase()`, `toUpperCase()` — переводят строку в верхний регистр. Синтаксис:

  строка`.toLocaleUpperCase()`

  строка`.toUpperCase()`

  Возвращают строку. Первый метод работает в IE5.5+, NN6, учитывает определенные языковые системы.

  Приведение строк к одному и тому же регистру требуется, например, при сравнении содержимого строк без учета регистра. Кроме того, многие серверы чувствительны к регистру, в котором определены имена файлов и папки.

  Примеры:

```
x="ЗдраВствуйТе"
x.toLocaleUpperCase()               // значение равно "ЗДРАВСТВУЙТЕ"
x.toUpperCase()                     // значение равно "ЗДРАВСТВУЙТЕ"
y="Здравствуйте"
x == y                              // значение равно false
x.toUpperCase() == y.toUpperCase()  // значение равно true
```

2.7.4. Методы объекта *String* форматирования строк

Тексты на Web-страницах обычно создаются и форматируются с помощью тегов HTML и каскадных таблиц стилей. Однако их можно создавать и с помощью сценариев. Например, чтобы вывести на Web-страницу строку "Привет всем!" полужирным шрифтом, в HTML-коде следует написать следующую инструкцию:

```
<B>Привет всем!</B>
```

Чтобы подготовить эту же строку в таком же формате средствами JavaScript, в сценарии следует написать такое выражение:

```
"Привет всем!".bold()
```

Здесь использован метод `bold()` строкового объекта для форматирования строк. Выполнение этого выражения лишь создает отформатированную строку, но не выводит ее в окно браузера. Чтобы сделать это, следует еще выполнить метод `write()` объекта `document` для записи этой строки в HTML-документ. В листинге 2.3 приведен пример кода.

Листинг 2.3. Вывод текста полужирным шрифтом с помощью сценария

```
<HTML>
  <SCRIPT>
    x="Привет всем!".bold()
    document.write(x)
  </SCRIPT>
  <P>Приветствие было вставлено сценарием JavaScript
</HTML>
```

В рассмотренном выше примере использован объект `documemt` и его метод `write()`.

Внимание

Методы форматирования строк носят названия, соответствующие тегам HTML. Их особенность в том, что в отличие от тегов, их следует записывать только в нижнем регистре (строчными буквами). Синтаксис такой же, как и у ранее рассмотренных методов:

строка`.`*метод*`(`*параметр*`)`

Большинство методов форматирования не имеют параметров:

- `anchor("`*anchor_имя*`")`
- `blink()`
- `bold()`
- `fixed()`
- `fontcolor(`*значение_цвета*`)`
- `fontsize(`*число от 1 до 7*`)`
- `italics()`
- `link(`*расположение или URL*`)`
- `big()`
- `small()`
- `strike()`
- `sub()`
- `sup()`

Примеры:

```
"Глава 2".anchor("volume2")   /* эквивалентно HTML-коду:
                                 <A NAME="volume2"> Глава 2 </A> */

"Страница автора".link("http:/www.admiral.ru/~dunaev")
     /* эквивалентно HTML-коду:
        <A HREF="http:/www.admiral.ru/~dunaev")> Страница автора </A> */
```

2.7.5. Функции вставки и замены подстрок

При обработке строк часто требуется вставить или заменить подстроки. Удаление подстроки из данной строки можно рассматривать как частный случай замены, а именно замены указанной подстроки пустой строкой. С помощью методов объекта `String` можно написать программу для решения этой задачи. Поскольку она часто встречается, то целесообразно оформить программу в виде функций.

Рассмотрим сначала функцию вставки строки в исходную строку. Назовем ее, например, `insstr`. Данная функция должна принимать три параметра: исходную строку `s1`, вставляемую строку `s2` и индекс позиции вставки `n`. Далее приведены ее определение и примеры вызова:

```
function insstr(s1,s2,n) {
  return s1.slice(0,n) + s2 + s1.slice(n)
}

insstr("Привет, друзья", " мои", 7)     // "Привет, мои друзья"
insstr("Привет, друзья", " мои", 100)   // "Привет, друзья мои"
```

Теперь займемся функцией, которая заменяет в исходной строке все вхождения заданной подстроки на подстроку замены. Назовем эту функцию `replacestr`. Она должна принимать три параметра: исходную строку `s1`, заменяемую подстроку `s2` и подстроку `s3`, которой следует заменить все вхождения `s2` в `s1` (`s2` может встречаться в `s1` несколько раз).

Очевидно, прежде всего, необходимо найти все вхождения `s2` в `s1`. Если исходная строка не содержит в себе подстрок `s2`, то функция должна вернуть исходную подстроку без всяких изменений. В противном случае требуется изъять из `s1` все вхождения `s2`, а на их место вставить подстроку `s3`. Полученная таким образом строка должна возвращаться функцией в качестве результата. Ниже приведено определение функции для замены подстрок и два примера ее вызова:

```
function replacestr(s1, s2, s3) {
  var s=""             // обработанная часть строки
  while (true) {
    i=s1.indexOf(s2)   // индекс вхождения s2 в s1
    if (i >= 0) {
      s=s + s1.substr(0, i) + s3    /* обработанная часть строки */
      s1=s1.substr(i + s2.length)   /* оставшаяся часть строки */
    }else break
  }
```

```
  return s + s1
}

replacestr("bacdae","a","X")   // "bXcdXe"
x="Иван Иванов"
replacestr(x,"Иван","Федор")   // "Федор Федоров"
```

Заметим, что если заменяемая подстрока `s2` является пустой, то цикл будет продолжаться бесконечно, поскольку пустая строка входит в состав любой строки. Чтобы избежать зацикливания, следует немного изменить код тела функции. Но что должна возвращать наша функция, если `s2` — пустая строка? Вы можете решить, что в этом случае следует просто вернуть исходную строку `s1`, или же, например, считать пустую строку строкой, содержащей пробел. Последний вариант представляется мне более интересным. Вот код, реализующий эту идею:

```
function replacestr(s1, s2, s3) {
  if (s2 == "")
    s2=" "            // заменяем пустую строку на строку с пробелом
  var s=""            // обработанная часть строки
  while (true) {
    i=s1.indexOf(s2)  // индекс вхождения s2 в s1
    if (i >= 0) {
      s=s + s1.substr(0, i) + s3    /* обработанная часть строки */
      s1=s1.substr(i + s2.length)   /* оставшаяся часть строки */
    }else break                     // выход из цикла
  }
  return s + s1
}
```

Рассмотренные выше функции вставки и замены подстрок (`insstr()` и `replacestr()`) готовы к практическому применению в программах, где требуется обработка строк. Вы можете сохранить определения этих функций в текстовом файле с расширением js, который будет играть роль библиотеки функций. Этот файл можно вызывать из основной программы, когда он потребуется.

2.7.6. Функции удаления ведущих и заключительных пробелов

При обработке строк (например, введенных пользователем в поля формы) нередко возникает задача удаления лишних ведущих и завершающих пробелов. Многие языки программирования имеют соответствующие встроенные

функции, но в JavaScript их нет. Поэтому не помешает создать их самостоятельно с помощью имеющихся средств и поместить в свою библиотеку (сохранить в текстовом файле с расширением js).

Решить поставленную задачу можно несколькими способами. Я придумал следующий алгоритм. Преобразуем исходную строку в массив слов, используя в качестве разделителя один пробел `" "`. Затем необходимо проанализировать первые или последние элементы массива в зависимости от того, что нам требуется: удалить ведущие или завершающие пробелы в исходной строке. Допустим, необходимо удалить ведущие пробелы. Тогда, если исходная строка содержала `N` пробелов, то первые `N` элементов массива будут содержать пустые строки `""`. Мы проверяем в цикле значения первых элементов, пока не найдем непустой элемент или пока не исчерпаем весь массив. Если первый непустой элемент массива имеет индекс `i`, то создадим новый массив, содержащий элементы исходного, начиная с этого индекса. Наконец, склеим элементы этого массива в одну строку, используя в качестве разделителя строку с единственным пробелом `" "`. Все, что нам понадобится, — это методы объекта `String`, оператор цикла и оператор условного перехода.

Сначала рассмотрим функцию `ltrim()`, удаляющую ведущие пробелы из исходной строки.

```
function ltrim(xstr){
  if (!(xstr.indexOf(" ") == 0))
  return xstr              /* вернуть исходную строку, если в ней нет
                              ведущих пробелов */
  var astr=xstr.split(" ") // создаем массив из слов строки
  var i=0
  while (i < astr.length){
    if (!(astr[i] == (""")))
      break                // выходим из цикла, если элемент не пуст
    i++
  }
  astr=astr.slice(i)       // создаем массив
  return astr.join(" ")    // склеиваем элементы массива в строку
}
```

Функция `rtrim()` для удаления заключительных пробелов в строке устроена аналогично. Ее отличие в том, что поиск пробелов происходит с конца строки. Обратите внимание, что счетчик цикла в этом случае убывающий.

```
function rtrim(xstr){
  if (!(xstr.lastIndexOf(" ") == xstr.length - 1))
    return xstr
```

```
  var astr=xstr.split(" ")
  var i=astr.length -1
  while (i>0){
    if (!(astr[i] == (""")))
      break
    i--
  }
  astr=astr.slice(0, i+1)
  return astr.join(" ")
}
```

Чтобы удалить из строки и ведущие, и концевые пробелы, достаточно выполнить следующее выражение:

```
ystr=rtrim(ltrim(xstr))
```

Впрочем, можно создать специальную функцию `trim()`, которая тримингует строку:

```
function trim(xstr) {
  return rtrim(ltrim(xstr))
}
```

2.8. Массивы

Массив представляет собой упорядоченный набор данных. Его удобно изобразить в виде одностолбцовой таблицы, содержащей некоторое количество строк. В ячейках такой таблицы могут находиться данные любого типа, в том числе и массивы. В последнем случае можно говорить о многомерных массивах (т. е. о массивах массивов). Количество элементов в массиве (строк в таблице) называется *длиной* массива. К элементам массива можно обращаться в программе по их порядковому номеру *(индексу)*. Нумерация элементов массива начинается с нуля, так что первый элемент имеет индекс 0, а последний — на единицу меньший, чем длина массива.

Массивы в JavaScript представляются экземплярами встроенного объекта `Array` (массив). Методы этого объекта позволяют выполнять различные операции над массивами, такие как объединение, сортировка и др.

2.8.1. Создание массива

Существует несколько способов создания массива. В любом случае, прежде всего, создается новый объект массива с использованием ключевого слова `new`:

```
имя_массива = new Array([длина_массива])
```

Здесь *длина_массива* является необязательным числовым параметром, о чем говорят квадратные скобки. Если длина массива не указана, то создается пустой массив, не содержащий ни одного элемента. В противном случае формируется массив с указанным количеством элементов, однако все они имеют значение `null` (т. е. не имеют значений).

Вы можете сначала создать пустой массив, а затем добавить к нему нужное количество элементов с помощью оператора присваивания. Заметим, что выражение с ключевыми словами `new Array` создает экземпляр (копию) объекта `Array`.

У объекта `Array` имеется свойство `length`, значением которого является длина массива. Чтобы получить значение этого свойства, необходимо использовать выражение

имя_массива`.length`

Создав массив, можно присвоить значения его элементам с помощью оператора присваивания. В левой части оператора присваивания указывается имя массива, а рядом с ним в квадратных скобках — индекс элемента. Мы уже говорили, что к элементам массива обращаются по индексу:

имя_массива`[`*индекс*`]`

Здесь квадратные скобки обязательны.

Рассмотрим создание массива `earth`, содержащего в качестве элементов некоторые характеристики нашей планеты. Обратите внимание, что элементы в этом массиве различных типов (строковые и числовые).

```
earth=new Array(4)      // массив из 4-х элементов
earth[0]="Планета"
earth[1]="24 часа"
earth[2]=6378
earth[3]=365.25

earth.length            // значение равно 4
```

Если нам потребуется значение, например, третьего элемента массива, то достаточно использовать выражение `earth[2]`.

Другой способ создания массива заключается в непосредственном определении элементов в круглых скобках за ключевым словом `Array`. Например,

```
earth=new Array("Планета", "24 часа", 6378, 365.25)
```

JavaScript автоматически создает индексы для элементов массива, так что к элементам массива, сформированного таким способом, также можно обращаться по индексам.

Третий способ создания массива — присвоить имя каждому элементу, подобно имени свойства объекта. Например,

```
earth=new Array()      // пустой массив
earth.xtype="Планета"
earth.xday="24 часа"
earth.radius=6378
earth.period=365.25
```

В этом случае обращение к элементу происходит как к свойству объекта, например, `earth.radius`. Возможен и такой вариант: `earth["radius"]`. По индексу к элементам в таком массиве обращаться нельзя.

2.8.2. Многомерные массивы

Массивы, рассмотренные выше, являются одномерными. Их можно представить себе в виде таблицы из одного столбца. Однако элементы массива могут содержать данные различных типов, в том числе и объекты, а значит, и массивы. Если в качестве элементов некоторого одномерного массива создать массивы, то получится двумерный массив. Обращение к элементам такого массива происходит в соответствии со следующим синтаксисом:

имя_массива[*индекс_уровня1*] [*индекс_уровня2*]

Если массив имеет размерность большую двух, то обращение к массивам имеет аналогичный синтаксис: следует добавить необходимое количество квадратных скобок, заключающих нужные индексы.

Типичным примером двумерного массива является массив опций меню. У такого меню есть горизонтальная панель с опциями, называемая *главным меню*. Некоторым опциям главного меню соответствуют раскрывающиеся вертикальные подменю со своими опциями. Мы создаем массив, длина которого равна количеству опций главного меню. Элементы этого массива определяем как массивы названий опций соответствующих подменю. Чтобы была ясна структура нашей конструкции, мы выбрали названия опций надлежащим образом. Например, "Меню 2.1" — название 1-й опции подменю, соответствующего 2-й опции главного меню.

```
menu=new Array()
menu[0]=new Array("Меню 1.1", "Меню 1.2", "Меню 1.3")
menu[1]=new Array("Меню 2.1", "Меню 2.2")
menu[2]=new Array("Меню 3.1", "Меню 3.2" , "Меню 3.3", "Меню 3.4")
```

Чтобы обратиться ко 2-й опции 3-го подменю, следует написать:

```
menu[2][1]       // значение равно "Меню 3.2"
```

Усложним нашу конструкцию, чтобы она содержала не только названия опций подменю, но и названия опций главного меню.

```
menu=new Array()
/* Массив опций главного меню: */
menu[0]=new Array("Меню 1", "Меню 2", "Меню 3")
menu[1]=new Array()
menu[1][0]=new Array("Меню 1.1", "Меню 1.2", "Меню 1.3")
menu[1][1]=new Array("Меню 2.1", "Меню 2.2")
menu[1][2]=new Array("Меню 3.1", "Меню 3.2" , "Меню3.3", "Меню 3.4")

menu[0][1]      // значение равно "Меню 2"
menu[0][2]      // значение равно "Меню 3"
menu[1][1][0]   // значение равно "Меню 2.1"
menu[1][2][3]   // значение равно "Меню 3.4"
```

2.8.3. Копирование массива

Иногда требуется создать копию массива, чтобы сохранить исходные данные и предохранить их от последующих модификаций. Например, метод сортировки элементов массива, который рассмотрен далее, изменяет исходный массив. Однако для копирования массива не достаточно присвоить его другой переменной. Используя новую переменную и оператор присваивания, мы создаем лишь новую ссылку на прежний массив, а не новый массив.

Пример:

```
a=new Array(5, 2, 4, 3)
x=a       // ссылка на массив a
a[2]=25   // изменение значение элемента с индексом 2
x[2]      // значение равно 25, т. е. новому значению a[2]
```

В этом примере массивы `a` и `x` совпадают. Изменение значения элемента в одном из них сразу таким же образом отразится на соответствующем элементе в другом массиве.

Чтобы скопировать массив, т. е. создать новый массив, элементы которого равны соответствующим элементам исходного, следует воспользоваться оператором цикла, в котором элементам нового массива присваиваются значения элементов исходного, например:

```
a=new Array(5, 2, 4, 3)
x=new Array()              // ссылка на массив a
for(i=0; i<a.length; i++) /* копирование значений массива a
                             в элементы массива x */
  { x[i]=a [i] }
```

2.8.4. Свойства объекта *Array*

Свойства объекта `Array` описаны далее.

- `length` — длина или, иными словами, количество элементов в массиве; целое число. Синтаксис:

  имя_массива`.length`

  Поскольку индексация элементов массива начинается с нуля, индекс последнего элемента на единицу меньше длины массива. Это обстоятельство удобно использовать при добавлении к массиву нового элемента:

  `myarray[mayarray.length]=`*значение*

- `prototype` — свойство (прототип), позволяющее добавить новые свойства и методы ко всем созданным массивам.

  Например, следующее выражение добавляет свойство `author` ко всем уже созданным массивам:

  ```
  Array.prototype.author="Иванов"
  ```

  Если теперь вместо `Array.prototype` написать имя существующего массива, то можно изменить значение свойства `author` только для этого массива:

  ```
  myarray=new Array()                  // создание массива myarray
  xarray=new Array()                   // создание массива xarray
  Array.prototype.author="Иванов"      /* добавление прототипа
                                          ко всем массивам */
  myarray.author="Иванов младший"      /* изменение свойства author
                                          для myarray */
  xarray.author="Сидоров"              /* изменение свойства author
                                          для xarray */
  ```

  Прототипу можно присвоить функции. При этом они пополнят множество методов объекта `Array`.

Пример: вычисление суммы элементов массива.

Определим функцию `aSum()`, которая возвращает сумму элементов числового массива. В качестве параметра эта функция принимает массив. Затем создадим конкретный массив чисел. Наконец, присоединим к прототипу массива новый метод `Sum` — определенную ранее функцию `aSum()`:

```
function aSum(xarray){
  var s=0
  for(i=0; i <= xarray.length -1; i++){
```

```
    s=s + xarray[i]
  }
  return s
}
myarray=new Array(2, 3, 4)     /* создаем массив myarray из 3-х чисел */
Array.prototype.Sum=aSum       /* присоединяем метод (около имени функции
                                  скобки указывать не нужно) */
myarray.Sum(myarray)           /* применяем метод Sum к массиву myarray,
                                  передавая его в качестве параметра */
```

Этот пример призван просто проиллюстрировать использование свойства `prototype`. Чтобы вычислить сумму элементов массива, достаточно написать следующее выражение:

```
s=aSum(myarray)
```

2.8.5. Методы объекта *Array*

Методы объекта `Array` предназначены для управления данными, сохраненными в структуре массива.

- `concat(`*массив*`)` — конкатенация массивов, объединяет два массива в третий массив. Синтаксис:

  имя_массива1`.concat(`*массив2*`)`

  Возвращает массив. Данный метод не изменяет исходные массивы.

  Пример:

  ```
  a1=new array(1, 2, "Звезда")
  a2=new array("а", "б", "в", "г")
  a3=a1.concat(a3)   // результат — массив с элементами:
                     // 1, 2, "Звезда", "а", "б", "в", "г"
  ```

- `join(`*разделитель*`)` — создает строку из элементов массива с указанным разделителем между ними; является строкой символов (возможно, пустой). Синтаксис:

  имя_массива`.join(`*строка*`)`

  Примеры:

  ```
  a=new array(1, 2, "Звезда")
  a.join(",")      // значение — строка "1,2,Звезда"

  a=new array(1, 2, "Звезда")
  a.join(" ")      // значение — строка "1 2 Звезда"
  ```

- `pop()` — удаляет последний элемент массива и возвращает его значение. Синтаксис:

  имя_массива`.pop()`

  Возвращает значение удаленного элемента массива. Совместимость: IE5.5+. Данный метод изменяет исходный массив.

- `push(`*значение*`|`*объект*`)` — добавляет к массиву указанное значение в качестве последнего элемента и возвращает новую длину массива. Синтаксис:

  имя_массива1`.push(`*значение*`|`*объект*`)`

  Возвращает число. Совместимость: IE5.5+. Данный метод изменяет исходный массив.

- `shift()` — удаляет первый элемент массива и возвращает его значение. Синтаксис:

  имя_массива`.shift()`

  Возвращает значение удаленного элемента массива. Совместимость: IE5.5+, NN4+. Данный метод изменяет исходный массив.

- `unshift(`*значение*`|`*объект*`)` — добавляет к массиву указанное значение в качестве первого элемента. Синтаксис:

  имя_массива`.unshift(`*значение*`|`*объект*`)`

  Совместимость: IE5.5+. Метод изменяет исходный массив.

- `reverse()` — изменяет порядок следования элементов массива на противоположный. Синтаксис:

  имя_массива`.reverse()`

  Возвращает массив. Данный метод изменяет исходный массив.

  Пример:

  ```
  a=new array(1, 2, "Звезда")
  a.reverse() // массив с элементами в следующем порядке: "Звезда", 2, 1
  ```

- `slice(`*индекс1*`[, `*индекс2*`])` — создает массив из элементов исходного массива с индексами указанного диапазона. Синтаксис:

  имя_массива`. slice(`*индекс1*`[, `*индекс2*`])`

  Возвращает массив. Данный метод не изменяет исходный массив.

  Второй параметр (конечный индекс) не является обязательным, о чем свидетельствуют квадратные скобки в описании синтаксиса. Если он не указан, то создаваемый массив содержит элементы исходного массива, начиная

с индекса *индекс1* и до конца. В противном случае создаваемый массив содержит элементы исходного массива, начиная с индекса *индекс1* и до индекса *индекс2*, за исключением последнего элемента. При этом исходный массив остается без изменений.

Пример:

```
a=new array(1, 2, "Звезда","a", "b")
a.slice(1,3)   // массив с элементами: 2, "Звезда"
a.slice(2)     // массив с элементами: "Звезда", "a", "b"
```

- `sort([функция_сортировки])` — сортирует (упорядочивает) элементы массива с помощью функции сравнения. Синтаксис:

  имя_массива. `sort([функция_сравнения])`

  Возвращает массив. Данный метод изменяет исходный массив.

  Параметр не обязателен, о чем свидетельствуют квадратные скобки. Если параметр не указан, то сортировка производится на основе сравнения ASCII-кодов символов значений. Это удобно для сравнения символьных строк, но не совсем подходит для сравнения чисел. Так, число 357 при сортировке считается меньшим, чем 85, поскольку сначала сравниваются первые символы и только в случае их равенства сравниваются следующие, и т. д. Таким образом, метод `sort()` без параметра подходит для простой сортировки массива со строковыми элементами.

  Можно создать собственную функцию для сравнения элементов массива, с помощью которой метод `sort()` упорядочит весь массив. Имя этой функции (без кавычек и круглых скобок) передается методу в качестве параметра. При работе метода функции передаются два элемента массива, а ее код возвращает методу значение, указывающее, какой из элементов должен следовать за другим. Допустим, сравниваются два элемента, `x` и `y`. Тогда в зависимости от числового значения (отрицательного, 0 или положительного), возвращаемого функцией сравнения, методом `sort()` принимается одно из трех возможных решений (табл. 2.7).

Таблица 2.7. *Результаты сравнения двух элементов массива*

Значение, возвращаемое функцией сравнения	**Результат сравнения `x` и `y`**
Меньше 0	`y` следует за `x`
0	Порядок следования `x` и `y` не изменяется
Больше 0	`x` следует за `y`

Итак, по какому критерию сортировать элементы массива, определяется кодом функции сравнения. Если элемент массива имеет значение `null`, то в Internet Explorer он размещается в начале массива.

Пример:

```
myarray=new Array(4, 2, 15, 3, 30  ) // числовой массив
function comp(x, y) {                // функция сравнения
  return x-y
}
myarray.sort(comp)        /* массив с элементами в порядке:
                             2, 3, 4, 15, 30 */
```

- `splice(`*индекс*, *количество*[, *элем1*[, *элем2*[, ..., *элемN*]]]`)` — удаляет из массива несколько элементов и возвращает массив из удаленных элементов, или заменяет значения элементов. Синтаксис:

  имя_массива `splice(`*индекс*, *количество*[, *элем1*[, *элем2*[, ..., *элемN*]]]`)`

  Совместимость: IE5.5+. Данный метод изменяет исходный массив.

  Первые два параметра обязательны, а следующие — нет. Первый параметр является индексом первого удаляемого элемента, а второй — количеством удаляемых элементов.

  Пример:

```
a=new Array("Вася", "Иван", "Марья", 12, 5)
x=a.splice(1,3)   /* x — массив элементов: "Иван", "Марья", 12
                     a — массив элементов: "Вася", 5 */
```

  Метод `splice()` позволяет также заменить значения элементов исходного массива, если указаны третий и, возможно, последующие параметры. Эти параметры представляют значения, которыми следует заменить исходные значения элементов массива. При таком использовании метода `splice()` важен первый параметр (индекс), а второй (количество) может быть равным нулю. В любом случае, если количество элементов замены больше значения второго параметра, то часть элементов исходного массива будет заменена, а часть элементов просто вставлена в него. При этом метод `splice()` возвращает другой массив, состоящий из элементов исходного, индексы которых соответствуют первому и второму параметрам. Но это справедливо, если второй параметр не равен 0.

  Пример.

```
a=new Array("Вася", "Иван", "Марья", 12, 5)
x=a.splice(1, 3, "Петр", "Кузьма", "Анна")
/* x — массив элементов: "Иван", "Марья", 12
   a — массив элементов: "Вася","Петр", Кузьма", "Анна",5 */
```

```
a=new Array("Вася", "Иван", "Марья", 12, 5)
x=a.splice(1, 0, "Петр", "Кузьма", "Анна", "Федор", "Ханс")
/* x — пустой массив
   a — массив элементов: "Вася","Петр", "Кузьма",
                         "Анна", "Федор", "Ханс" */
```

- `toLocaleString()`, `toString()` — преобразуют содержимое массива в символьную строку.

  Метод `toLocaleString()` поддерживается браузерами IE5.5+ и NN3+, а метод `toString()` — и более ранними версиями. Алгоритм преобразования по методу `toLocaleString()` зависит от версии браузера.

  Для преобразования содержимого массива в строку рекомендую использовать метод `join()`.

2.8.6. Функции обработки числовых массивов

Во многих приложениях требуется получить статистические характеристики числовых данных, хранящихся в виде массива: сумму всех чисел, среднее, максимальное и минимальное значения. Коды функций, вычисляющих эти величины, необходимо написать самому. Затем их можно сохранить в отдельном файле с расширением js как библиотеку. Далее приведены коды некоторых функций, вычисляющих характеристики числовых данных, объединенных в массиве.

- Функция, возвращающая сумму значений всех элементов непустого массива:

```
function S(aN){
  var S=aN[0]
  for(var i=1; i<= aN.length-1; i++){
    S += aN[i]
  }
  return S
}
```

  Очевидно, для вычисления среднего значения элементов массива `aN` следует просто воспользоваться выражением: `S(aN)/aN.length`.

- Функция, возвращающая минимальное значение среди элементов массива:

```
function Nmin(aN){
  var Nmin=aN[0]
  for(var i=1; i <= aN.length-1; i++){
```

```
      if (aN[i] < Nmin)
        Nmin=aN[i]
      }
    return Nmin
  }
```

- Функция, возвращающая максимальное значение среди элементов массива:

```
function Nmax(aN){
  var Nmax=aN[0]
  for(var i=1; i <= aN.length -1; i++){
    if (aN[i] > Nmax)
      Nmax=aN[i]
    }
  return Nmax
}
```

Мы можем создать одну функцию, которая вычисляет все перечисленные выше статистические характеристики и возвращает их как значения массива:

```
function statistic(aN){
  if (aN == 0 || aN == null || aN == "")
    return new Array(0,0,0,0)
  var S=aN[0]
  var Nmin=aN[0]
  var Nmax=aN[0]
  for(var i=1; i<=aN.length-1; i++){
    S += aN[i]
    if (aN[i] < Nmin)
      Nmin=aN[i]
    if (aN[i] > Nmax)
      Nmax=aN[i]
  }
  return new Array(S, S/aN.length, Nmin, Nmax)
}
```

В начале кода функции `statistic()` мы проверяем, не является ли параметр пустым. Если это так, то все статистические характеристики считаются равными нулю.

2.9. Числа

В JavaScript числа могут быть только двух типов: целые и с плавающей точкой. Целые числа не имеют дробной части и не содержат разделительной точки. Числа с плавающей точкой имеют целую и дробную части, разделенные точкой.

Операции с целыми числами процессор компьютера выполняет значительно быстрее, чем операции с числами, имеющими точку. Это обстоятельство имеет смысл учитывать, когда расчетов много. Например, индексы, длины строк являются целочисленными. Число π, многие числа, полученные с помощью оператора деления, денежные суммы и т. п. являются числами с плавающей точкой.

В JavaScript можно производить операции с числами различных типов. Однако при этом следует знать, какого числового типа будет результат. Если результат операции является числом с дробной частью, то он представляется как число с плавающей точкой. Если результат оказался без дробной части, то он приводится к целочисленному типу, а не представляется числом, у которого в дробной части одни нули. Вот несколько примеров:

```
2 + 3       // 5 — целое число
2 + 3.5     // 5.5 — число с плавающей точкой
2.4 + 3.6   // 6 — целое число
6.00        // число с плавающей точкой
```

Числа можно представлять и в так называемой экспоненциальной форме, т. е. в формате: *число1*e*число2* или *число1*E*число2*. Такая запись числа означает $число1 \times 10^{число2}$. Например:

```
1e5        // 100 000
2e6        // 2 000 000
1.5e3      // 1500
+1.5e3     // 1500
-1.5e3     // -1500
3e-4       // 0.0003
```

Числа в JavaScript можно представлять в различных системах счисления, т. е. в системах с различными основаниями: 10 (десятичной), 16 (шестнадцатеричной), 8 (восьмеричной) и 2 (двоичной). К десятичной форме представления чисел мы привыкли, однако следует помнить, что числа в этой форме не должны начинаться с 0, потому что так записываются числа в восьмеричной системе.

Запись числа в шестнадцатеричной форме начинается с префикса `0x` (или `0X`), где первый символ ноль, а не буква О, затем следуют символы шестнадцатеричных цифр: `0`, `1`, `2`, ..., `9`, `a`, `b`, `c`, `d`, `e`, `f` (буквы могут быть в любом регистре). Например, шестнадцатеричное число `0x4af` в десятичном представлении есть `1199`.

Запись числа в восьмеричной форме начинается с нуля, за которым следуют цифры от `0` до `7`. Например, `027` (в десятичном представлении — `23`).

В двоичной системе числа записываются посредством двух цифр — `0` и `1`.

В арифметических выражениях числа могут быть представлены в любой из перечисленных выше систем счисления, однако результат всегда приводится к десятичной форме.

- Пример: функция преобразования из десятичной формы в шестнадцатеричную.

  Функция `to16()` в этом примере принимает в качестве параметра десятичное число и преобразует его в строку, содержащую это же число, но в шестнадцатеричной форме. При этом мы ограничиваемся числами, не превышающими 255 (тогда для представления числа потребуется не более двух шестнадцатеричных цифр).

```
function to16(n10) {
  hchars="0123456789abcdef"     /* строка, содержащая все
                                   шестнадцатеричные цифры */
  if (n10 > 255) return null
  var i=n10%16
  var j=(n10 - i)/16
  rezult="0x"
  rezult += hchars.charAt(j)
  rezult += hchars.charAt(i)
  return rezult
}

to16(250)                 // "0xfa"
to16(30)                  // "0x1e"
to16(30)+10               // "0x1e10" - склейка, а не сложение
parseInt(to16(30)) + 10   // 40
```

  Напомним, что выражение вида `x+=y` эквивалентно выражению `x=x+y`.

Функцию `to16()` можно также создать и на основе массивов:

```
function to16(n10) {
  hchars=new Array("0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
"a", "b", "c", "d", "e", "f"}
  if (n10 > 255) return null
  var i=n10%16
  var j=(n10 — i)/16
  rezult="0x"
  rezult += hchars[j]
  rezult += hchars[i]
  return rezult
}
```

- Пример: функция преобразования из десятичной формы в двоичную.

  Функция `to2()` в этом примере принимает в качестве параметра десятичное число и преобразует его в строку, содержащую это же число, но в двоичной форме. Результирующая строка состоит из нулей и единиц. Здесь осуществляется рекурсивный вызов функции `to2()` самой себя. Обратите внимание, что использование ключевого слова `var` в данном случае принципиально важно из-за рекурсии, иначе функция будет работать неправильно.

```
function to2(n10) {
if (n10 <2)
  return "" + n10 // чтобы результат был строковым
var i=n10%2
var j=(n10-i)/2
return to2(j)+i
}
```

Для преобразования строк, содержащих числа, в данные числового типа служат встроенные функции `parseInt()` и `parseFloat()` соответственно для представления в целочисленном виде и в виде числа с плавающей точкой. Их мы уже рассматривали ранее *(см. разд. 2.6.1)*. Здесь следует отметить, что параметрами этих функций могут быть строки, содержащие числа не только в десятичной, но и в шестнадцатеричной, восьмеричной и двоичной формах. Для указания основания системы счисления служит второй параметр этих функций.

Примеры:

```
parseInt("ff",16)    // значение равно 255
parseInt("ff")       // значение равно NaN
parseInt("010")      // значение равно 8
```

```
parseInt("010", 8)   // значение равно 8
parseInt("010", 10)  // значение равно 10
parseInt("010", 2)   // значение равно 2
parseInt("010", 16)  // значение равно 17
```

Примечание

Функция `parseFloat()` в IE6.0 работает неправильно, если указан второй параметр (основание системы счисления). Для правильного преобразования шестнадцатеричного числа, последнее не должно быть представлено как данные строкового типа, в отличие от функции `parseInt()`.

Для преобразования числа в строку, содержащую это число, достаточно использовать выражение сложения пустой строки с числом.

Примеры:

```
"" + 25.78 // значение равно "25.78"
"" + 2.5e3 // значение равно "2500"
```

2.9.1. Создание объекта *Number*

Числа являются экземплярами объекта `Number`. Их можно создавать обычным образом с помощью переменных и оператора присваивания, не прибегая к объекту `Number`. Однако этот объект обладает некоторыми полезными свойствами и методами, которые иногда могут пригодиться.

Объект `Number` создается с помощью выражения вида:

переменная=new Number(*число*)

Доступ к свойствам и методам строкового объекта обеспечивается такими выражениями:

число.*свойство*

Number.*свойство*

число.*метод*([*параметры*])

Number.*метод*([*параметры*])

2.9.2. Свойства объекта *Number*

Перечислим их:

- `MAX_VALUE` — константа, значение которой равно наибольшему допустимому в JavaScript значению числа (`1.7976931348623157e+308`);
- `MIN_VALUE` — константа, значение которой равно наименьшему допустимому в JavaScript значению числа (`5e-324`);

- `NEGATIVE_INFINITY` (минус бесконечность) — число, меньшее, чем `Number.MIN_VALUE`;
- `POSITIVE_INFINITY` (плюс бесконечность) — число, большее, чем `Number.MAX_VALUE`;
- `NaN` — константа, имеющая значение `NaN`, посредством которой JavaScript сообщает, что данные (параметр, возвращаемое значение) не является числом (Not a Number);
- `prototype` — свойство (прототип), играющее такую же роль, что и в случае объекта `String` *(см. разд. 2.7.2)*.

2.9.3. Методы объекта *Number*

Объект `Number` имеет несколько методов, из которых мы рассмотрим только четыре, предназначенные для представления чисел в виде строки в том или ином формате.

- `toExponential(`*количество*`)` — представляет число в экспоненциальной форме. Синтаксис:

  число`.toExponential(`*количество*`)`

  Возвращает строку. Совместимость: IE5.5+, NN6+.

  Параметр представляет собой целое число, определяющее, сколько цифр после точки следует указывать.

  Примеры:

  ```
  x=new Number(456)
  x.toExponential(3)   // 4.560e+2
  x.toExponential(2)   // 4.56e+2
  x.toExponential(1)   // 4.6e+2
  x.toExponential(0)   // 5e+2
  ```

- `toFixed(`*количество*`)` — представляет число в форме с фиксированным количеством цифр после точки. Синтаксис:

  число`.toFixed(`*количество*`)`

  Возвращает строку. Совместимость: IE5.5+, NN6+.

  Параметр — целое число, определяющее, сколько цифр после точки следует указывать.

  Примеры:

  ```
  x=new Number(25.65)
  x.toFixed(3)    // 25.650
  ```

```
x.toFixed(2)    // 25.65
x.toFixed(1)    // 25.7
x.toFixed(0)    // 26
```

- `toPrecision(`*точность*`)` — представляет число с заданным общим количеством значащих цифр. Синтаксис:

  число`.toPrecision(`*точность*`)`

  Возвращает строку. Совместимость: IE5.5+, NN6+.

  Параметр — целое число, определяющее, сколько всего цифр, до и после точки, следует указывать.

  Примеры:

```
x=new Number(135.45)
x.toPrecision(6)  // 135.450
x.toPrecision(5)  // 135.45
x.toPrecision(4)  // 135.5
x.toPrecision(3)  // 135
x.toPrecision(2)  // 1.4e2
x.toPrecision(1)  // 1e2
x.toPrecision(0)  // Сообщение об ошибке
```

- `toString([`*основание*`])` — возвращает строковое представление числа в системе счисления с указанным основанием. Синтаксис:

  число`.toString([`*основание*`])`

  Если параметр не указан, имеется в виду десятичная система счисления.

  Вы можете указать 2 для двоичной системы, или 16 — для шестнадцатеричной. Заметим, что этот метод имеют все объекты.

  Примеры:

```
x=new Number(127.18)
x.toString()     // "127.18"
x.toString(10)   // "127.18"
x.toString(16)   // "7f.2e147ae147b"
x.toString(8)    // "177.134121727024366"

x=new Number(5)
x.toString(2)    // "101"
```

2.10. Математические вычисления

Элементарные математические функции и некоторые константы представлены методами и свойствами встроенного объекта `Math` (математика). Доступ к свойствам и методам объекта `Math` обеспечивается следующими выражениями:

`Math.`*свойство*

`Math.`*метод*`(`*параметры*`)`

2.10.1. Свойства объекта *Math*

Свойства объекта `Math` имеют в качестве своих значений математические константы (табл. 2.8).

Таблица 2.8. *Константы объекта* `Math`

Константа	Описание
`E`	Постоянная Эйлера
`LN10`	Значение натурального логарифма числа 10
`LN2`	Значение натурального логарифма числа 2
`LOG10E`	Значение десятичного логарифма экспоненты (числа `e`)
`LOG2E`	Значение двоичного логарифма экспоненты
`PI`	Значение постоянной π
`SQRT1_2`	Значение квадратного корня из 1/2
`SORT`	Значение квадратного корня из 2

Пример. Для вычисления длины окружности при известном радиусе требуется число π, которое можно взять как свойство объекта `Math`.

```
var R=10              // радиус окружности
circus=2*R*Math.PI    // длина окружности
```

2.10.2. Методы объекта *Math*

Перечислим их:

- `abs(`*число*`)` — модуль (абсолютное значение) числа;
- `acos(`*число*`)` — арккосинус числа;
- `asin(`*число*`)` — арксинус числа;

- `atan(`*число*`)` — арктангенс числа;
- `atan2(`*x*`, `*y*`)` — угол в полярных координатах точки;
- `ceil(`*число*`)` — округление числа вверх до ближайшего целого;
- `cos(`*число*`)` — косинус числа;
- `exp(`*число*`)` — число `e` в степени *число*;
- `floor(`*число*`)` — округление числа вниз до ближайшего целого;
- `log(`*число*`)` — натуральный логарифм числа;
- `max(`*число1*`, `*число2*`)` — большее из чисел *число1*, *число2*;
- `min(`*число1*`,`*число2*`)` — меньшее из чисел *число1*, *число2*;
- `pow(`*число1*`, `*число2*`)` — *число1* в степени *число2*;
- `random()` — случайное число между 0 и 1;
- `round(`*число*`)` — округление числа до ближайшего целого;
- `sin(`*число*`)` — синус числа;
- `sqrt(`*число*`)` — квадратный корень из числа;
- `tan(`*число*`)` — тангенс числа.

Примеры.

- Метод `random()` возвращает случайное число, лежащее в интервале от 0 до 1.
  - Чтобы получить случайное число в пределах от 0 до `Nmax`, следует написать следующее выражение:

    ```
    x=Nmax*Math.random()
    ```

  - Если требуется получить случайное число в интервале от `Nmin` до `Nmax`, то из элементарного отношения пропорций получаем следующее выражение:

    ```
    x=Nmin + (Nmax – Nmin)*Math.random()
    ```

  - Можно также создать функцию для вычисления случайного числа в заданном интервале:

    ```
    function rand(a, b) {
      return a+ (b-a)*Math.random()
    }
    ```

    Эта функция может потребоваться, например, для внесения некоторой непредсказуемости (нерегулярности) перемещения элементов на Web-странице, выбора цветов для мигающей надписи и т. п.

- Для вычисления значения тригонометрической функции `sin(x)`, у которой аргумент `x` выражен в градусах, необходимо применить следующее выражение:

  ```
  Math.sin(Math.PI*x/180)
  ```

2.11. Дата и время

В основе всех операций, связанных с датами и временем, лежат текущие системные дата и время, установленные на вашем компьютере. Со временем дела обстоят не так просто, как кажется на первый взгляд. Вспомните, что существуют временны́е зоны (часовые пояса), а также сезонные поправки времени. Чтобы иметь возможность координировать деятельность во времени организаций и физических лиц в различных точках нашей планеты, была введена система отсчета времени. Она связана с меридианом, проходящим через астрономическую обсерваторию в городе Гринвич в Великобритании. Эту временную зону называют средним временем по Гринвичу (Greenwich Mean Time, GMT) или всеобщим скоординированным временем (Coordinated Universal Time, UTC).

Если системные часы вашего компьютера установлены правильно, то отсчет времени производится в системе GMT. Однако в Панели управления обычно устанавливается локальное время, соответствующее конкретному часовому поясу. При создании и изменении файлов на вашем компьютере фиксируется именно локальное время. Вместе с тем операционная система знает разницу между локальным временем и GMT. При перемещении компьютера из одного часового пояса в другой необходимо изменить установки именно часового пояса, а не текущего системного времени (показания системных часов). Даты и время, генерируемые в сценариях, сохраняются в памяти в системе GMT, но пользователю выводятся, как правило, в локальном виде.

В программе на JavaScript нельзя просто написать 10.01.2006, чтобы получить значение даты, с которым в дальнейшем можно производить некие операции. Значения даты и времени создаются как экземпляры специального объекта `Date`. При этом объект сам будет "знать", что не бывает 31 июня и 30 февраля, а в високосных годах 366 дней.

2.11.1. Создание объекта *Date*

Чтобы создать значение даты и времени, а затем использовать его в программе, необходимо сначала создать экземпляр встроенного объекта `Date` с помощью следующего синтаксиса:

имяОбъектаДаты=new Date([*параметры*])

Параметры не обязательны, на что указывают квадратные скобки.

Для манипуляций объектом `Date` применяется множество методов. При этом используется такой синтаксис:

переменная=имяОбъектаДаты.метод()

Если, например, объекту `Date` требуется присвоить новое значение, то для этого применяется соответствующий метод:

переменная=имяОбъектаДаты.метод(*новое_значение*)

Рассмотрим в качестве примера изменение значения года текущей системной даты:

```
xdate=new Date()        /* создание объекта, содержащего текущую дату
                           и время */
Year=xdate.getYear()    /* в переменной Year содержится
                           значение текущего года */
Year=Year + 3           /* в переменной Year содержится значение,
                           большее, чем текущий год на 3 */
xdate.setYear(Year)     /* в объекте устанавливается новое
                           значение года */
```

При создании объекта даты с помощью выражения `new Date()` можно указать в качестве параметров, какую дату и время следует установить в этом объекте. Это можно сделать пятью способами:

new Date("*Месяц дд, гггг чч:мм:сс*")

new Date("*Месяц дд, гггг*")

new Date(*гг, мм, дд, чч, мм, сс*)

new Date(*гг, мм, дд*)

new Date(*миллисекунды*)

В первых двух способах параметры задаются в виде строки, в которой указаны компоненты даты и времени. Буквенные обозначения определяют шаблон параметров. Обратите внимание на разделители — запятые и двоеточия. Время указывать не обязательно. Если компоненты времени опущены, то устанавливается значение 0 (полночь). Компоненты даты обязательно должны быть указаны. Месяц задается своим полным английским названием (аббревиатуры не допускаются). Остальные компоненты указываются в виде чисел. Если число меньше 10, то можно писать одну цифру, не записывая ведущий 0 (например, 3:05:32).

В третьем и четвертом способах компоненты даты и времени представляются целыми числами, разделенными запятыми.

В последнем способе дата и время задаются целым числом, которое представляет количество миллисекунд, прошедших с начала 1 января 1970 года (т. е. с момента 00:00:00). Количество миллисекунд, отсчитанное от указанной стартовой даты, позволяет вычислить все компоненты и даты, и времени.

2.11.2. Методы объекта *Date*

Для чтения и изменения информации о дате и времени, хранящейся в объекте даты, служат методы объекта `Date` (табл. 2.9). Напомним, что объект даты создается с помощью выражения:

имяОбъектаДаты=new Date([*параметры*])

Затем, чтобы применить метод *метод*() к созданному объекту, следует написать:

имяОбъектаДаты.*метод*([*параметры*])

Довольно большое множество всех методов можно разделить на две категории: методы получения значений (их названия имеют префикс `get`) и методы установки новых значений (их названия имеют префикс `set`). В каждой категории выделяются две группы методов — для локального формата и формата UTC. Методы позволяют работать с отдельными компонентами даты и времени (годом, месяцем, числом, днем недели, часами, минутами, секундами и миллисекундами).

Таблица 2.9. *Методы объекта* `Date`

Метод	Диапазон значений	Описание
getFullYear()	1970 и далее	Год
getYear()	70 и далее	Год
getMonth()	0—11	Месяц (январь соответствует 0)
getDate()	1—31	Число
getDay()	0—6	День недели (воскресенье соответствует 0)
getHours()	0—23	Часы в 24-часовом формате
getMinutes()	0—59	Минуты
getSeconds()	0—59	Секунды
getTime()	0 и далее	Миллисекунды с 1.1.70 00:00:00 GMT
getMilliseconds()	0 и далее	Миллисекунды с 1.1.70 00:00:00 GMT
getUTCFullYear()	1970 и далее	Год UTC

Таблица 2.9 *(продолжение)*

Метод	Диапазон значений	Описание
`getUTCMonth()`	0—11	Месяц UTC (январь соответствует 0)
`getUTCDate()`	1—31	Число UTC
`getUTCDay()`	0—6	День недели UTC (воскресенье соответствует 0)
`getUTCHours()`	0—23	Часы UTC в 24-часовом формате
`getUTCMinutes()`	0—59	Минуты UTC
`getUTCSeconds()`	0—59	Секунды UTC
`getUTCMilliseconds()`	0 и далее	Миллисекунды UTC с 1.1.70 00:00:00 GMT
`setYear(`*знач*`)`	1970 и далее	Установка года (четырехзначного)
`setFullYear(`*знач*`)`	1970 и далее	Установка года
`setMonth(`*знач*`)`	0—11	Установка месяца (январь соответствует 0)
`setDate(`*знач*`)`	1—31	Установка числа
`setDay(`*знач*`)`	0—6	Установка дня недели (воскресенье соответствует 0)
`setHours(`*знач*`)`	0—23	Установка часов в 24-часовом формате
`setMinutes(`*знач*`)`	0—59	Установка минут
`setSeconds(`*знач*`)`	0—59	Установка секунд
`setMilliseconds(`*знач*`)`	0 и далее	Установка миллисекунд с 1.1.70 00:00:00 GMT
`setTime(`*знач*`)`	0 и далее	Установка миллисекунд с 1.1.70 00:00:00 GMT
`setUTCFullYear(`*знач*`)`	1970 и далее	Установка года UTC
`setUTCMonth(`*знач*`)`	0–11	Установка месяца UTC (январь соответствует 0)
`setUTCDate(`*знач*`)`	1—31	Установка числа UTC
`setUTCDay(`*знач*`)`	0—6	Установка дня недели UTC (воскресенье соответствует 0)
`setUTCHours(`*знач*`)`	0—23	Установка часов UTC в 24-часовом формате
`setUTCMinutes(`*знач*`)`	0—59	Установка минут UTC

Таблица 2.9 *(окончание)*

Метод	Диапазон значений	Описание
`setUTCSeconds(`*знач*`)`	0—59	Установка секунд UTC
`setUTCMilliseconds(`*знач*`)`	0 и далее	Установка миллисекунд UTC с 1.1.70 00:00:00 GMT
`getTimezoneOffset()`	0 и далее	Разница в минутах по отношению к GMT/UTC
`toDateString()`		Строка с датой (без времени) в формате браузера (IE5.5)
`toGMTString()`		Строка с датой и временем в глобальном формате
`toLocaleDateString()`		Строка с датой без времени в локализованном формате системы (NN6, IE5.5)
`toLocaleString()`		Строка с датой и временем в локализованном формате системы
`toLocaleTimeString()`		Строка со временем без даты и в локализованном формате системы (NN6, IE5.5)
`toString()`		Строка с датой и временем в формате браузера
`toTimeString()`		Строка со временем без даты в формате браузера (IE5.5)
`toUTCString()`		Строка с датой и временем в глобальном формате
`Date.parse("`*дата*`")`		Преобразование строки с датой в число миллисекунд
`Date.UTC(`*знач*`)`		Преобразование строки с датой GMT в число

Вычислять разность двух дат или создавать счетчик времени, оставшегося до некоторого заданного срока, можно с помощью методов как для локального формата, так и UTC. Не следует применять выражения, использующие различные форматы времени, поскольку результаты могут оказаться неправильными. Формат UTC обычно применяется в расчетах, учитывающих часовой пояс.

Следует также принять во внимание, что нумерация месяцев, дней недели, часов, минут и секунд начинается с 0. Для компонентов времени это естест-

венно. Однако при такой нумерации декабрь оказывается 11-м месяцем в году, а не 12-м, как это принято повсеместно. Воскресенье (Sunday) является 0-м днем недели, а не 7-м.

Значение года XX века представляется в двухзначном формате как разность между этим годом и 1900. Например, 1996 год представляется как 96. Годы до 1900 и после 1999 обозначаются в четырехзначном формате. Например, 2005 год нужно так и писать — 2005, поскольку 05 — это 1905 год. Метод `getFullYear()` возвращает четырехзначное значение года.

Примеры.

Допустим, что сейчас 2006 год.

```
today=new Date()          // текущие дата и время
Year=today.getYear()      // 2006

today=new Date(98,11,6)   /* объект даты today содержит
                             информацию о дате 6 ноября 1998 года,
                             00:00:00 */
Year=today.getYear()      // 98
today.getFullYear()       // 1998
```

Изменение любого компонента даты производится с помощью соответствующего метода, название которого начинается с приставки `set`. При этом значения других компонентов пересчитываются автоматически.

В следующем примере создается объект `Date`, содержащий некоторую конкретную дату. Затем мы устанавливаем новое значение года. При этом в объекте даты изменяется и число.

```
mydate=new Date(1952, 10,6)   /* "Thu Nov 6 00:00:00 UTC+0300 1952" */
myday=mydate.getDay()         // 6
mydate.setYear(2006)          // установка 2006 года
myday=mydate.getDay()         // 1
```

При попытке отобразить значение объекта `Date` оно автоматически преобразуется в строку методом `toString()`. Формат этой строки зависит от операционной системы и браузера. Например, для Windows XP и Internet Explorer 5.5+ строка, содержащая информацию объекта даты, имеет следующий вид:

```
Mon Jan 9 13:29:17 UTC+0300 2006
```

т. е. понедельник, январь, 17, 13 часов 29 минут 17 секунд Всеобщего времени, скорректированного на 3 часа, 2006 год.

Если коррекцию времени с учетом часового пояса производить не требуется, то для строкового представления даты и времени можно воспользоваться методом `toLocaleString()`:

```
mydate=new Date()
mydate.toLocaleString()        // "9 января 2006 г. 13:29:17"
```

В браузерах IE5.5+ и NN6+ работают еще два метода представления отдельно даты и времени в виде строки:

```
mydate=new Date()
mydate.toLocaleDateString()    // "9 января 2006 г."
mydate.toLocaleTimeString()    // "13:29:17"
```

Заметим, что формат представления даты и времени, обеспечиваемый методами `toLocalDateString()` и `toLocalTimeString()`, зависит от настроек операционной системы и браузера. Если вы будете использовать эти методы для вывода даты и времени на Web-страницу, то они будут выглядеть так, как пользователь привык их видеть на своем компьютере.

Иногда приходится выполнять различные вычисления, такие как определение даты через заданное количество дней от текущей или количество дней между двумя датами. Вот здесь как раз и требуется предварительный подсчет миллисекунд, содержащихся в минуте, часе, сутках и т. д.

Примеры.

- Определим дату, которая наступит через неделю относительно текущей:

```
week=1000*60*60*24*7              /* количество миллисекунд в неделе —
                                     604800000 */
mydate=new Date()                 // объект Date с текущей датой
mydate_ms=mydate.getTime()        /* текущая дата, представленная
                                     количеством миллисекунд
                                     от 1.01.1970 00:00:00 */
mydate_ms += week                 // maydate_ms=mydate_ms + week
mydate.setTime(mydate_ms)         /* установка новой даты в объекте
                                     mydate */
newdate=mydate.toLocaleString()   /* новая дата в виде строки */
```

- Определим количество дней между двумя датами, например, 10 февраля 2006 года и 5 марта 2005 года. Для этого создадим сначала два соответствующих объекта даты:

```
date1=new Date(2006,01,10)
date2=new Date(2006,02,5)
```

Переменные date1 и date2 хранят длинные строки, содержащие даты. Чтобы перевести их в количество миллисекунд, воспользуемся методом parse() объекта Date. Затем вычислим разность Date.parse(date2) и Date.parse(date1) и разделим ее на количество миллисекунд в одних сутках:

```
days=(Date.parse(date2) - Date.parse(date1))/1000/60/60/24
  // результат: 23
```

Часто приходится иметь дело с датами, представленными в виде строк в формате "*дд.мм.гггг*", причем месяцы нумеруются с 1 до 12, а не с 0 до 11. В этом случае, чтобы воспользоваться методами объекта Date, необходимо предварительно выполнить соответствующие преобразования.

Допустим, исходная дата strdate представляется в виде строки в формате "*дд.мм.гггг*", а для создания объекта Date используется формат параметров "*гг, мм, дд*". Тогда необходимые преобразования исходных данных выглядят следующим образом:

```
astrdate=strdate.split(".")   /* массив подстрок, полученных из строки
                                 strdate с использованием точки
                                 как разделителя */
/* создаем объект Date */
mydate=new Date(astrdate[2], astrdate[1]-1, astrdate[0])
```

Развивая идею, рассмотренную в предыдущем примере, мы могли бы создать функцию, принимающую в качестве параметра строку, содержащую дату и время в привычном для нас формате "*дд.мм.ггг чч:мм:сс*" и возвращающую объект даты. Будем считать, что в строке параметра нашей функции дата отделена от времени одним пробелом, однако допускается, что информации о времени вообще может не быть. Далее, если время присутствует, секунды не обязательно должны указываться. Если параметр является пустой строкой или вообще отсутствует, то функция возвращает объект с текущей датой.

```
function mydate(strdate){
  if (strdate == "" || strdate == null) {
    return new Date()
  }
  var astrdate=strdate.split(" ")
  if (astrdate.length == 1)
    astrdate[1]="00:00:00"
  astrdate[0]=astrdate[0].split(".")
  astrdate[1]=astrdate[1].split(":")
  if (astrdate[1].length == 1)
    astrdate[1]=new Array(astrdate[1][0], "00","00")
  else {
    if (astrdate[1].length == 2)
```

```
      astrdate[1]=new Array(astrdate[1][0], astrdate[1][1] , "00")
  }
  return new Date(astrdate[0][2], astrdate[0][1]-1, astrdate[0][2],
              astrdate[1][0], astrdate[1][1], astrdate[1][2])
}
```

С использованием функции `mydate()` вычисление количества дней между двумя заданными датами может выглядеть следующим образом:

```
date1=mydate("10.2.2003")
date2=mydate("5.3.2003")
days=(Date.parse(date2) — Date.parse(date1))/1000/60/60/24
```

Иногда в приложениях, в том числе и на Web-страницах, устанавливают часы (таймер), показания которых постоянно изменяются в соответствии с ходом системных часов. Чтобы сделать это, требуется в цикле создавать объект `Date` и формировать строку из составляющих времени, используя подходящие для этого методы. Однако при этом программа, в которой имеется код, играющий роль часов, будет заниматься только этими часами (зациклится). Чтобы этого не произошло, используют метод `setInterval()` из объектной модели браузера.

2.11.3. Календарь

Рассмотрим в качестве примера работы с объектом даты календарь, показанный на рис. 2.3.

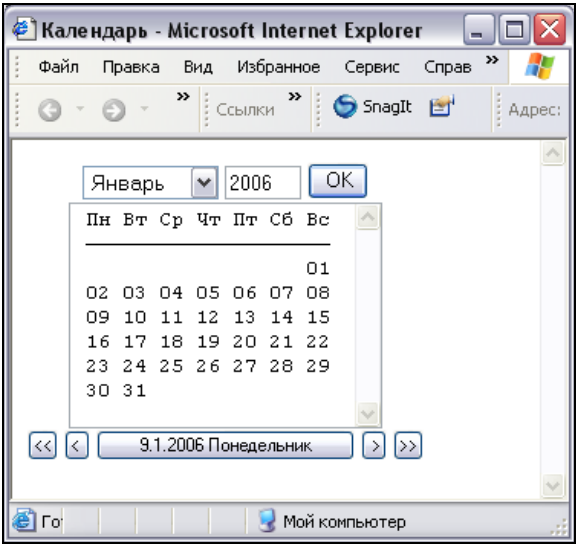


Рис. 2.3. Календарь

Календарь содержит текстовую область, в которой отображаются числа и названия дней недели, раскрывающийся список, из которого можно выбрать месяц, текстовое поле для ввода года и несколько кнопок для переходов к другим месяцам и годам. С помощью элементов пользовательского интерфейса можно вывести календарь на любой месяц и год. Код программы представлен в листинге 2.4.

Листинг 2.4. Календарь

```
<HTML>
  <HEAD>
    <TITLE>Календарь</TITLE>
  </HEAD>
  <SCRIPT LANGUAGE="JavaScript">
    var nowdate

    function setToday() {      // Установка текущей даты
      var now  =new Date();
      var day  =now.getDate();
      var month=now.getMonth();
      var year =now.getYear();
      var anameday=new Array("Воскресенье", "Понедельник",
          "Вторник", "Среда", "Четверг", "Пятница", "Суббота")
      var nameday=anameday[now.getDay()] // Название дня недели
      if (year < 2000)
        year=year + 1900;
      this.focusDay=day;
      document.calControl.month.selectedIndex=month;
      document.calControl.year.value=year;
      displayCalendar(month, year);
      nowdate=day+"." + parseInt(month+1) + "." + year;
      document.all.PrevYear2.value=nowdate + " " + nameday;
    }

    function isFourDigitYear(year) {
      if (year.length != 4) {
        alert ("Год должен содержать 4 цифры");
        document.calControl.year.select();
        document.calControl.year.focus();
```

```
    } else { return true; }
  }

  function selectDate() {
    var year =document.calControl.year.value;
    if (isFourDigitYear(year)) {
      var day  =0;
      var month=document.calControl.month.selectedIndex;
      displayCalendar(month, year);
    }
  }

  function setPrevYear() {   // Предыдущий год
    var year =document.calControl.year.value;
    if (isFourDigitYear(year)) {
      var day  =0;
      var month=document.calControl.month.selectedIndex;
      year--;
      document.calControl.year.value=year;
      displayCalendar(month, year);
    }
  }

  function setPrevMonth() {  // Предыдущий месяц
    var year =document.calControl.year.value;
    if (isFourDigitYear(year)) {
      var day  =0;
      var month=document.calControl.month.selectedIndex;
      if (month == 0) {
        month=11;
        if (year > 1000) {
          year--;
          document.calControl.year.value=year;
        }
      } else { month-- }
      document.calControl.month.selectedIndex=month;
      displayCalendar(month, year);
    }
  }
```

```
function setNextMonth() {  // Следующий месяц
  var year =document.calControl.year.value;
  if (isFourDigitYear(year)) {
    var day  =0;
    var month=document.calControl.month.selectedIndex;
    if (month == 11) {
      month=0;
      year++;
      document.calControl.year.value=year;
    } else { month++ }
    document.calControl.month.selectedIndex=month;
    displayCalendar(month, year);
  }
}

function setNextYear() {   // Следующий год
  var year=document.calControl.year.value;
  if (isFourDigitYear(year)) {
    var day=0;
    var month=document.calControl.month.selectedIndex;
    year++;
    document.calControl.year.value=year;
    displayCalendar(month, year);
  }
}

function displayCalendar(month, year) {   // Показать календарь
  month=parseInt(month);
  year=parseInt(year);
  var i=0;
  var days=getDaysInMonth(month+1,year);
  var firstOfMonth=new Date (year, month, 1);
  var startingPos=firstOfMonth.getDay()-1;
  if (startingPos<0) startingPos=6
  days += startingPos;
  document.calButtons.calPage.value =  " Пн Вт Ср Чт Пт Сб Вс";
  document.calButtons.calPage.value += "\n ———————————————————";
  for (i =0; i < startingPos; i++) {
```

```
        if ( i%7 == 0 ) document.calButtons.calPage.value += "\n ";
        document.calButtons.calPage.value += "   ";
      }
      for (i=startingPos; i < days; i++) {
        if ( i%7 == 0 ) document.calButtons.calPage.value += "\n ";
        if (i-startingPos+1 < 10)
          document.calButtons.calPage.value += "0";
        document.calButtons.calPage.value += i-startingPos+1;
        document.calButtons.calPage.value += " ";
      }
      for (i=days; i < 42; i++)  {
        if ( i%7 == 0 ) document.calButtons.calPage.value += "\n ";
        document.calButtons.calPage.value += "   ";
      }
      document.calControl.Go.focus();
    }

    function getDaysInMonth(month,year)  { // Получить день в месяце
      var days;
      if (month==1 || month==3 || month==5 || month==7 || month==8 ||
month==10 || month==12)  days=31;
      else if (month==4 || month==6 || month==9 || month==11) days=30;
      else if (month==2) {
        if (visYear(year)) { days=29; }    // Если год високосный
        else { days=28; }
      }
      return (days);
    }

    function visYear (Year) {  // Високосные года
      if (((Year % 4)== 0) && ((Year % 100)!= 0) || ((Year % 400)== 0)) {
        return (true);
      } else { return (false); }
    }
  </SCRIPT>
  <FORM NAME="calControl" onSubmit="return false">
    <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
      <TR><TD COLSPAN=7>
```

```
        <CENTER>
        <SELECT NAME="month" onchange="selectDate()">
          <OPTION>Январь
          <OPTION>Февраль
          <OPTION>Март
          <OPTION>Апрель
          <OPTION>Май
          <OPTION>Июнь
          <OPTION>Июль
          <OPTION>Август
          <OPTION>Сентябрь
          <OPTION>Октябрь
          <OPTION>Ноябрь
          <OPTION>Декабрь
        </SELECT>
       <INPUT NAME="year" TYPE=TEXT SIZE=4 MAXLENGTH=4
              onchange="selectDate()">
      <INPUT TYPE="button" NAME="Go" value="  OK "
             onclick="selectDate()"
             TITLE="Если изменили год, щелкните здесь">
      </TD>
      </TR>
  </FORM>
  <FORM NAME="calButtons">
    <TR><TD align="center">
<TEXTAREA FONT="Courier"   NAME="calPage" WRAP=no ROWS=8 COLS=24 >
</TEXTAREA>
    </TD>
    <TR><TD>
      <CENTER>
      <INPUT TYPE=BUTTON NAME="PrevYear" VALUE="<<"
             style="font-size:9"
             onclick="setPrevYear()"
             TITLE="Год -">
      <INPUT TYPE=BUTTON NAME="PrevYear" VALUE="< "
             style="font-size:9"
             onclick="setPrevMonth()"
             TITLE="Месяц -">
```

```
      <INPUT TYPE=BUTTON NAME="PrevYear2" VALUE=""
             style="font-size:9"
             onclick="setToday()"
             TITLE="Календарь на текущую дату">
      <INPUT TYPE=BUTTON NAME="PrevYear"
             VALUE=" >"
             style="font-size:9"
             onclick="setNextMonth()"
             TITLE="Месяц +">
      <INPUT TYPE=BUTTON NAME="PrevYear" VALUE=">>"
             style="font-size:9"
             onclick="setNextYear()"
             TITLE="Год +">
      </CENTER>
    </TD></TR>
    </TABLE>
  </FORM>
  <! Запускаем функцию setToday >
  <SCRIPT>setToday()</SCRIPT>
</HTML>
```

В приведенном выше коде в разделе сценария используется доступ к свойствам и методам элементов документа, а также обработка событий `onclick` (щелчок) и `onchange` (изменение содержимого в поле ввода данных). Обработка событий, изменение свойств элементов и ряд других связанных с этим вопросов будут подробно рассмотрены в *главе 3*.

2.12. Объект *Boolean*

Объект `Boolean` (логический) создается с помощью выражения вида:

переменная=new Boolean(*логическое_значение*)

Он имеет свойство `prototype` и методы `toString()` и `valueOf()`, которыми обладают также объекты `String` и `Number`.

Смысл свойства `prototype` мы уже рассматривали применительно к объектам `String` и `Array`. Объект `Boolean` может понадобиться в том случае, когда всем логическим объектам, создаваемым с помощью выражения с ключевыми словами `new Boolean`, нужно добавить новые свойства или методы с помощью прототипа (свойства `prototype`).

2.13. Объект *Function*

2.13.1. Создание объекта *Function*

Ранее мы уже рассматривали стандартный способ определения функции:

```
function имя_функции(параметры) {
    код
}
```

Существует и другой способ, основанный на выражении с ключевыми словами `new Function`. Согласно этому способу функция создается как экземпляр объекта `Function`:

```
имя_функции=new Function(["пар1"[,"парN"], "оператор1[; операторN]")
```

Названия всех параметров являются строковыми значениями. Они разделяются запятыми. Заключительная строка содержит операторы кода тела функции, разделенные точкой с запятой.

Вызов функции, определенной как экземпляр объекта `Function`, можно выполнить обычным способом:

```
имя_функции(параметры)
```

Примеры:

```
Srectangle=new Function("width", "height", "var s=width*height;
                        return s")
Srectangle(2, 3)     // возвращает 6

var expr="var s=width*height; return s"
Srectangle=new Function("width", "height", expr)
Srectangle(2, 3)     // возвращает 6

a="width"
b="height"
expr="var s=width*height; return s"
Srectangle=new Function(a, b, expr)
Srectangle(2, 3)     // возвращает 6
```

При любом задании функции, стандартном или с помощью ключевого слова `new`, автоматически создается экземпляр объекта `Function`, который обладает своими свойствами и методами.

2.13.2. Свойства объекта *Function*

Далее перечислены свойства объекта `Function`.

- `arguments` — массив значений параметров, переданных функции. Индексация элементов массива производится с 0. Поскольку это массив, он имеет свойства и методы объекта `Array` (в частности, свойство `length` — длина массива).

  Свойство `arguments` применяется в теле определения функции, когда требуется проанализировать параметры, переданные ей при вызове. Например, можно узнать, сколько в действительности было передано параметров, не являются ли их значения пустыми (`""`, `0`, `null`) и т. п. Это свойство особенно полезно при разработке универсальных библиотечных функций.

  Синтаксис выражения следующий:

  имя_функции`.arguments`

  Пример. Функция, приводимая в этом примере, возвращает строку, содержащую значения параметров и их общее количество, которые были указаны в вызове функции (а не в ее определении).

```
function myfunc(a, b,c){
  var arglenth=myfunc.arguments.length   /* количество переданных
                                            параметров */
  var x=""
  for (i=0; i< myfunc.arguments.length;i++) {
    x += myfunc.arguments[i] + ","
  }
  return x+"Всего: "+ myfunc.arguments.length
}

myfunc(5, "Вася")        // "5, Вася, Всего: 2"
myfunc()                 // "Всего: 0"
myfunc(null,"",0,25)     // "null,,0,25,всего: 4"
```

- `length` — количество параметров, указанных в определении функции. Синтаксис:

  имя_функции`.length`

  В отличие от свойства `arguments`, количество параметров функции можно определить в программе за пределами тела этой функции.

  Пример:

```
function myfunc(a, b, c, d){
  return myfunc.arguments.length
```

```
}
myfunc(a,b)   // 2
myfunc.length // 4
```

- `caller` — содержит ссылку на функцию, из которой была вызвана данная функция. Если функция не вызывалась из другой функции, то значение этого свойства равно `null`. Совместимость: IE4+, NN4. Синтаксис:

  имя_функции`.caller`

  В свойстве *имя_функции*`.caller` содержится все определение функции, из которой была вызвана функция *имя_функции*.

  Пример:

```
function f1() {
  f2()
}
function f2(){
  alert(f2.caller)   // диалоговое окно со значением f2.caller
}
f1()
```

  Вызов функции `f1()` приведет к вызову функции `f2()`, которая отобразит на экране окно с сообщением, содержащим код определения функции `f1()` (рис. 2.4).

Рис. 2.4. Результат выполнения функции `f1()` — окно со значением `f2.caller`

2.13.3. Методы объекта *Function*

Перечислим методы объекта `Function`.

- `toString()` — возвращает определение функции в виде строки. Синтаксис:

  имя_функции`.toString()`

  Иногда этот метод используют в процессе отладки программ с помощью диалоговых окон.

Пример:

```
function myfunc(a, b){
  return a*b/2
}
alert(myfunc.toString())
```

Результат выполнения последнего выражения показан на рис. 2.5.

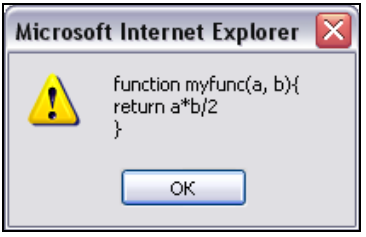


Рис. 2.5. Результат выполнения `alert(myfunc.toString())`

❒ `apply([`*`текущий_объект`*`[, `*`массив_параметров`*`]])`
`call([`*`текущий_объект`*`[, `*`пар1`*`[, `*`пар2`*`[,..., `*`парN`*`]]]])`

Оба метода используются для вызова функции и дают одинаковые результаты. Отличаются они лишь формой представления параметров. Как известно, функцию можно вызвать просто по имени, за которым следует список параметров в круглых скобках. Особенностью этих методов является то, что с их помощью можно вызвать функцию по ссылке на нее. При первом чтении рекомендуется сначала ознакомиться с *разд. 2.14*, а затем вернуться к данной теме, если в этом возникнет необходимость.

Первый параметр обоих методов — ссылка на объект, являющийся текущим для данной функции. Ссылка на объект используется, когда вызываемая функция определена как метод пользовательского объекта. Более подробно эта тема обсуждается в *разд. 2.14*.

Пример.

```
/* Функция-конструктор объекта car (автомобиль)*/
function car(name, model, color) {
  this.name=name      // название
  this.model=model    // модель
  this.color=color    // цвет
  this.show=showcar   // метод
}
/* Функция, вызываемая как метод объекта car */
function showcar(){
```

```
  alert(this.name + "-" + this.model)   // окно с сообщением
}
/* Создание экземпляра объекта car */
mycar=new car("Лада", "ВАЗ21099","green")
```

Обычный способ применения метода `show` к объекту `car` (вывод окна с сообщением) выглядит так:

```
mycar.show()
```

При этом в действительности вызывается функция `showcar()`, заданная для этого метода. Текущий объект `car` рассматривается как контекст для ссылок `this`, указанных в теле функции.

Однако методы `apply()` и `call()` позволяют не связывать функцию `showcar()` с объектом `mycar`. Более того, в конструкторе объекта `car` можно не указывать выражение, определяющее функцию `showcar` в качестве метода объекта. Вместо этого можно вызвать метод `showcar` как объект и применить к нему метод `call()`, указав в качестве параметра `mycar` как текущий объект:

```
showcar.call(mycar)
```

С помощью метода `call()` можно передавать параметры, отделенные друг от друга запятыми. Если параметры определены как элементы массива, то вместо `call()` используется метод `apply()`.

2.14. Пользовательские объекты

Рассмотренные в предыдущих разделах объекты `String`, `Array`, `Number`, `Date`, `Boolean` и `Function` являются встроенными объектами JavaScript. Однако вы можете создать и свои объекты. Объект представляет собой удобное средство организации данных и функций. Это контейнер, содержащий переменные, называемые свойствами, и функции, называемые методами. Применение объекта в программах состоит в манипулировании его свойствами и методами.

2.14.1. Создание объекта

Объекты в JavaScript можно создать несколькими способами. Мы рассмотрим три таких способа:

- способ 1

```
function имя_конструктора([пар1,...[, парN]){
    код
   }
имяОбъекта=new имя_конструктора(["пар1",...[, "парN"])
```

- способ 2

  имяОбъекта=new Object()

  имяОбъекта.свойство=значение

- способ 3

  имяОбъекта={свойство1: значение1 [, свойство2: значение2 [,...,N]}

Для обращения к свойствам и методам объекта используется следующий синтаксис:

ссылка_на_объект.свойство

ссылка_на_объект.метод([параметры])

Рассмотрим указанные выше способы создания объектов более подробно.

Первый способ основан на функции, в теле которой описываются все свойства и методы создаваемого объекта. Поскольку эта функция играет определяющую роль в создании объекта, ее называют *функцией-конструктором* или просто *конструктором* объекта. Что, собственно, должен делать конструктор? Очевидно, он должен ввести имя создаваемого объекта, а также его свойства и методы. Кроме того, он должен допускать возможность присваивания начальных значений свойствам. Имя функции-конструктора объекта является одновременно и именем создаваемого объекта. Свойства и методы создаваемого объекта задаются в теле функции-конструктора с помощью операторов присваивания. При этом имена переменных-свойств записываются с ключевым словом `this` (этот):

this.*переменная*

Рассмотрим в качестве примера функцию-конструктор, определяющую объект `car` (автомобиль) со свойствами `name` (название), `model` (модель) и `color` (цвет):

```
function car(name, model, color) {
  this.name=name
  this.model=model
  this.color=color
}
```

Эта функция определяет объект `car`. Чтобы создать конкретный экземпляр объекта `car`, следует выполнить выражение с вызовом этой функции, которой можно передать значения параметров. Мы уже знаем, что экземпляр объекта создается с помощью оператора присваивания с ключевым словом `new`:

```
mycar=new car("Лада", "ВАЗ21099", "green")
```

Итак, мы создали объект `mycar`, являющийся экземпляром объекта `car`. Таких экземпляров с различными именами можно создать несколько. Значения свойств объекта `mycar` можно изменять в программе:

```
mycar.model             // значение равно "ВАЗ21099"
mycar.name              // значение равно "Лада"
mycar.model="ВАЗ2110"   // значение равно "ВАЗ2110"
```

Как видите, объект — это просто особый способ группировки данных и их использования (составное имя переменной: *`объект.свойство`*).

Второй способ основан на использовании конструктора `new Object()`:

```
mycar=new Object()
mycar.name="Лада"
mycar.model="ВАЗ21099"
mycar.color="green"
```

В этом способе создаваемый объект является экземпляром встроенного объекта `Object` подобно тому, как, например, создаваемый массив является экземпляром объекта `Array`. Объект `Object` является корневым для всех ранее рассмотренных встроенных объектов.

В третьем способе применяется следующая компактная запись определения объекта:

```
mycar={name:"Лада", model:"ВАЗ21099", color:"green"}
```

Здесь все определение свойств заключается в фигурные скобки. Пары *`свойство:значение`* разделяются запятыми, а имя свойства отделяется от значения двоеточием.

При создании объекта мы можем задать значения свойств по умолчанию, т. е. значения, которые будут иметь свойства, если при создании экземпляра этого объекта значения его свойств не указаны явным образом (т. е. имеют значения `null`, 0 или `""`). Это делается с помощью логического оператора ИЛИ (обозначаемого `||`) в конструкторе объекта в виде функции:

```
function car(name, model, color) {   // конструктор объекта car
  this.name=name || "неизвестно"
  this.model=model || "неизвестно"
  this.color=color || "black"
}
mycar=new car("Жигули","")   // создание экземпляра mycar объекта car
mycar.name                   // "Жигули"
mycar.model                  // "неизвестно"
mycar.color                  // "black"
```

2.14.2. Добавление свойств

Если возникает необходимость добавить новое свойство к существующему объекту, а также ко всем его экземплярам, то это можно сделать с помощью свойства `prototype`. В приведенном далее примере мы создаем объект `car` (автомобиль), затем его экземпляр `mycar`, а потом добавляем к свойству `prototype` свойство `owner` (владелец) с конкретным значением:

```
function car(name, model, color) {   // конструктор объекта car
  this.name=name || "неизвестно"
  this.model=model || "неизвестно"
  this.color=color || "black"
}

mycar=new car("Жигули","")     /* создание экземпляра mycar объекта car */
mycar.name                     // "Жигули"
mycar.model                    // "неизвестно"
mycar.color                    // "black"
car.prototype.owner="Иванов"   // добавляем новое свойство
mycar.owner                    // "Иванов"
```

Если нужно добавить новое свойство только к конкретному объекту (к данному экземпляру объекта), то это можно сделать просто с помощью оператора присваивания:

имяОбъекта.новое_свойство=значение

В следующем примере мы создаем объект `car` вообще без свойств, а затем формируем его экземпляры и добавляем к ним различные свойства.

```
function car(){}
mycar1=new car()
mycar2=new car()
mycar1.name="Жигули"
mycar2.model="ВАЗ2106"

mycar1.name       // "Жигули"
mycar1.model      // undefined (не определено)
mycar2.name       // undefined (не определено)
mycar2.model      // "ВАЗ2106"
```

2.14.3. Связанные объекты

Объект по желанию его создателя может быть усложнен настолько, чтобы в качестве одного или нескольких своих свойств содержать другие объекты. Иначе говоря, в объекте в виде свойства может содержаться ссылка на другой

объект. В этом случае оба объекта оказываются *связанными*: один из них оказывается подобъектом или, иными словами, свойством другого. Например, мы можем создать объект `photo`, содержащий в качестве своих свойств название автомобиля и URL-адрес файла с его изображением. Такой объект можно связать с объектом `car`, содержащим основную информацию об автомобилях, добавив к нему свойство, значением которого является ссылка на объект `photo`.

Ниже приведены конструкторы объектов `car` и `photo`. Чтобы добавить объект `photo` в объект `car`, мы добавили в объект `car` свойство, значением которого является ссылка на объект `photo`.

```
function car(name, model, color, photo) {  /* конструктор объекта car */
  this.name=name
  this.model=model
  this.color=color
  this.photo=photo // ссылка на объект photo
}

function photo (name, url) {               // конструктор объекта photo
  this.name=name
  this.url=url
}
```

Создадим пару конкретных объектов, являющихся экземплярами объекта `photo`:

```
photo1=new photo("Лада","/images/lada1.jpg")
photo2=new photo("Лада","/images/lada2.gif")
```

Теперь создадим экземпляры объекта `car`:

```
mycar=new car("Лада", "ВАЗ21099","white", photo1)
mycar.photo.url      // "/images/lada1.jpg"
mycar.model          // "ВАЗ21099"

mycar=new car("Лада", "ВАЗ21099","white", photo2)
mycar.photo.url      // "/images/lada2.gif"
mycar.model          // "ВАЗ21099"
```

Обратите внимание, чтобы узнать значение URL-адреса расположения файла с картинкой, относящейся к объекту `mycar`, мы обращаемся к свойству `url` объекта `photo`, который сам является свойством объекта `mycar` или, другими словами, подобъектом объекта `mycar`.

2.14.4. Пример создания базы данных с помощью объектов

Базы данных, предназначенные для хранения и быстрого доступа к данным, обычно создаются средствами СУБД (системами управления базами данных), такими как SQL Server, MySQL, PostgreSQL и др. Эти средства применяются, когда база данных должна содержать тысячи и более записей. Однако это — другая тема. Здесь мы рассмотрим пример базы данных, которая вполне годится в случае нескольких сотен записей. Она может быть сформирована средствами языка JavaScript, а именно путем создания объекта, содержащего несколько строк, каждая из которых имеет несколько полей. Разумеется, эта база данных не сохраняется на жестком диске вашего компьютера, а создается динамически в результате выполнения сценария. Она существует и доступна, пока документ, содержащий генерирующий ее скрипт, загружен в браузер. Сохранение изменений в базе данных, воспроизводимых в последующих сеансах связи с ней, — это другая тема, к которой мы обратимся позже. А сейчас мы находимся в пространстве оперативной памяти компьютера и все, что мы инициализируем, будет существовать до тех пор, пока HTML-документ со скриптом загружен в браузер.

Пусть необходимо создать базу данных автомобилей, находящихся на одной стоянке. Список всех автомобилей можно представить в виде таблицы со столбцами: `name` (название), `model` (модель), `regnum` (номер), `owner` (владелец) и `photo` (фотография). Возможно, вы захотите добавить еще какие-нибудь столбцы. Сделайте это по аналогии. Будем считать, что на фотографии номер автомобиля не показан, так что она отображает не конкретный автомобиль, а его тип (название, модель и цвет). Конечно, можно создать базу данных в виде одной таблицы, однако мы поступим иначе и создадим две таблицы. Одна из них будет вспомогательной и играть роль справочника типов автомобилей, а другая таблица будет содержать записи о конкретных автомобилях, находящихся на стоянке или, точнее, приписанных к ней.

Справочник типов автомобилей будет содержать наиболее общие сведения об автомобилях: название, модель и фотографию (точнее, URL-адрес ее файла). На стоянке может находиться несколько автомобилей одного типа (например, "Жигули", ВАЗ2101 белого цвета), но в справочнике этот тип будет упомянут лишь один раз. В нашей базе данных справочник типов автомобилей будет представлять объект `ref`. Этот объект мы создаем с помощью функции-конструктора:

```
function ref(name, model, url) { /* конструктор справочника
                                    типов автомобилей */
this.name=name        // название марки автомобиля
```

```
this.model=model      // модель
this.url=url          // URL-адрес файла с фотографией
}
```

Заполним справочник конкретными записями. Эта операция аналогична заполнению строк таблицы. Несмотря на простоту аналогии, будьте внимательны. Множество всех записей, добавляемых в объект (таблицу), удобно определить как массив (назовем его `aref`). Тогда элементы этого массива определим как экземпляры объекта `ref`:

```
aref=new Array()  /* массив записей справочника типов автомобилей */

aref[0]=new ref("Жигули", "Ваз2101", "pict0.gif")
aref[1]=new ref("Жигули", "Ваз2106", "pict1.gif")
aref[2]=new ref("Волга", "ГАЗ24", "pict2.gif")
aref[3]=new ref("Лада", "Ваз21099", "pict3.gif")
```

В нашем справочнике описаны только четыре типа автомобилей. Но это не предел. Вы можете добавить сколько угодно аналогичных записей. Разумеется, предел существует и обусловлен физическими ограничениями аппаратной мощности вашего компьютера.

Теперь создадим список всех автомобилей на стоянке. Этот список удобно оформить как массив, каждый элемент которого представляет запись о конкретном автомобиле. Формально элементы этого массива мы определим как экземпляры объекта `car`, который будет содержать ссылку на справочник — объект `ref` .

Итак, сначала напишем код функции-конструктора объекта `car`:

```
function car(regnum, owner, ref) {
  this.regnum=regnum   // номер автомобиля
  this.owner=owner     // владелец
  this.ref=ref         // ссылка на справочник
}
```

Далее, создадим записи о конкретных автомобилях на стоянке. Массив записей назовем `acar`.

```
acar=new Array()
acar[0]=new car("А123ВХ", "Иванов", aref[1])
acar[1]=new car("М345СТ", "Петров", aref[1])
acar[2]=new car("Е678СА", "Сидоров", aref[0])
acar[3]=new car("К056ОХ", "Михайлов", aref[2])
acar[4]=new car("о089КС", "Дунаев", aref[3])
```

```
acar[5]=new car("P340HY", "Павлов", aref[2])
acar[6]=new car("O321OK", "Николаев", aref[2])
```

В нашем списке всего 7 записей. Модель ВАЗ 2106 в нем встречается два раза, ГАЗ 24 — три раза, а остальные — по одному разу. Если бы мы создавали базу данных без справочника, то нам пришлось бы многократно повторять одни и те же данные при определении массива `acar`.

Ниже приводятся примеры обращения к свойствам нашей базы данных:

```
acar[4].ref.model   // "ВАЗ21099"
acar[3].ref.name    // "Волга"
acar[3].owner       // "Михайлов"
```

Поскольку база данных представляет собой массив, то с помощью методов объекта `Array` можно организовать фильтрацию (выборку) записей по тому или иному критерию. Попробуйте сделать это самостоятельно.

Базу данных, создание которой мы рассмотрели выше, можно отобразить в окне браузера. Для этого необходимо, прежде всего, создать HTML-программу с таким элементом, как таблица. Напомним, что таблица определяется тегами `<TABLE>`, `<TR>`, `<TH>`, `<TD>` и др. Далее, вы можете создать в HTML-программе раздел, обрамленный тегами `<SCRIPT>` и `</SCRIPT>`, между которыми разместить рассмотренный выше код на языке JavaScript. А далее есть два пути.

- Первый путь: вручную написать теги таблицы. Если количество строк в таблице не велико, то именно так и следует поступать.
- Второй путь: написать программу на JavaScript, генерирующую нужную последовательность тегов и данных как строку, а затем воспользоваться методом `write()` объекта `document` для ее записи в текущий HTML-документ и исполнения браузером. Этот способ интереснее и предпочтительней, если таблица содержит много строк. Небольшой объем кода обеспечивается тем, что формирование строки с HTML-кодом происходит в цикле. К этому располагает то, что строки (записи) нашей базы данных определены в виде массива.

В листинге 2.5 приведен HTML-код, содержащий только сценарий, который делает все. Для упрощения ситуации формируется таблица только с тремя столбцами и не обращается внимание на дизайн таблицы, который вы можете определить специальными атрибутами тегов (рис. 2.6).

Листинг 2.5. Код для отображения базы данных автомобилей в браузере

```
<HTML>
  <HEAD><TITLE>База данных автомобилей</TITLE></HEAD>
  <SCRIPT>
```

```
/* Конструкторы объектов */
function ref(name, model, url) {     /* конструктор справочника
                                        типов автомобилей */
  this.name=name          // название марки автомобиля
  this.model=model        // модель
  this.url=url            // URL-адрес файла с фотографией
}
function car(regnum, owner, ref) {   /* конструктор списка
                                        автомобилей на стоянке */
  this.regnum=regnum      // номер автомобиля
  this.owner=owner        // владелец
  this.ref= ref           // ссылка на справочник
}

/* Собственно база данных */
aref=new Array()   /* массив записей справочника типов автомобилей */
aref[0]=new ref("Жигули", "Ваз2101", "pict0.gif")
aref[1]=new ref("Жигули", "Ваз2106", "pict1.gif")
aref[2]=new ref("Волга", "ГАЗ24", "pict2.gif")
aref[3]=new ref("Лада", "Ваз21099", "pict3.gif")

acar=new Array()
acar[0]=new car("А123ВХ", "Иванов", aref[1])
acar[1]=new car("М345СТ", "Петров", aref[1])
acar[2]=new car("Е678СА", "Сидоров", aref[0])
acar[3]=new car("К056ОХ", "Михайлов", aref[2])
acar[4]=new car("о089КС", "Дунаев", aref[3])
acar[5]=new car("Р340НУ", "Павлов", aref[2])
acar[6]=new car("О321ОК", "Николаев", aref[2])

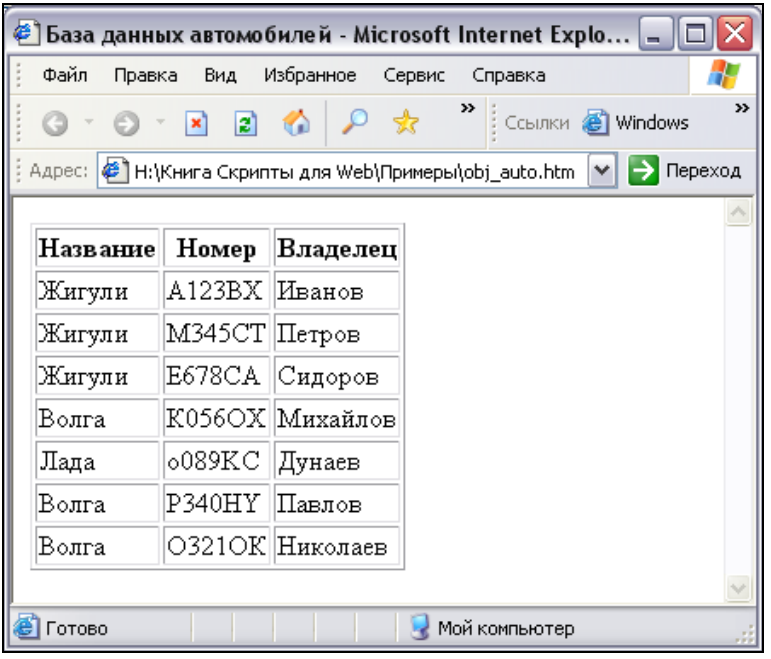
strTab="<TABLE BORDER=1> <TR>"
strTab += "<TH>Название</TH><TH> Номер</TH> <TH>Владелец</TH></TR>"

/* Формирование строк таблицы */
for(i=0;i<=acar.length-1;i++){
  strTab += "<TR><TD>" + acar[i].ref.name +
            "</TD><TD>" +acar[i].regnum
  strTab += "</TD><TD>" + acar[i].owner + "</TD></TR>"
```

```
    }
    strTab += "</TABLE>"
    document.write(strTab)    /* записываем строку strTab в HTML-документ
                                 и выполняем его */
  </SCRIPT>
</HTML>
```

Примечание

Здесь в теле оператора цикла `for` мы использовали разновидность оператора присваивания `+=`. Напомним, что выражение `x+=y` эквивалентно выражению `x=x+y`. Мы дважды вставили выражение с этим оператором только для того, чтобы выражение справа от него не было слишком длинным.



Название	Номер	Владелец
Жигули	A123BX	Иванов
Жигули	M345CT	Петров
Жигули	E678CA	Сидоров
Волга	K056OX	Михайлов
Лада	o089KC	Дунаев
Волга	P340HY	Павлов
Волга	O321OK	Николаев

Рис. 2.6. Так выглядит база данных автомобилей в окне браузера

Мы могли бы включить в отображаемую таблицу и другие столбцы. Например, для вывода фотографий автомобилей достаточно добавить в строку `strTab` следующий фрагмент:

```
"<TD><IMG SRC='" + acar[i].ref.url + "'></TD>"
```

2.15. Специальные операторы

Рассмотрим операторы, которые при программировании на JavaScript используются относительно редко.

2.15.1. Побитовые операторы

Побитовые (поразрядные) операторы применяются к целочисленным значениям и возвращают целочисленные значения. При их выполнении операнды предварительно приводятся к двоичной форме представления, в которой число является последовательностью из нулей и единиц длиной 32. Эти нули и единицы называются *двоичными разрядами* или *битами*. Далее производится некоторое действие над битами, в результате которого получается новая последовательность битов. В конце концов, эта последовательность битов преобразуется к обычному целому числу — результату побитового оператора. В табл. 2.10 приведен список побитовых операторов.

Таблица 2.10. *Побитовые операторы*

Оператор	Название	Левый операнд	Правый операнд
`&`	Побитовое И	Целое число	Целое число
`|`	Побитовое ИЛИ	Целое число	Целое число
`^`	Побитовое исключающее ИЛИ	Целое число	Целое число
`~`	Побитовое НЕ	—	Целое число
`<<`	Смещение влево	Целое число	Количество битов, на которое производится смещение
`>>`	Смещение вправо	Целое число	Количество битов, на которое производится смещение
`>>>`	Заполнение нулями при смещении вправо	Целое число	Количество битов, на которое производится смещение

Операторы `&`, `|`, `^` и `~` напоминают логические операторы, но их область действия — биты, а не логические значения. Оператор `~` изменяет значение бита на противоположное: 0 на 1, а 1 — на 0. В табл. 2.11 поясняется, как работают операторы `&`, `|`, `^`.

Таблица 2.11. *Действие операторов* `&`, `|`, `^`

X	Y	X&Y	X|Y	X^Y
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Например, `2&3` равно 2, а `2|3` равно 3. Действительно, в двоичном представлении 2 есть 10, а 3 — 11. Применение побитовых операторов даст в случае оператора `&` двоичное число 10, т. е. десятичное число 2, а в случае оператора `|` — двоичное число 11, т. е. десятичное 3.

У операторов смещения один операнд и один параметр, указывающий, на какое количество битов следует произвести смещение. Например, `3<<2` равно 12, потому что смещение влево на два бита двоичного числа 11 (десятичное 3) дает 1100, что в десятичной форме есть 12. Результат вычисления выражения `6>>2` равен 1. Действительно, число 6 в двоичной форме — это 110; смещение его вправо на два бита дает 1 как в двоичной, так и в десятичной форме.

2.15.2. Объектные операторы

Оператор удаления свойств объекта *delete*

Удалить свойство объекта, а также элемент массива можно с помощью оператора `delete`. Обычно этот оператор используют для удаления элементов массива:

```
delete элемент
```

Внимание

При удалении элемента массива удаляется и его индекс, но оставшиеся элементы сохраняют свои прежние индексы, а длина массива не изменяется.

Пример:

```
myarray=new Array("a","b","c","d")
myarray.length     // 4
delete myarray[1]
myarray.length     // 4
```

```
myarray[0]          // "a"
myarray[1]          // undefined
myarray[2]          // "c"
myarray[3]          // "d"
```

Использование оператора `delete` не приводит к немедленному освобождению памяти. Решение об освобождении памяти принимается так называемым *ядром JavaScript*, а пользовательская программа лишь создает к этому предпосылки, но не может контролировать данный процесс абсолютно.

Оператор проверки наличия свойств *in*

Этот оператор позволяет проверить, имеется ли некоторое свойство или метод у того или иного объекта. Левый операнд представляет собой ссылку в виде строки на интересующее нас свойство или метод, а правый операнд — объект. Ссылка на метод содержит лишь его название без круглых скобок. Если свойство или метод содержится в объекте, то возвращается `true`, иначе — `false`. Отсюда следует, что оператор `in` можно применять в условных выражениях (в операторах `if`, `switch`, `for`, `while`, `do-while`).

Рассмотрим примеры.

- Объект `document`, представляющий загруженный в браузер HTML-документ, имеет метод `write()`. Чтобы убедиться в этом, следует написать выражение:

  ```
  "write" in document   // значение равно true
  ```

- Создадим объект и проверим наличие в нем некоторых свойств.

  ```
  function Сотрудник(Имя, Отдел, Телефон, Зарплата) {
    this.Имя=Имя
    this.Отдел=Отдел
    this.Телефон=Телефон
    this.Зарплата=Зарплата
  }
  agent007=new Сотрудник("Джеймс Бонд", 5, "223-332")
  "Телефон" in agent007         // true
  "Ученая степень" in agent007  // false
  ```

Оператор `in` поддерживается браузерами IE5.5+, NN6+.

Оператор проверки принадлежности объекта модели *instanceof*

Этот оператор позволяет проверить, принадлежит ли некоторый объект объектной модели JavaScript. Левый операнд представляет проверяемое значение, а правый — ссылку на корневой объект, такой как `Array`, `String`, `Date` и т. п. Выражение с оператором `instanceof` возвращает `true` или `false` и, таким образом, может использоваться в условных выражениях (в операторах `if`, `switch`, `for`, `while`, `do-while`).

Пример. Созданный массив является экземпляром объекта `Array`, а последний, в свою очередь, — экземпляром корневого объекта `Object`.

```
myarray=new Array()
myarray instanceof Array    // true
Array instance Object       // true
Myarray instanceof String   // false
```

Оператор `instanceof` поддерживается браузерами IE5.5+ и NN6+.

Глава 3

Клиентские сценарии на JavaScript

В этой главе мы рассмотрим основные принципы и примеры создания клиентских сценариев на языке JavaScript для решения типичных задач разработки Web-сайтов. Сценарии работают с так называемой объектной моделью браузера и загруженного в него документа. Поэтому, чтобы разработать какой-либо сценарий, необходимо знать, по крайней мере, основные элементы объектной модели. Краткое описание основных объектов дано в *приложении 1*.

3.1. Объекты, управляемые сценариями

Параметры элементов документа, заданные с помощью атрибутов соответствующих тегов и таблиц стилей, можно изменить. Более того, можно заменить одни теги другими и даже заменить весь загруженный HTML-документ на другой. Делается это сценариями, но не напрямую с тегами и значениями атрибутов (т. е. с HTML-кодами), а с представляющими их объектами.

Окну браузера соответствует объект `window`, а HTML-документу, загруженному в окно, — объект `document`. Эти объекты содержат в своем составе другие объекты. В частности, объект `document` входит в состав объекта `window`. Элементам HTML-документа соответствуют объекты, которые входят в состав объекта `document`. Все множество объектов имеет иерархическую структуру, называемую *объектной моделью*. В *главе 2* мы уже встречались со встроенными объектами JavaScript и объектами, создаваемыми пользователем. Напомним, что объект представляет собой своего рода контейнер для хранения информации. Он характеризуется свойствами, методами, а также событиями, на которые может реагировать.

Доступ к свойствам и методам объекта осуществляется с помощью выражений вида:

объект.свойство

объект.метод()

Объекты могут находиться в отношении вложенности (подчиненности). Объект, содержащий в себе другой объект, называют *родительским*. Объект, который содержится в каком-нибудь объекте, называют *дочерним* по отношению к нему. Таким образом, устанавливается иерархия. Чтобы указать конкретный объект, требуется перечислить все содержащие его объекты, начиная с объекта самого верхнего иерархического уровня, подобно тому, как указывается полный путь к файлу на диске:

объект1.объект2. ... объектN

Если объект входит в состав другого объекта (является подобъектом другого), то для доступа к его свойствам и методам используют следующий синтаксис:

объект1.объект2. ... объектN.свойство

объект1.объект2. ... объектN.метод()

Нередко подобъект некоторого объекта называют его *свойством* (так сказать, сложным свойством). В этом случае можно говорить, что свойства объектов бывают трех типов:

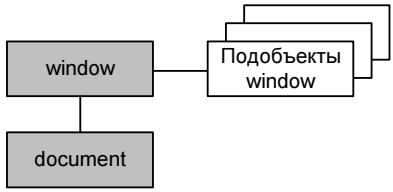
- просто свойства (простые свойства);
- методы (свойства-функции);
- объекты (сложные свойства, имеющие свои свойства).

Поскольку объект `document` является подобъектом объекта `window`, то ссылка на HTML-документ, загруженный в текущее (активное) окно браузера, будет выглядеть следующим образом: `window.document`. Объект `document` имеет метод `write(строка)`, позволяющий записать в текущий HTML-документ строку, содержащую просто текст или теги HTML. Чтобы применить этот метод, следует записать: `window.document.write(строка)`.

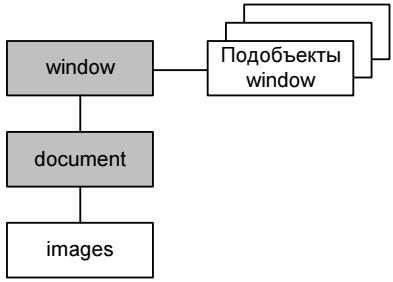
Как уже отмечалось выше, при загрузке HTML-документа в браузер создается объектная модель этого документа. Рассмотрим в общих чертах, на простых примерах, что именно происходит. Прежде всего, создается объект окна `window`. Это корневой объект, имеющий свои подобъекты, такие как `location` для хранения информации об URL-адресе загруженного документа и `screen` для хранения данных о возможностях экрана монитора пользователя. Затем создается объект `document`, являющийся подобъектом `window`. Далее форми-

руются объекты, представляющие некоторые отдельные элементы HTML-документа, такие как объекты графических изображений, форм и их элементов (поля ввода, переключатели, кнопки) и др.

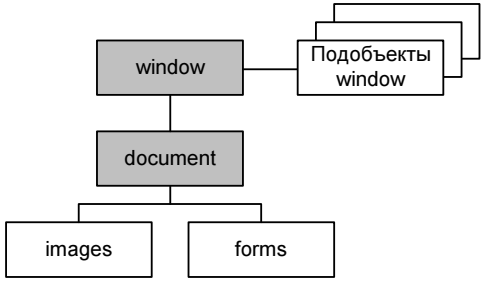
```
<HTML>
  <H1>Моя Web-страница</H1>
</HTML>
```




```
<HTML>
  <H1>Моя Web-страница</H1>
  <IMG SRC = "pict.jpg">
</HTML>
```




```
<HTML>
  <H1>Моя Web-страница</H1>
  <IMG SRC = "pict.jpg">
  <FORM>
  </FORM>
</HTML>
```




```
<HTML>
  <H1>Моя Web-страница</H1>
  <IMG SRC = "pict.jpg">
  <FORM>
    <INPUT  TYPE = "text"  VALUE = "">
    <BUTTON>Нажми здесь</BUTTON>
  </FORM>
</HTML>
```

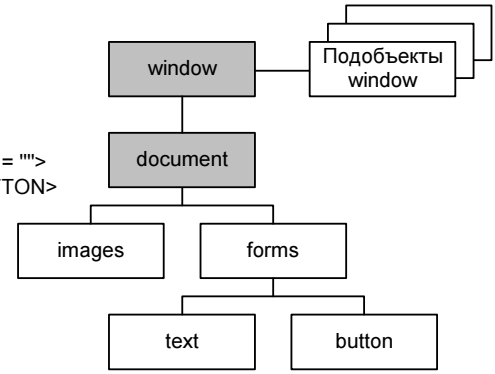


Рис. 3.1. Структуры множеств объектов для различных HTML-документов

На рис. 3.1 показано, как усложняется структура объектного представления документа с добавлением новых тегов. Это упрощенные схемы, дающие лишь самые общие представления об объектной модели. Сведения об объектной модели играют роль своего рода путеводителя в обширном множестве объектов документа и окна браузера.

В объектной модели документа объекты сгруппированы в так называемые *коллекции*. Коллекцию можно рассматривать как промежуточный объект, предназначенный для группировки объектов собственно документа. С другой точки зрения, коллекция является массивом объектов, отсортированных в порядке упоминания соответствующих им элементов в HTML-документе. Индексация объектов в коллекции начинается с нуля. Синтаксис обращения к элементам коллекции такой же, как к элементам массива. Коллекция даже имеет свойство `length` — количество всех ее элементов. Так например, коллекция всех объектов графических изображений в документе называется `images`, коллекция всех форм — `forms`, коллекция всех ссылок — `links`. Это примеры так называемых *частных* или *тематических коллекций*. Кроме них имеется коллекция всех объектов документа, которая называется `all`. Один и тот же объект может входить в частную коллекцию (например, `images`), но обязательно входит в коллекцию `all`. При этом индексы данного объекта в частной коллекции и коллекции `all` могут быть различными.

Рассмотрим способы обращения к свойствам объектов документа. Общее правило заключается в том, что в ссылке должно быть упомянуто название коллекции. Исключением из этого правила является объект формы. Далее, если речь идет о документе, загруженном в текущее окно, то объект `window` можно не упоминать, а сразу начинать с ключевого слова `document`. Возможны несколько способов обращения к объектам документа:

`document.`*`коллекция`*`.`*`id_объекта`*

`document.`*`коллекция`*`["`*`id_объекта`*`"]`

`document.`*`коллекция`*`[`*`индекс_объекта`*`]`

Здесь *`id_объекта`* — значение атрибута ID в теге, который определяет соответствующий элемент в HTML-документе. Величина *`индекс_объекта`* — целое число, указывающее порядковый номер объекта в коллекции. Первый объект в коллекции имеет индекс 0. Если при создании HTML-документа вы не использовали атрибут `ID` для некоторых элементов, то для обращения к их объектам остается только взять индексы.

Заметим, что некоторые теги (например, `<FORM>`, `<INPUT>`) имеют атрибут `NAME`. Значение этого атрибута можно использовать при обращении к объекту наравне со значением атрибута `ID`.

К объекту формы, кроме описанных выше способов, можно обращаться по значению атрибута `NAME` (имя формы), если он указан в теге `<FORM>`, но не по значению атрибута `ID`:

`document.`*`имя_формы`*

Как известно, тег формы `<FORM>` является контейнерным и может содержать в себе теги других элементов, такие как `<INPUT>`, `<BUTTON>` и др. Обращение к элементам формы возможно через коллекцию `all`. Однако имеется и специфический для них способ:

`document.`*`имя_формы`*`.elements[`*`индекс_элемента`*`]`

`document.`*`имя_формы`*`.elements[`*`id_элемента`*`]`

`document.`*`имя_формы`*`.`*`id_элемента`*

Здесь *`индекс_элемента`* — порядковый номер элемента формы, а не порядковый номер в коллекции всех элементов; первый элемент имеет индекс 0. Заметим, что для Internet Explorer вместо квадратных скобок допустимо использование и круглых скобок.

Обобщая вышеизложенное, отметим, что универсальный способ обращения к объектам документа — обращение посредством коллекции `all`. Однако если вы не позаботились об атрибуте `ID` в тегах элементов HTML-документа, которые должны стать предметами работы ваших сценариев, то обращение к объектам через `all` посредством индексов, особенно в случае объемистого документа, становится проблематичным. Чтобы убедиться в этом, внимательно отнеситесь к следующей задаче.

Предположим, у нас есть некоторый HTML-документ. Интересно, какие именно теги этого документа соответствуют элементам коллекции `all` объекта `document`? Чтобы ответить на этот вопрос, я написал простой сценарий и вставил его в конце тестируемого HTML-документа. Этот сценарий основан на использовании свойства `tagName`, возвращающего название тега, с помощью которого был создан соответствующий объект. Код сценария приведен в листинге 3.1.

Листинг 3.1. Тестируемый HTML-код вместе с тестирующим сценарием

```
<HTML>
  <H1>Моя Web-страница</H1>
  <IMG SRC="pict.jpg">
  <A HREF="www.admiral.ru/~dunaev">
    Сайт автора www.admiral.ru/~dunaev
  </A>
```

```
  <FORM>
    <INPUT TYPE="text" VALUE="">
    <P>
    <BUTTON onclick="my()">
      Нажми здесь
    </BUTTON>
  </FORM>
  <SCRIPT>
    msg=""
    for(i=0; i < document.all.length; i++){
      msg += i + " " + document.all[i].tagName + "\n"
    }
    alert(msg)   // Вывод перечня элементов
  </SCRIPT>
</HTML>
```

В результате выполнения этого HTML-кода окно браузера будет выглядеть, как показано на рис. 3.2. В диалоговом окне, выведенном с помощью метода `alert()`, перечислены теги HTML-документа. Их порядковые номера соответствуют индексам в коллекции `all`.

Рис. 3.2. В диалоговом окне перечислены теги HTML-документа и их индексы в коллекции `all`

Например, объекту `document.all[5]` соответствует элемент HTML-документа, созданный тегом `<IMG>`. Обратите внимание, что хотя теги `<HEAD>`, `<TITLE>` и `<BODY>` в нашем документе явно не упоминаются, соответ-

ствующие им объекты присутствуют в объектной модели этого документа и, следовательно, в коллекции `all`. Кроме того, в коллекцию `all` входит элемент, соответствующий тегу `<HTML>`.

Теперь рассмотрим тот же пример, но добавим к основным тегам атрибуты `ID`, с помощью которых можно персонифицировать соответствующие объекты. Как и раньше, мы добавим к HTML-документу сценарий, выводящий окно с информацией о тегах. При этом кроме порядкового номера и названия тега будем выводить и значение атрибута `ID` (рис. 3.3).

Рис. 3.3. В диалоговом окне перечислены теги HTML-документа, их индексы в коллекции `all`, а также значения атрибутов `ID`

Чтобы получить значение атрибута `ID`, необходимо воспользоваться свойством `id` соответствующего объекта. HTML-код приведен в листинге 3.2.

Листинг 3.2. Тестируемый код с добавлением атрибутов `ID`

```
<HTML>
  <H1>Моя Web-страница</H1>
  <IMG ID="myimage" SRC="pict.jpg">
  <A ID="myref" HREF="www.admiral.ru/~dunaev">
    Сайт автора www.admiral.ru/~dunaev
  </A>
  <FORM ID="myform">
    <INPUT ID="myinput" TYPE="text" VALUE ="qwerty">
    <P>
    <BUTTON ID="mybutton" onclick="myfunc()">
      Нажми здесь
```

```
    </BUTTON>
  </FORM>
  <SCRIPT>
    msg=""
    for(i =0; i < document.all.length; i++){
      msg+= i + " " + document.all[i].tagName + " id="+ document.all[i].id + "\n"
    }
    alert(msg) // Вывод перечня элементов
  </SCRIPT>
</HTML>
```

Из коллекции всех элементов документа можно выделить некоторое его подмножество однотипных элементов по названию тега. Для этого служит метод

`tags(`*`название_тега`*`)`

В следующем примере выделяется подмножество всех заголовков 1-го уровня, созданных с помощью тегов `<H1>`, и для каждого из них устанавливается красный цвет:

```
var x=document.all.tags("H1")
for (i=0; i < x.length; i++)
  x(i).style.color='red'
```

Итак, универсальный способ обращения к объекту документа базируется на использовании коллекции `all`. Однако кроме него в IE5+ и NN6 существует еще один способ, основанный на методе `getElementById()`:

`document.getElementById(`*`значение_ID`*`)`

Метод `getElementById(`*`значение_ID`*`)` позволяет обратиться к любому элементу по значению его идентификатора — значению атрибута `ID`. Если несколько элементов документа имеют одинаковый `ID`, метод возвращает первый элемент с указанным значением `ID`. Отсюда видно, насколько важно присваивать различным элементам уникальные значения `ID`.

Выше мы рассмотрели различные способы обращения к объектам. Ссылки на объекты часто используются главным образом как промежуточный этап для доступа к конечным свойствам и методам. В конце концов, именно они и интересуют нас при создании сценариев. Заметим попутно, что ссылки на объекты можно сохранять в переменных, которые потом легко использовать для обращения к свойствам и методам.

В приведенном выше примере (см. листинг 3.2) элемент изображения, созданный с помощью тега `<IMG ID="myimage" SRC="pict.jpg">`, имеет атрибуты `ID` и `SRC`. Этим атрибутам соответствуют свойства `id` и `src` объекта данной картинки.

Ниже приведены примеры различных вариантов обращения к данным свойствам:

```
document.images(0).id        // "myimage"
document.images("myimage").id
document.all("myimage").id
document.all.myimage.id
var x=document.all.myimage   // объект
x.id                         // "myimage"
document.images(0).src       /* строка URL, например,
                                file:///C:/Мои%20документы/pict.jpg */
document.images("myimage").src
document.all("myimage").src
document.all.myimage.src
var x=document.all.myimage   // объект
x.src                        /* строка URL, например,
                                file:///C:/Мои%20документы/pict.jpg */
```

Объекты документа имеют много интересных и полезных свойств. Одни из них доступны только для чтения и могут использоваться в сценариях в качестве информации для выполнения каких-то действий. Другие свойства можно изменять, т. е. присваивать им иные значения (например, для изменения картинки достаточно изменить значение свойства `src`).

3.2. Обработка событий

Одним из главных, но далеко не единственным, назначений клиентских сценариев в HTML-документе является обработка событий. Каждое действие пользователя (нажатие клавиши, щелчок кнопкой мыши, изменение содержимого поля ввода данных и т. п.) формирует некоторое событие, т. е. сообщение о произошедшем. Операционная система анализирует это сообщение, чтобы узнать, откуда оно взялось и что с ним делать дальше. Если, например, пользователь нажал кнопку мыши в момент, когда ее указатель находился над окном браузера, то операционная система пошлет браузеру сообщение о том, какая кнопка мыши была нажата, какие при этом клавиши удерживаются, а также координаты указателя мыши. Если пользователь щелкнул где-то на панели инструментов браузера, то браузер отработает это сообщение сам. Если же в момент щелчка указатель находился на "территории" HTML-документа, то браузер пропустит сообщение о событии через свою объектную модель. В HTML-коде документа может находиться сценарий для обработки этого события. Инструкции этого сценария направляются к браузеру опять же через объектную модель.

3.2.1. Привязка обработчиков к элементам и событиям

Большинство тегов HTML имеют специальные атрибуты, определяющие события, на которые могут отреагировать соответствующие элементы. Список всех допустимых событий довольно обширен и рассчитан практически на все случаи жизни. События называются довольно просто, особенно если вы знаете английский язык. Например, щелчок левой кнопкой мыши — `onclick`; изменение в поле ввода данных — `onchange`; событие `onmouseover` происходит, когда указатель мыши помещается на элемент HTML-документа. Список событий приведен в *приложении 1*. Значением таких атрибутов-событий в тегах HTML является строка, содержащая сценарий, который играет роль обработчика события. Например, следующий HTML-код определяет заголовок второго уровня, который реагирует на щелчок кнопкой мыши тем, что выполняет некоторую функцию `myfunc()`:

```
<H2 onclick="myfunc()">Щелкни здесь</H2>
```

Определение функции `myfunc()` должно находиться либо в одном из контейнеров `<script>`, либо в файле, который должен быть вставлен в HTML-документ с помощью тега `<script>`.

Для одного и того же элемента можно определить несколько событий, на которые он будет реагировать. Иначе говоря, для одного и того же тега можно указать несколько атрибутов-событий. Имена этих атрибутов, как и других, можно писать в любом регистре. Порядок следования атрибутов не имеет значения.

Итак, значением атрибута-события, как уже отмечалось, является код сценария, заключенный в кавычки. Этот сценарий называют также *обработчиком события*. В приведенном выше примере обработчиком события `onclick` является функция `myfunc()`. Если обработчик события содержит несколько выражений, то они разделяются точкой с запятой. Например:

```
<H2 onclick="var x=5; x=x + myfunc()">Щелкни здесь</H2>
```

На рис. 3.4 показаны в общем виде варианты привязки обработчиков событий к элементам HTML-документа.

Обычно обработчики событий оформляются в виде функций, определения которых помещают в контейнерный тег `<SCRIPT>`. В качестве примера ниже приведены два варианта оформления обработчика щелчка на изображении.

Изображение в HTML-документе определяется, как известно, тегом `<IMG>`. Файл с изображением задается атрибутом `SRC`. Обработчик события `onclick` задается в примере как функция `clickimage()`, которая должна быть опреде-

лена где-то в контейнере `<SCRIPT>`. В результате щелчка на графическом изображении из файла picture.jpg выводится окно с сообщением (листинги 3.3 и 3.4).

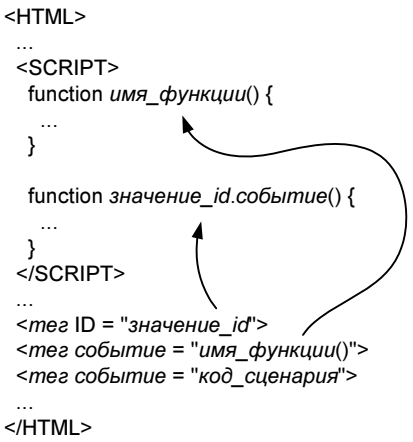


Рис. 3.4. Схема вариантов привязки обработчиков событий к элементам HTML-документа

Листинг 3.3. Вывод сообщения после щелчка кнопкой мыши на изображении *(вариант 1)*

```
<HTML>
  <SCRIPT>
    function clickimage(){
      alert("Привет!")
    }
  </SCRIPT>
  <IMG SRC="picture.jpg" onclick="clickimage()">
</HTML>
```

Листинг 3.4. Вывод сообщения после щелчка кнопкой мыши на изображении *(вариант 2)*

```
<HTML>
   <IMG SRC="picture.jpg" onclick="alert("Привет!")">
</HTML>
```

Если сценарий обработки события небольшой и используется в HTML-документе один раз, то целесообразно оформлять его непосредственно в виде значения атрибута-события. В других случаях предпочтителен первый вариант, т. е. оформление обработчика в виде функции.

Чтобы указать браузеру явным образом, что сценарий написан на языке JavaScript, можно в значении атрибута-события написать префикс `"javascript:"`. Например:

```
<IMG onclick="javascript: alert("Привет!")">
```

Если не указывать язык сценария, то браузер будет подразумевать JavaScript.

Теперь рассмотрим еще один способ оформления обработчиков событий. Прежде всего отметим, что почти для всех тегов HTML можно помимо прочих указать еще один атрибут — `ID` (идентификатор). Этот атрибут принимает любые строковые значения, которые играют роль индивидуальных имен элементов в их объектном представлении. Если в теге указан атрибут `ID`, то для задания обработчика события можно не использовать атрибуты-события. Вместо этого в контейнере `<SCRIPT>` достаточно написать определение функции обработчика события, имя которой образуется по следующему шаблону:

значение_ID.событие()

Пример — в листинге 3.5.

Листинг 3.5. Использование атрибута `ID`

```
<HTML>
  <H1 ID="myheader">Привет всем!</H1>
  <SCRIPT>
    function myheader.onclick(){
      alert("Привет!")
    }
  </SCRIPT>
</HTML>
```

Обратите внимание на следующие два обстоятельства, важные только для рассмотренного выше способа оформления обработчиков событий:

- элемент HTML-документа должен быть загружен раньше, чем функция-обработчик события;
- составное имя функции-обработчика события содержит название события в нижнем регистре.

Совет

Браузер, встречая в HTML-документе тег с определенным `ID`, создает в объектной модели этого документа объект с таким же именем. Для этого объекта имеется метод обработки события. Название метода совпадает с названием события, но синтаксис использования метода требует, чтобы его название было написано в нижнем регистре. В связи с этим я советую вам в любом случае писать название события в нижнем регистре.

Если требуется, чтобы сценарий обработал событие независимо от того, с каким элементом оно связано, то можно воспользоваться таким составным именем функции-обработчика:

```
function document.событие(){
   ...
}
```

Например:

```
function document.onclick(){
  ...
}
```

Приведенных выше способов обработки событий (привязки обработчиков к элементам) вполне достаточно для практических нужд разработки Web-сайтов и других приложений. Однако следует упомянуть еще об одном способе, который применим для Internet Explorer версии 4.0 и старше. Он основан на использовании атрибутов `FOR` и `EVENT` в теге `<SCRIPT>`.

Атрибуту `FOR` присваивается ссылка на объект, который должен реагировать на событие, а атрибуту `EVENT` — название события. В качестве ссылки на объект обычно используется значение идентификатора объекта, т. е. значение атрибута `ID` (листинг 3.6).

Листинг 3.6. Использование атрибутов `FOR` и `EVENT`

```
<HTML>
  <SCRIPT FOR="MYBUTTON" EVENT="onclick">
    alert("Был щелчок")
  </SCRIPT>
  <BUTTON ID="MYBUTTON">Нажми здесь</BUTTON>
</HTML>
```

Выражения в контейнере `<SCRIPT>` будут выполняться только при наступлении события, указанного в `EVENT`, которое связано с объектом, записанным

в FOR. В приведенном выше примере при щелчке на кнопке (<BUTTON...>) будет выведено диалоговое окно с сообщением.

Рассмотренный выше способ позволяет не использовать атрибуты-события и указания обработчиков событий в тегах элементов. Однако при этом код сценария оказывается привязанным только к одному элементу. Чтобы использовать этот же код для других элементов, придется повторить его в секциях сценария (контейнерах <SCRIPT>), отдельно для каждого элемента.

В рассмотренных выше способах обработчики событий привязывались к специальным атрибутам тегов HTML — ID, атрибутам-событиям, FOR и EVENT. Сценарий-обработчик выполнялся, если возникало событие, связанное с элементом, для которого был указан этот обработчик. Вместе с тем, имеется возможность вызывать обработчик в сценарии просто как метод объекта. Он будет выполняться так же, как и при возникновении соответствующего события. Рассмотрим пример (листинг 3.7).

Листинг 3.7. Вызов обработчика события как метода объекта

```
<HTML>
  <BUTTON ID="Mybutton" onclick="butclick()">Нажми здесь</BUTTON>
  <SCRIPT>
    document.all.Mybutton.onclick()   // вызов обработчика
    function butclick(){
      alert('Щелчок на кнопке Mybutton')
    }
  </SCRIPT>
</HTML>
```

В данном примере сразу же после загрузки документа в браузер вызывается обработчик события onclick, связанного с кнопкой. Этот обработчик — функция butclick(). Обратите внимание, что обработчик выполнится, несмотря на то, что самого щелчка не было.

Внимание

Выражение вызова метода (в данном случае это onclick()) должно записываться только строчными буквами.

Следует иметь в виду, что рассмотренный выше способ использования обработчика как метода существенно отличается от эмуляции действия для собы-

тия. Например, допустим, что имеется форма с несколькими полями для ввода данных, одно из которых имеет обработчик события `onfocus` (активизация поля). Чтобы поле активизировалось, необходимо щелкнуть на нем мышью или перейти к нему с помощью клавиатуры. При этом выполнится обработчик события `onfocus`. Однако даже если поле неактивно, обработчик события `onfocus` все равно можно вызвать, но при этом поле не активизируется:

```
document.имя_формы.имя_поля.onfocus()
```

Благодаря возможности использовать обработчик события в качестве метода объекта, допускается задавать обработчик в виде функции, имеющей название, совпадающее с названием события:

```
<HTML>
  <BODY onload="onload()"
    ...
  </BODY>
  <SCRIPT>
    function onload(){
      ...
    }
  </SCRIPT>
</HTML>
```

3.2.2. Свойства события

Сообщение о событии формируется в виде объекта, т. е. контейнера для хранения информации. Как только объект события создан, браузер присваивает значения его свойствам. Например, объект, соответствующий щелчку мышью, содержит координаты указателя мыши, а также сведения о том, какая кнопка была нажата. Кроме того, объект события в одном из своих свойств содержит ссылку на элемент, с которым связано данное событие (например, на кнопку, изображение, поле ввода и т. п.).

Объект события хранится в памяти столько времени, сколько необходимо сценарию для его обработки. Пока выполняется обработчик события, объект события вместе со своими свойствами доступен сценарию, находящемуся в памяти браузера. Как только обработчик события завершит работу, объект события становится пустым (возвращается в исходное состояние).

В любой момент времени существует не более одного объекта события. Все инициированные события заносятся операционной системой в буфер и выполняются последовательно в том порядке, в каком они туда попали.

В объектной модели имеется объект event, являющийся подобъектом объекта окна window. Он содержит информацию о том, какое событие произошло, какой элемент должен на него реагировать и ряд других характеристик. Объект event можно использовать в сценариях для документов с большим количеством интерактивных элементов, для выяснения причин неправильного отображения Web-страниц. Далее мы рассмотрим некоторые наиболее часто используемые свойства объекта event.

В качестве примера в листинге 3.8 приведен HTML-документ, не содержащий видимых элементов. Щелчок левой кнопкой мыши или нажатие клавиши выводит диалоговое окно со значениями некоторых свойств объекта event. Данный пример иллюстрирует использование объекта event для определения того, какая клавиша была нажата.

Листинг 3.8. HTML-документ, выводящий окно со значениями свойств объекта event

```
<HTML>
  <BODY ID="test">
  </BODY>
  <SCRIPT>
    function test.onclick(){
      var str=""
      str += "x=" + window.event.x + "\n"
      str += "y=" + window.event.y + "\n"
      str += "Вы нажали клавишу: "
      if (window.event.shiftKey){str += "Shift"}
      if (window.event.ctrlKey){str += "Ctrl"}
      if (window.event.altKey){str += "Alt"}
      alert(str) // вывод сообщения
    }

    function test.onkeypress(){
      alert("Код клавиши: " + window.event.keyCode)
    }
  </SCRIPT>
</HTML>
```

Так, свойства x и y содержат значения координат указателя мыши в момент щелчка, отсчитанные в данном случае относительно окна браузера. Свойство

`keyCode` возвращает код нажатой клавиши с символом в системе Unicode. Для латинских и цифровых символов кодировки Unicode и ASCII совпадают. При нажатии навигационных клавиш и клавиш дополнительной цифровой клавиатуры `keyCode` возвращает `null`. По существу `keyCode` — это код соответствующего символа, а не код клавиши. Для фиксации самой клавиши используются события `onkeydown` и `onkeyup`, а не `onkeypress`.

В Internet Explorer 4.0 и старше, среди прочих, часто оказываются полезными свойства `button` и `srcElement`.

Свойство `button` возвращает целочисленное значение, указывающее, какая кнопка или кнопки мыши были нажаты (табл. 3.1).

Таблица 3.1. *Значения свойства* `button`

Значение	Описание
0	Кнопки не нажаты
1	Нажата левая кнопка мыши
2	Нажата правая кнопка мыши
3	Одновременно нажаты левая и правая кнопки мыши
4	Нажата средняя кнопка мыши
5	Нажаты левая и средняя кнопки мыши
6	Нажаты правая и средняя кнопки мыши
7	Все три кнопки нажаты

Свойство `srcElement` возвращает ссылку на объект элемента HTML-документа, который инициировал событие. При получении такой ссылки можно узнать или изменить значения свойств этого объекта и применить к нему любой из его методов.

Приведенный в листинге 3.9 HTML-код формирует документ, содержащий две кнопки. Сценарий обрабатывает событие `onclick` — щелчок мышью. Он привязан к элементу, заданному тегом `<BODY>`. Щелкнуть можно на любой кнопке, а также на незанятом месте окна браузера. В любом случае событие `onclick` будет обработано сценарием (функцией `changetext()`), поскольку кнопки заданы внутри тега `<BODY>`. Сценарий заменит текст, который находится внутри тега элемента, инициировавшего событие. Какой именно элемент инициировал событие, определяется с помощью свойства `srcElement`, а замена текста осуществляется посредством свойства `innerText`. Об этом свойстве мы еще поговорим отдельно. Обратите внимание, что результат зависит от того, где находился указатель мыши в момент щелчка.

Листинг 3.9. Обработка события `onclick` *(вариант 1)*

```
<HTML>
  <BODY onclick="changetext()">
    <button>Первая кнопка</button>
    <button>Вторая кнопка</button>
  </BODY>
  <SCRIPT>
    function changetext(){
      x=window.event.srcElement       // ссылка на объект
      x.innerText="Уже щелкнули"      // замена текста
    }
  </SCRIPT>
</HTML>
```

Чтобы на щелчок реагировали только кнопки, приведенный выше код необходимо несколько модифицировать (листинг 3.10).

Листинг 3.10. Обработка события `onclick` *(вариант 2)*

```
<HTML>
  <BODY>
    <button onclick="changetext()">Первая кнопка</button>
    <button onclick="changetext()">Вторая кнопка</button>
  </BODY>
  <SCRIPT>
    function changetext(){
      x=window.event.srcElement       // ссылка на объект
      x.innerText="Уже щелкнули"      // замена текста
    }
  </SCRIPT>
</HTML>
```

Объект `event` имеет несколько свойств, содержащих координаты в пикселах указателя мыши в момент возникновения события, связанного с мышью. Многообразие типов этих координат обусловлено различиями в начале их отсчета. Рассмотрим эти свойства по порядку.

- `screenX`, `screenY` — координаты указателя мыши относительно левого верхнего угла экрана. Например, если экран монитора находится в режиме

800×600, то координаты правого нижнего угла экрана будут равны 800 и 600 пикселам соответственно.

- `clientX`, `clientY` — координаты указателя мыши относительно области клиента (рабочей области браузера), в которой находится HTML-документ, без учета рамок, полос прокрутки, меню и т. п.
- `offsetX`, `offsetY` — координаты указателя мыши относительно верхнего левого угла элемента, инициировавшего событие.
- `x`, `y` — координаты указателя мыши относительно верхнего левого угла первого абсолютно или относительно позиционированного контейнера, в котором находится элемент, инициировавший событие. Позиционированный контейнер — это элемент, заданный каким-нибудь контейнерным тегом (например, `<BODY>`, `<DIV>`, `<H1>`) с атрибутом `STYLE`, для которого указано свойство `position`. Если такого контейнера нет, то свойства `x` и `y` возвращают координаты относительно главного документа, такие же, как и `clientX`, `clientY`.

Чтобы лучше разобраться в этих свойствах, создайте тестовый HTML-документ, как показано в листинге 3.11, и поэкспериментируйте с ним, щелкая мышью в различных точках. Этот документ содержит два контейнера, созданных с помощью позиционированных различным образом тегов `<DIV>`. Первый контейнер позиционирован абсолютно, а второй, вложенный в него, — относительно. Сценарий определяет значения координат различных типов и отображает их в диалоговом окне.

Листинг 3.11. Использование свойств объекта `event`

```
<HTML>
  <HEAD><TITLE>Координаты щелчка</TITLE></HEAD>
  <BODY ID="Mybody">
    <DIV ID="DIV1" STYLE="position:absolute; left:50; top:50;
                          width:300;height:100; background-color:blue">
    <DIV ID="DIV2" STYLE="position:relative; left:50; top:25;
                          width:200;height:50; background-color:yellow">
    </DIV>
    </DIV>
  <BODY>
  <SCRIPT>
    function document.onclick(){
      var e=window.event        // ссылка на объект event
```

```
      var str="ID=" + e.srcElement.id +
              "\n screenX=" + e.screenX + ", screenY=" + e.screenY +
              "\n clientX=" + e.clientX + ", clientY=" + e.clientY +
              "\n offsetX=" + e.offsetX + ", offsetY=" + e.offsetY +
              "\n offsetX=" + e.offsetX + ", offsetY=" + e.offsetY +
              "\n x=" + e. x + ", y=" + e.y
      alert(str)
    }
  </SCRIPT>
</HTML>
```

Нам требовалось, чтобы на щелчок реагировал любой объект документа, поэтому мы не стали привязывать обработчик события индивидуально ко всем элементам, а вместо этого назвали функцию-обработчик составным именем `document.onclick()`. Обратите внимание на использование свойства `srcElement` для получения значения `id` элемента, инициировавшего (или, как еще говорят, получившего) событие.

У объекта `event` имеется изменяемое свойство `returnValue` (возвращаемое значение). С его помощью можно отменять действия, принятые по умолчанию. Для этого достаточно присвоить ему значение `false`. Например, щелчок на ссылке по умолчанию означает переход по указанному адресу. Однако вы можете отменить это действие или запросить у пользователя подтверждение перехода, как показано в листинге 3.12.

Листинг 3.12. Генерация подтверждения для перехода по ссылке

```
<HTML>
  <A ID="myref" HREF="www.chat.ru">Переход на chat.ru</A>
  <SCRIPT>
    function myref.onclick(){
      ret=confirm("Вы действительно хотите перейти на chat.ru?")
      if (!ret)
        window.event.returnValue=false
    }
  </SCRIPT>
</HTML>
```

Здесь метод `confirm()` выводит диалоговое окно. Если пользователь не подтвердил переход, то `confirm` возвращает `false`, и свойству `returnValue` тоже присваивается `false`. В результате переход по ссылке не произойдет.

Нередко требуется разместить в документе элемент, который внешне выглядит как обычная ссылка (при наведении указателя мыши его вид изменяется), но щелчок должен приводить не к переходу на другой документ, а просто к выполнению некоторого сценария. Этого можно добиться несколькими способами. Один из них основан на использовании свойства `returnValue` (листинг 3.13).

Листинг 3.13. Использование свойства `returnValue`

```
<HTML>
  <A ID="myref" HREF="">Щелкни здесь</A>
  <SCRIPT>
    function myref.onclick(){
      alert("Был щелчок на ссылке")
      window.event.returnValue=false
    }
  </SCRIPT>
</HTML>
```

Если бы мы не использовали оператор `window.event.returnValue=false`, то из-за не указанного в атрибуте `HREF` адреса перехода по ссылке после выполнения обработчика события открылось бы новое окно браузера с содержимым текущей папки.

3.2.3. Прохождение событий

Как уже отмечалось ранее, для элементов документа можно указать события, на которые они должны реагировать, а также обработчики этих событий. Например, если требуется, чтобы элемент реагировал на щелчок кнопкой мыши выполнением некоторой функции `myfunc()`, то в тег этого элемента достаточно вставить запись: `onclick="myfunc()"`. Однако нам известно, что большинство тегов в HTML являются контейнерными и, следовательно, могут содержать другие теги. При этом может оказаться, что одно и то же событие будет обозначено в различных, но вложенных друг в друга тегах. Что произойдет при наступлении этого события? Как оно распространяется по объектам и как перехватывается элементами документа?

Рассмотрим в качестве примера HTML-документ, содержащий единственную кнопку (листинг 3.14).

Листинг 3.14. HTML-документ с кнопкой *(вариант 1)*

```
<HTML>
  <BODY onclick="alert('Щелчок на body')">
    <BUTTON onclick="alert('Щелчок на button')">Нажми здесь</BUTTON>
  </BODY>
</HTML>
```

Особенность этого документа состоит в том, что одно и то же событие (с различными обработчиками) привязано к различным тегам, один из которых содержит в себе другой (`<BUTTON>` содержится в `<BODY>`). Щелчок на кнопке приведет к выполнению сначала обработчика щелчка для кнопки, а затем обработчика для `<BODY>`. Если щелкнуть где-нибудь в рабочей области окна браузера вне кнопки, то сработает только обработчик для `<BODY>`. В листингах 3.15 и 3.16 мы покажем, как можно избавиться от этого нежелательного эффекта.

Модели прохождения событий в Internet Explorer и Netscape Navigator отличаются друг от друга. Более того, имеются различия между этими моделями для различных версий одного и того же браузера, даже начиная с IE4 и NN4.

Модель прохождения событий в IE4 называется *"всплыванием событий"*. События, подобно воздушным пузырькам в воде, как бы всплывают от целевого объекта самого нижнего уровня вверх по иерархии объектов. В рассмотренном выше примере событие `onclick` "всплывает" от объекта кнопки к объекту тела документа. В NN4 модель прохождения событий называют *"захватом событий"*. Согласно этой модели событие распространяется, наоборот, от самого верхнего в иерархии объекта `window` к целевому объекту. Чтобы события обрабатывались на уровне текущих объектов, требуется включить для них режим захвата. Это делается с помощью специального метода `cuptureEvent()`. В IE4 для управления всплыванием событий используют свойство прерывания всплывания `cancelBubble`.

В современных браузерах IE5.5+ и NN6 реализована модель, объединяющая и всплывание, и захват событий. Всплывание организовано как в IE4+, а захват — как в NN4. По умолчанию событие всплывает, однако вы можете включить его захват. Тогда событие сначала достигает целевого объекта, а затем начинает всплывать в обратном направлении.

В IE4+ управлять прохождением событий можно с помощью свойства `canselBubble` объекта `event`. Когда в документе (на Web-странице) происходит какое-либо событие, объект `event` первым получает информацию о нем и решает, какому элементу его передать. Например, когда объект `event` полу-

чает событие `onclick` (щелчок кнопкой мыши), то выясняет, какой элемент находился в тот момент под указателем мыши. Если там было два элемента (один над другим), то выбирается элемент с наименьшим z-индексом, т. е. нижний. Напомню, что `z-index` является одним из параметров позиционирования элементов с помощью таблиц стилей. Установив элемент, связанный с событием, объект `event` ищет обработчик этого события и выполняет его. Например, если элемент имеет уникальное имя (ID) `Myelement`, то ищется функция `Myelement.onclick()`. Далее, объект `event` выясняет, какой объект является контейнером для данного элемента. Если таковой имеется, то событие переходит внутрь этого контейнера. Если, например, контейнером является объект, созданный тегом `<DIV>` с идентификатором `Mydiv`, то будет предпринята попытка выполнить функцию `Mydiv.onclick()`. Этот процесс продолжается, пока имеются доступные контейнеры. Например, последним доступным контейнером может оказаться документ (объект `document`), и тогда объект `event` ищет функцию `document.onclick()`.

Чтобы определить главного виновника события (объект, инициировавший событие), используется свойство `srcElement`. О нем уже говорилось ранее. Например, может потребоваться одинаковая реакция на одно и то же событие всех элементов, за исключением некоторых. В таких случаях для элементов, на которых процесс всплывания события должен закончиться, создаются специальные функции-обработчики. В этих функциях, помимо прочего, должно быть выражение присваивания свойству `cancelBubble` значения `true`, чтобы прервать дальнейшее прохождение события. Например, если мы хотим разорвать цепную передачу события на элементе с именем `Myelement`, необходимо создать для него функцию-обработчик события следующего вида:

```
function Myelement.onclick(){
  // код обработки события onclick
  window.event.cancelBubble=true
}
```

Делать такое в функции-обработчике для объекта `document` бессмысленно, поскольку выше событие все равно всплыть не может, а отменить реакцию на это событие элемента-инициатора (первого звена цепочки) невозможно.

В листинге 3.14 уже рассматривался пример HTML-документа с кнопкой, вложенной в контейнер `<BODY>`. Особенность этого примера в том, что если пользователь щелкнет на кнопке, сработает обработчик не только кнопки, но и тела документа (элемента, заданного тегом `<BODY>`). Если же

требуется предотвратить это, то можно переписать код, как показано в листинге 3.15.

Листинг 3.15. HTML-документ с кнопкой *(вариант 2)*

```
<HTML>
  <BODY onclick="alert('Щелчок на body')">
    <BUTTON onclick="alert('Щелчок на button');
              window.event.cancelBubble=true ">Нажми здесь</BUTTON>
  </BODY>
</HTML>
```

Возможен также и такой вариант кода (листинг 3.16).

Листинг 3.16. HTML-документ с кнопкой *(вариант 3)*

```
<HTML>
  <BODY onclick="alert('Щелчок на body')">
    <BUTTON ID="Mybut">Нажми здесь</BUTTON>
  </BODY>
  <SCRIPT>
    function Mybut.onclick(){
      alert('Щелчок на button')
      window.event.cancelBubble=true /* прекратить всплывание события */
    }
  </SCRIPT>
</HTML>
```

3.3. Динамическое изменение элементов документа

HTML-документ, загруженный в браузер, можно изменять с помощью сценариев. Поскольку речь идет не об исходном файле документа на диске, а о его образе в браузере в оперативной памяти компьютера, то говорят о динамическом изменении документа. Динамическое изменение — изменение, не связанное с загрузкой в браузер исходного файла. С другой стороны, динамические изменения документа не будут воспроизведены при следующем запуске вашего документа: предыдущая динамика никак не сохраняется в статике (оперативное, т. е. временное состояние, не сохраняется в долговременной

памяти, т. е. на диске или на других долговременных носителях). Рассмотрим три основных способа динамических изменений:

- с помощью метода `write()`;
- путем изменения значений свойств, соответствующих атрибутам HTML-тегов и параметрам каскадных таблиц стилей;
- путем изменения значений свойств `innerText`, `outerText`, `innerHTML`, `outerHTML`, которые имеют почти все объекты, заданные с помощью тегов.

Примечание

В объектной модели документа W3C имеются средства создания и изменения элементов документа, основанные на концепции *узлов* (nodes). Согласно этой концепции каждый контейнер является узлом, в котором могут находиться другие контейнерные или неконтейнерные элементы. С помощью специальных объектов и их методов можно создавать новые узлы, перемещать их в иерархии узлов документа, а также изменять содержимое узлов. В этой книге данный подход к динамическому изменению документа не рассматривается. Для практических целей в большинстве случаев оказывается достаточным применение способов, перечисленных выше.

3.3.1. Использование метода *write()*

Метод `write()` объекта `document` принимает в качестве параметра строку, содержащую HTML-код и/или просто текст. Выполнение в сценарии выражения `document.write(`*строка*`)` приводит к дописыванию в текущий HTML-документ содержимого параметра *строка* и немедленной его интерпретации браузером. В результате документ и его объектная модель обновляются. При этом файл с исходным HTML-кодом остается без изменений. Если требуется полностью заменить текущий документ, то сначала применяют метод очистки документа `document.clear()`, а затем `document.write(`*строка*`)`. Однако при такой кардинальной трансформации текущего документа следует быть осторожным. Наиболее безопасный прием — сначала сгенерировать содержимое нового HTML-документа с помощью сценария в текущем документе, а затем отправить HTML-код в новое окно или в другой фрейм многофреймового документа.

Если быть более точным, то следует отметить, что метод `write()` может принимать произвольное количество строковых параметров:

`document.write(`*строка1* `[,` *строка2* `...[,` *строкаN*`]])`

Здесь квадратные скобки указывают лишь на необязательность заключенных в них параметров. Если указывается несколько параметров, то они разделя-

ются запятыми. Заметим также, что весь HTML-код документа можно записать как одну строку.

Кроме метода `write()` можно использовать для тех же целей и метод `writeln()`, имеющий такой же синтаксис. Его особенность в том, что он добавляет в конце каждой строки документа невидимый символ перехода на другую строку.

Методы `write()` и `writeln()` работают в браузерах IE3+ и NN2+.

3.3.2. Изменение значений атрибутов элементов

Элементы HTML-документа задаются тегами, большинство из которых имеют атрибуты (параметры). В объектной модели документа тегам элементов соответствуют объекты, а атрибутам — свойства этих объектов. В большинстве случаев названия свойств объектов совпадают с названиями атрибутов, но в отличие от последних записываются в *нижнем регистре*. Это же относится и к параметрам таблиц стилей. Составным именам параметров, записываемых в таблицах стилей через дефис, соответствуют свойства, названия которых образуются так: удаляются дефисы, а второе и последующие слова записываются с прописной буквы. Например, параметру `z-index` в таблице стилей соответствует свойство `zIndex`.

Рассмотрим в качестве примера HTML-код вставки графического изображения:

```
<IMG ID="myimg" SRC="picture.jpg">
```

Этому HTML-элементу соответствует объект `document.all.myimg`, а атрибуту `SRC` — свойство `document.all.myimg.src`, значением которого является имя (URL-адрес) файла с изображением. С помощью сценария можно присвоить данному свойству новое значение, и в HTML-документе произойдет замена графического элемента.

В листинге 3.17 приведен пример документа, содержащего графическое изображение. Щелчок на изображении приводит к его замене.

Листинг 3.17. Щелчок на изображении приводит к его замене

```
<HTML>
<IMG ID="myimg" SRC="picture1.jpg"
   onclick='document.all.myimg.src="picture2.jpg"'>
</HTML>
```

В приведенном примере смена изображения происходит лишь при первом щелчке на нем. Последующие щелчки не приведут к видимым изменениям, поскольку второе изображение будет заменяться самим собой. Чтобы повторный щелчок приводил к отображению предыдущего рисунка, сценарий необходимо слегка усложнить: следует создать переменную-триггер (так называемый флаг), принимающий одно из двух возможных значений. По текущему значению флага сценарий может определить, какое именно из двух изображений следует отобразить. После смены изображения необходимо изменить и значение флага (листинг 3.18).

Листинг 3.18. Поочередная смена двух изображений

```
<HTML>
  <IMG ID="myimg" SRC="picture1.jpg"
       onclick="imgchange()">
  <SCRIPT>
    var flag=false              // флаг (триггер)
    function imgchange(){       // обработчик щелчка на изображении
      if (flag) document.all.myimg.src="picture1.jpg"
      else document.all.myimg.src="picture1.jpg"
      flag=!flag                /* изменяем значение флага на
                                   противоположное */
    }
  </SCRIPT>
</HTML>
```

Рассмотрим теперь случай нескольких изображений. Например, на Web-странице расположена галерея миниатюр (thumbnails — уменьшенных копий крупномасштабных изображений). Мы хотим, чтобы при щелчке кнопкой мыши на миниатюре она увеличивалась, а затем при щелчке на увеличенном изображении оно уменьшалось. Для решения этой задачи потребуется столько флагов, сколько имеется изображений. Флаги определим как массив, каждый элемент которого будет соответствовать отдельному изображению. Далее мы не станем создавать отдельную функцию-обработчик события `onclick` для каждого графического объекта. Вместо этого запрограммируем одну функцию-обработчик, которая, кроме всего прочего, будет сама определять, на каком именно изображении произошел щелчок. Идентификаторы `ID` тегов `<IMG>` зададим некоторым регулярным образом (например, `"i0"`, `"i1"`, `"i2"`, ...). Так часто поступают при использовании массивов. Создадим еще два массива, содержащих имена графических файлов: один для исходных изображений, а

другой — для замещающих. Наконец, HTML-документ с изображениями сгенерируем с помощью сценария. Для этого сначала сформируем строку, содержащую теги `<IMG...>`, а затем запишем ее в документ. В листинге 3.19 приводится код, реализующий эту программу.

Листинг 3.19. Увеличение миниатюр при щелчке кнопкой мыши

```
<HTML>
  <SCRIPT>
    var apict1=new Array("pict1.gif", ...) /* массив имен исходных
                                              файлов */
    var apict2=new Array("pict2.jpg", ...) /* массив имен замещающих
                                              файлов */
    var aflag=new Array(apict1.length)     // массив флагов
    /* Формирование строки тегов, описывающих изображения */
    var xstr=""
    for(i=0; i < apict1.length; i++){
      xstr+= '<IMG ID="i' + i + '" SRC="' + apict1[i]+'"
                onclick="imgchange()">'
    }
    document.write(xstr)            // запись в документ

    function imgchange(){           /* обработчик щелчка на
                                       изображении */
      var xid=event.srcElement.id   / * id изображения,
                                        на котором был щелчок */
      var n=parseInt(xid.substr(1)) // выделяем номер элемента

      if (aflag[n])
        document.all[xid].src=apict1[n]
      else
        document.all[xid].src=apict2[n]
      aflag[n]=!aflag[n]      /* изменяем значение флага на
                                 противоположное */
    }
  </SCRIPT>
</HTML>
```

Обратите внимание, как происходит обращение к свойству `src`: `document.all[xid].src`, а не `document.all.xid.src` и, тем более, не `document.all["xid"].src`. Это объясняется тем, что `xid` является строковой переменной, содержащей значение идентификатора `ID`, а не собственно значением идентификатора.

Многие параметры элементов задаются с помощью таблиц стилей, например, посредством атрибута `STYLE`. Так, для позиционирования изображения можно использовать следующий HTML-код:

```
<IMG ID="myimg" SRC="picture.jpg"
     STYLE="position:absolute; top:20; left: 50; z-index:3">
```

Чтобы изменить в сценарии параметры стиля элемента, следует присвоить новые значения соответствующим свойствам объекта `style`:

```
document.all.myimg.style.top=30
document.all.myimg.style.top=100
document.all.myimg.style.zIndex=-2
```

Обращение со стилевыми параметрами более подробно рассмотрено в *разд. 3.4.*

3.3.3. Изменение элементов

Под изменением элемента будем понимать изменение содержимого HTML-тега или даже самого тега. Наиболее удобный способ динамического изменения элементов HTML-документа в IE4+ основан на применении свойств `innerText`, `outerText`, `innerHTML` и `outerHTML`. В NN6+ можно использовать только свойство `innerHTML`.

Теги элементов документа могут содержать текст и/или другие теги. С помощью перечисленных выше свойств можно получить доступ к содержимому элемента. Изменяя значения этих свойств, легко изменить сам элемент, частично или полностью. Например, можно заменить только надпись на кнопке, а можно превратить кнопку в картинку или Flash-анимацию.

Значением свойства `innerText` является все текстовое содержимое, помещенное между открывающим и закрывающим тегами элемента. Если в эту строку входят HTML-теги, то они игнорируются. Обратите внимание, что в значение свойства `innerText` данные открывающего и закрывающего тегов соответствующего элемента не входят. Таким образом, `innerText` следует понимать как весь внутренний текст, содержащийся в контейнере.

Свойство `outerText` аналогично свойству `innerText`, но отличается от него тем, что включает в себя и данные, содержащиеся в открывающем и закры-

вающем тегах элемента. Таким образом, `outerText` следует понимать как весь текст, содержащийся в контейнере, включая и его внешние теги.

Рассмотрим в качестве примера следующий фрагмент HTML-кода, выводящий ссылку, картинку и форматированный текст:

```
<DIV ID="my">
  <A HREF='raznoe.htm'>
    <IMG SRC='picture.jpg'>Ссылка на раздел <B>Разное</B>
  </A>
</DIV>
```

В этом случае свойства `innerText` и `outerText` для элемента, заданного контейнерным тегом `<DIV>`, совпадают:

```
document.all.my.innerText  // значение равно: "Ссылка на раздел Разное"
```

При присвоении свойствам `innerText` и `outerText` новых значений следует иметь в виду, что если значение содержит HTML-теги, то они не интерпретируются, а выводятся на экран просто как текст. Так, для приведенного выше фрагмента HTML-кода выполнение в сценарии выражения

```
document.all.my.innerText="<BUTTON>Щелкни здесь</BUTTON>"
```

не создаст в документе кнопку, а лишь выведет строку с текстом, указанным справа от оператора присваивания.

Рассмотренные выше свойства `innerText` и `outerText` не столь эффективны как замечательные свойства `innerHTML` и `outerHTML`. Без них мне было бы просто скучно заниматься программированием на HTML и JavaScript.

Свойство `innerHTML` для любого элемента имеет в качестве значения строку, содержащую HTML-код, заключенный между открывающим и закрывающим тегами элемента. Иначе говоря, `innerHTML` содержит внутренний HTML-код контейнера элемента. Присвоение этому свойству нового значения, содержащего HTML-код, приводит к интерпретации этого кода. Разумеется, новое значение может и не содержать тегов.

Свойство `outerHTML` аналогично свойству `innerHTML`, но отличается тем, что содержит весь HTML-код, включая внешние открывающий и закрывающий теги элемента.

Для приведенного выше фрагмента HTML-кода имеет место следующее:

```
document.all.my.innerHTML    /* значение равно:
        "<A HREF='raznoe.htm'><IMG SRC='picture.jpg'> Ссылка на раздел
         <B>Разное</B></A>" */
```

```
document.all.my.outerHTML    /* значение равно: "<DIV ID='my'>
         <A HREF='raznoe.htm'><IMG SRC='picture.jpg'> Ссылка на раздел
         <B>Разное</B></A></DIV>" */
```

Если в сценарии выполнить, например, выражение

```
document.all.my.innerHTML="<BUTTON>Щелкни здесь</BUTTON>"
```

то ссылка, картинка и текст будут заменены кнопкой **Щелкни здесь**. При этом контейнерный тег `<DIV ID="my">` сохранится. Если же вместо свойства `innerHTML` использовать свойство `outerHTML`, то кнопка также появится, но уже без контейнера `<DIV ID="my">`.

Заметим, что свойства `innerHTML` и `outerHTML` могут применяться и к элементам, которые задаются не контейнерными тегами, например, тегом `<IMG>`. В этом случае значения `innerHTML` и `outerHTML` всегда совпадают.

Далее приводится код функции `getproperties(xid)`, которая может использоваться в сценарии для получения значений свойств `innerHTML`, `outerHTML`, `innerText` и `outerText` элементов HTML-документа. Эта функция принимает в качестве параметра значение идентификатора `ID` элемента (или значение его атрибута `NAME`) и выводит диалоговое окно со значениями перечисленных выше свойств. Вы можете применять ее на этапе изучения.

```
function getproperties(xid){
  var x=eval("document.all." + xid)
  alert(" innerHTML: " + x.innerHTML + "\n outerHTML: " +
          x.outerHTML + "\n innerText: " + x.innerText +
          "\n outerText: " + x.outerText)
}
```

Пример использования функции `getproperties(xid)` приведен в листинге 3.20.

Листинг 3.20. Использование функции `getproperties(xid)`

```
<HTML>
  <DIV ID="my">
    <A HREF='raznoe.htm'>
    <IMG SRC='picture.jpg'>Ссылка на раздел <B>Разное</B></A>
  </DIV>
  <H2 ID="privet">Привет <I>читателям</I></H3>>
  <SCRIPT>
    getproperties("my")       // отображение свойств элемента my
    getproperties("privet")   // отображение свойств элемента privet
```

```
    function getproperties("xid"){
      // код функции
    }
  </SCRIPT>
</HTML>
```

3.4. Работа с каскадными таблицами стилей

Доступ к каскадным таблицам стилей в сценариях осуществляется через специальные объекты:

- объект элемента, соответствующего тегу `<STYLE>` или `<LINK>` (объект элемента стиля);
- объект `styleSheet` (элемент коллекции `styleSheets`);
- объект `cssRule` (элемент коллекций `rules` и `cssRules`);
- объект `style`.

Несмотря на то, что тег `<STYLE>` содержит строки правил таблицы стилей, объект элемента стиля не предназначен для получения доступа к отдельным правилам. Для этого служит объект `styleSheet`, являющийся элементом коллекции документа `styleSheets`. С помощью этого объекта можно подключить или отключить всю таблицу, получить доступ к отдельным правилам, добавить и удалить правила. Доступ к отдельным правилам производится посредством подобъектов `rules` или `cssRules`, являющихся коллекциями правил. Указанные выше коллекции представляют собой массивы и, следовательно, к их элементам можно обращаться с помощью индексов.

Для получения доступа к конкретному правилу можно воспользоваться одним из следующих выражений:

```
document.styleSheets[i].rules[j]      // для IE4+
document.styleSheets[i].cssRules[j]   // для IE5+, NN6+
```

Напомним, что индексация элементов массива в JavaScript начинается с 0.

Каждому тегу `<STYLE>` можно задать атрибут `ID`, чтобы вместо `styleSheets` использовать обращение по значению идентификатора. Объект элемента стиля обладает логическим свойством `disabled`. Значение `true` этого свойства деактивизирует (отключает) таблицу стилей, а значение `false` — активизирует (подключает).

Пример — в листинге 3.21.

Листинг 3.21. Подключение таблицы стилей в сценарии

```
<HTML>
  <STYLE ID="mystyle">
    H1 {font-size:96pt; color:red}
    P  {color:blue}
  </STYLE>
  <H1>Привет!</H1>
  <P>Пишет тебе Вася Пупкин</P>
  <BUTTON onclick="document.all.mystyle.disabled=true">
  Отключить стиль
  </BUTTON>
  <BUTTON onclick="document.all.mystyle.disabled=false">
  Подключить стиль
  </BUTTON>
</HTML>
```

Объект `style` (не путайте с объектом элемента стиля) является основным. Он определен для любого HTML-элемента и обычно применяется для изменения стилевых параметров элементов документа. Для этого можно использовать следующий синтаксис:

`document.all.`*`id_элемента`*`.style.`*`параметр=значение`*

Например:

```
document.all.myelement.style.color="blue"
```

Кроме того, что объект `style` является свойством объектов, представляющих HTML-элементы документа, он еще является и свойством объектов `rules` и `cssRules`. Поэтому к конкретному стилевому параметру элемента документа, находящемуся под действием определенного правила таблицы стилей, можно обратиться, например, и так:

```
document.styleSheets[i].rules[j].style.color="blue"     // для IE4+
document.styleSheets[i].cssRules[j].style.color="blue"  // для IE5+, NN6+
```

Пример — в листинге 3.22.

Листинг 3.22. Обращение к стилевому параметру в сценарии

```
<HTML>
  <STYLE>
```

```
    H1 {font-size:96pt; color:red}
    P  {color:blue}
  </STYLE>
  <H1>Привет!</H1>
  <P>Пишет тебе Вася Пупкин</P>
  <SCRIPT>
    x=document.styleSheets[0].rules[0].style.color // red
    y=document.styleSheets[0].rules[1].style.color // blue
    alert("Цвет в 1-м правиле: " + x+"\nЦвет во 2-м правиле: "+ y)
  </SCRIPT>
</HTML>
```

Таблицы стилей, вставленные в HTML-документ посредством тега `<STYLE>`, могут быть импортированы из внешних css-файлов с помощью оператора `@import`. В этих случаях для доступа к конкретному правилу импортированной таблицы используют такое выражение:

```
document.styleSheets[i].imports[k].rules[j]
```

Как уже отмечалось выше, чтобы задать значение стилевого параметра, необходимо использовать объект `style`. Это можно сделать, даже если таблица стилей для элемента документа не определена.

Пример — в листинге 3.23.

Листинг 3.23. Использование стилевого параметра без таблицы стилей

```
<HTML>
  <H1 ID=myheader>Добро пожаловать!</H1>
  <SCRIPT>
    document.all.myheader.style.color="red"
  </SCRIPT>
</HTML>
```

Обратите внимание, что в этом HTML-коде таблица стилей нигде не определена, хотя сценарий устанавливает красный цвет для стилевого параметра `color` заголовка 1-го уровня.

Если таблица стилей для элемента документа не определена, то в объекте `style` для этого элемента все свойства имеют пустые значения, а не значения стилевых параметров, принятые по умолчанию. Поэтому в листинге 3.24 размер шрифта для заголовка будет не увеличен на 10, а установлен равным 10.

Листинг 3.24. Установка размера шрифта

```
<HTML>
  <H1 ID=myheader>Добро пожаловать!</H1>
  <SCRIPT>
    x=document.all.myheader.style.fontSize       // пустая строка
    document.all.myheader.style.fontSize=x + 10
    alert(document.all.myheader.style.fontSize)  // 10px
  </SCRIPT>
</HTML>
```

Значения стилевых параметров (свойств объекта `style`) являются строковыми. Поэтому при их задании следует использовать кавычки. Например, выражение

```
document.all.myheader.style.color=red
```

приведет к ошибке. Для параметров, связанных с размерами и координатами (например, `width`, `height`, `top`, `left`, `fontSize`), значения можно задавать в виде чисел без кавычек. Однако в действительности этим параметрам будут присвоены соответствующие строковые значения с указанием `px` (пикселы) как единицы измерения. Это обстоятельство необходимо иметь в виду особенно при использовании арифметических выражений для вычисления значений.

В примере, представленном в листинге 3.25, используется функция `parseInt()` для преобразования строки `"10px"` в целое число 10. Это необходимо для корректности арифметической операции.

Листинг 3.25. Использование арифметических выражений для вычисления значений

```
<HTML>
  <H1 ID=myheader>Добро пожаловать!</H1>
  <SCRIPT>
    document.all.myheader.style.fontSize=10               // 10px
    document.all.myheader.style.fontSize =
      parseInt(document.all.myheader.style.fontSize) + 10 // 20px
  </SCRIPT>
</HTML>
```

Если бы нам потребовалось задать размер шрифта не в пикселах, а в пунктах (`pt`), то следовало бы воспользоваться кодом из листинга 3.26.

Листинг 3.26. Установка размера шрифта в пунктах

```
<HTML>
  <H1 ID=myheader>Добро пожаловать!</H1>
  <SCRIPT>
    document.all.myheader.style.fontSize="10pt" // 10pt
    document.all.myheader.style.fontSize =
      String(parseInt(document.all.myheader.style.fontSize) + 10)
        + "pt"                                  // 20pt
  </SCRIPT>
</HTML>
```

Если таблица стилей вставлена в документ с помощью тега `<STYLE>` или `<LINK>` и задает параметры для некоторого HTML-элемента, то даже в этом случае, используя объект `style`, не удастся прочитать значения стилевых параметров (листинг 3.27).

Листинг 3.27. Попытка чтения стилевых параметров *(вариант 1)*

```
<HTML>
  <STYLE>
    H1 {color:red}
  </STYLE>
  <H1 ID=myheader>Добро пожаловать!</H1>
  <SCRIPT>
    alert(document.all.myheader.style.color) // выводится пустая строка
  </SCRIPT>
</HTML>
```

Если же таблица стилей задана как значение атрибута `STYLE`, то с помощью объекта `style` можно прочитать значения определенных стилевых параметров (листинг 3.28).

Листинг 3.28. Попытка чтения стилевых параметров *(вариант 2)*

```
<HTML>
  <H1 ID=myheader STYLE="color:red">Добро пожаловать!</H1>
  <SCRIPT>
    alert(document.all.myheader.style.color) // выводится строка "red"
  </SCRIPT>
</HTML>
```

Чтобы прочитать значения стилевых параметров, заданных в таблицах с помощью тега `<STYLE>` или `<LINK>`, не используя `styleSheets` и `rules`, в браузере IE5+ можно применять объект `currentStyle` (текущий стиль). Этот объект доступен только для чтения, т. е. с его помощью нельзя изменить значения стилевых параметров. Он возвращает таблицу стилей, примененную к HTML-документу (листинг 3.29).

Листинг 3.29. Использование объекта `currentStyle`

```
<HTML>
  <STYLE>
    H1 {color:red}
  </STYLE>
  <H1 ID=myheader>Добро пожаловать!</H1>
  <SCRIPT>
    alert(document.all.myheader.currentStyle.color) // выводится строка
                                                    // "red"
  </SCRIPT>
</HTML>
```

Для браузера IE5.5+ имеется еще один объект — `runtimeStyle` (стиль времени исполнения), который позволяет отменить любое свойство стиля, заданное с помощью атрибута `STYLE`. Новое значение свойства стиля отразится на внешнем виде элемента, но при этом не изменится значение этого же параметра в таблице стилей (листинг 3.30).

Листинг 3.30. Использование объекта `runtimeStyle`

```
<HTML>
  <H1 ID=myheader STYLE="color:red">Добро пожаловать!</H1>
  <SCRIPT>
    alert(document.all.myheader.currentStyle.color) // выводится строка
                                                    // "red"
    document.all.myheader.runtimeStyle.color='blue'
    alert(document.all.myheader.runtimeStyle.color) // выводится строка
                                                    // "blue"
    alert(document.all.myheader.currentStyle.color) // выводится строка
                                                    // "blue"
    alert(document.all.myheader.style.color)        // выводится строка
                                                    // "red"
```

```
  </SCRIPT>
</HTML>
```

Внимание

Свойства, соответствующие стилевым параметрам, в сценариях записываются не так, как в правилах каскадных таблиц стилей. Названия свойств, как и имена переменных, в сценариях не должны содержать дефисов, широко используемых в таблицах стилей. Поэтому названия стилевых параметров в сценариях образуются из имен соответствующих параметров таблиц стилей путем перезаписи последних в стиле interCap. Это означает, что в составных именах опускаются дефисы, а второе и последующие слова записываются с прописной буквы.

В табл. 3.2 показано соответствие между способами представления некоторых параметров в правилах таблиц стилей и сценариях (как свойств стиля).

Таблица 3.2. *Соответствие между именами параметров в CSS и в сценариях*

Имя параметра в таблице стилей	Свойство стиля
width	width
height	height
z-index	zIndex
font-size	fontSize
border-color	borderColor
border-top-color	borderTopColor
list-style-image	listStyleImage

3.5. Окна и фреймы

HTML-документ загружается в окно браузера. Однако можно открыть несколько таких окон и загрузить в них различные документы, а также разбить одно окно на несколько прямоугольных областей, называемых *фреймами*. В каждый такой фрейм можно загрузить отдельный документ. При этом существует возможность организовать взаимодействие между фреймами, например, с помощью действий в одном фрейме управлять содержимым другого фрейма.

Загрузка в браузер HTML-документа приводит к тому, что в обозревателе создается иерархическая объектная модель этого документа, на самом верх-

нем уровне которой находится объект `window`. Доступ к свойствам и методам данного объекта имеет уже знакомый вам синтаксис:

`window.`*`свойство`*

`window.`*`метод`*`([`*`параметры`*`])`

У объекта `window` имеется синоним `self`, используемый при обращении к окну, содержащему текущий документ. Иначе говоря, идентификатор `self` применяется в многооконных или многофреймовых системах, когда требуется указать окно с документом, в котором находится данный сценарий. Его рекомендуется вставлять, чтобы не запутаться. При запуске сценария в ссылках на объекты только текущего документа (типичная ситуация) идентификаторы `window` и `self` можно опускать.

У объекта `window` есть ряд подобъектов. Мы уже рассматривали некоторые свойства объекта события `event`, который является подобъектом объекта `window`. Другой объект, `location`, содержит информацию, полезную для работы в сети и для создания ссылок в документах с многофреймовой структурой. Кроме этого, свойство `href` объекта `location` применяется для загрузки документа в текущее окно:

`window.location.href="`*`URL-адрес_документа`*`"`

Данный способ загрузки документов в текущее окно браузера доступен во всех версиях Internet Explorer и Netscape Navigator. В Internet Explorer можно использовать также и метод `navigate()`:

`window.navigate("`*`URL-адрес_документа`*`")`

3.5.1. Создание новых окон

Главное окно браузера создается автоматически, когда пользователь запускает браузер, а также при открытии документа с определенным URL-адресом или другого файла. В HTML открыть документ в новом окне можно с помощью атрибута `TARGET` тега ссылки `<A HREF=...>`. Например:

```
<A HREF="http://www.yandex.ru" TARGET="newWindow">Яndex</A>
```

С помощью сценария можно создать любое количество окон. Для этого применяется метод `open()`:

`window.open([`*`параметры`*`])`

Этому методу передаются следующие необязательные параметры:

- адрес документа, который нужно загрузить в создаваемое окно;
- имя окна (как имя переменной);
- строка описания свойств окна (`features`).

В строке свойств записываются пары "*свойство=значение*", которые отделяются друг от друга запятыми. В табл. 3.3 приведен список свойств окна, передаваемых в строке `features`. Значения `yes` и `no` можно заменить числовыми эквивалентами 1 и 0 соответственно.

Таблица 3.3. *Свойства окна, передаваемые в строке* `features`

Свойство	Значения	Описание
`channelmode`	`yes`, `no`, `1`, `0`	Показывает элементы управления **Channel**
`directories`	`yes`, `no`, `1`, `0`	Включает кнопки каталога
`fullscreen`	`yes`, `no`, `1`, `0`	Полностью разворачивает окно
`height`	Число	Высота окна в пикселах
`left`	Число	Положение по горизонтали относительно левого края экрана в пикселах
`location`	`yes`, `no`, `1`, `0`	Текстовое поле **Address**
`menubar`	`yes`, `no`, `1`, `0`	Стандартные меню браузера
`resizeable`	`yes`, `no`, `1`, `0`	Может ли пользователь изменять размер окна
`scrollbars`	`yes`, `no`, `1`, `0`	Горизонтальная и вертикальная полосы прокрутки
`status`	`yes`, `no`, `1`, `0`	Стандартная строка состояния
`toolbar`	`yes`, `no`, `1`, `0`	Включает панели инструментов браузера
`top`	Число	Положение по вертикали относительно верхнего края экрана в пикселах
`width`	Число	Ширина окна в пикселах

Примеры:

```
window.open("mypage.htm","NewWin", "height=150, width=300")
window.open("mypage.htm")

strfeatures="top=100,left=15,width=400, height=200, location=no, menu-
bar=no"
window.open("www.admiral.ru/~dunaev",
            "Сам себе Web-дизайнер", strfeatures)
```

Вместо третьего параметра (строки `strfeatures`) можно использовать значение `true`. В этом случае указанный документ загружается в уже существующее окно, вытесняя предыдущий. Например:

```
window.open("mypage.htm","NewWin", true)
```

Метод `window.open()` возвращает ссылку на объект окна. Эту ссылку можно сохранить в переменной, чтобы потом использовать, например, при закрытии окна.

Для закрытия окна служит метод `close()`. Однако выражение `window.close()` или `self.close()` закрывает главное окно, а не дополнительное, которое вы создали методом `open()`. В этом случае как раз и необходима ссылка на созданное окно. Эту ссылку следует сохранить в глобальной переменной, чтобы иметь доступ к ней до тех пор, пока главный документ загружен в браузер. Вот пример:

```
var objwin=window.open("mypage.htm","Моя страница")
objwin.close()
```

Метод `window.open()` открывает новое независимое окно как экземпляр браузера. В этом случае при закрытии главного окна браузера новое окно остается открытым. Независимые окна называют еще *немодальными* (modalless). Однако можно создать и так называемое *модальное окно*. Пока открыто модальное окно, пользователь не может обратиться к другим окнам, в том числе и к главному. Так обычно работают стандартные диалоговые окна. Например, окна, создаваемые методами `alert()`, `prompt()` и `confirm()`, являются модальными. В модальное окно можно загрузить любой документ.

Для создания модального окна используется метод `showModalDialog()`. Так же, как и метод `open()`, он принимает в качестве параметров адрес документа (файла), имя окна и строку свойств. Однако формат этой строки иной. В частности, параметры в строке разделяются точкой с запятой, размеры окна и координаты его верхнего левого угла требуют указания единиц измерения (например, `px` — пикселы). Кроме того, данный метод не возвращает ссылку на объект окна, поскольку она не нужна для модального окна.

В табл. 3.4 приведен список передаваемых в строке `features` свойств окна, созданного методом `showModalDialog()`.

Таблица 3.4. *Свойства окна, созданного методом* `showModalDialog()`

Свойство	**Значения**	**Описание**
`order`	`thick`, `thin`	Размер рамки вокруг окна (толстая или тонкая)
`center`	`yes`, `no`, `1`, `0`	Выравнивание окна по центру главного
`dialogHeight`	Число с единицей измерения	Высота окна
`dilalogLeft`	Число с единицей измерения	Горизонтальная координата

Таблица 3.4 *(окончание)*

Свойство	Значения	Описание
dialogTop	Число с единицей измерения	Вертикальная координата
dialogWidth	Число с единицей измерения	Ширина окна
font	Строка таблицы стилей	Стиль, определенный по умолчанию для окна
font-family	Строка таблицы стилей	Вид шрифта, определенный по умолчанию для окна
font-size	Строка таблицы стилей	Размер шрифта, определенный по умолчанию для окна
font-style	Строка таблицы стилей	Тип шрифта, определенный по умолчанию для окна
font-variant	Строка таблицы стилей	Вариант шрифта (обычный или курсив), определенный по умолчанию для окна
font-weight	Строка таблицы стилей	Толщина шрифта, определенная по умолчанию для окна
help	yes, no, 1, 0	Включение кнопки **Справка** (Help) в заголовок окна
maximize	yes, no, 1, 0	Включение кнопки **Развернть** (Maximize) в заголовок окна
minimize	yes, no, 1, 0	Включение кнопки **Свернуть** (Minimize) в заголовок
font-weight	Строка таблицы стилей	Толщина шрифта, определенная по умолчанию для окна
help	yes, no, 1, 0	Включение кнопки **Справка** (Help) в заголовок окна
maximize	yes, no, 1, 0	Включение кнопки **Развернть** (Maximize) в заголовок окна
minimize	yes, no, 1, 0	Включение кнопки **Свернуть** (Minimize) в заголовок

Пример — в листинге 3.31.

Листинг 3.31. Использование свойств модального окна

```
<HTML>
  <BUTTON onclick="return OpenWin1()">Открыть окно 1</BUTTON>
  <BUTTON onclick="return OpenWin2()">Открыть окно 2</BUTTON>
  <SCRIPT>
    var newWindow    // глобальная переменная для ссылки на окно
```

```
    function OpenWin1(){              // Открытие 1-го окна
      window.status="Первое окно"     /* статусная строка главного окна */
      strfeatures="top=100,left=50,width=300,height=270,toolbar=no"
      window.open("1.htm","win1",strfeatures)
    }
    function OpenWin2(){              // Открытие 2-го модального окна
      window.status="Второе окно "    /* статусная строка главного окна */
      strfeatures="dialogWidth=500px;dialogHeight=320px," +
                  "border=thin;help=no"
      window.showModalDialog("2.htm","win2", strfeatures)
   }
   function CloseWin1(){              // Закрытие 1-го окна
     if (newWindow){                  // если 1-е окно открыто
       newWindow.close()              // закрыть 1-е окно
       newWindow=null                 // очистить ссылку на 1-е окно
       window.status=""
     }
    }
  </SCRIPT>
</HTML>
```

3.5.2. Фреймы

Фрейм представляет собой прямоугольную область окна браузера, в которую можно загрузить HTML-документ. Разбиение окна на фреймы происходит с помощью HTML-кода, содержащегося в отдельном HTML-файле, который называется *установочным*. Установочный файл содержит только теги разметки фреймов, т. е. их относительное расположение, размеры, ссылки на загружаемые в них документы и другие параметры фреймов. В нем не должны присутствовать теги других элементов и текстовая информация, за исключением разве что тегов `<META>`. Подробно фреймы описаны в *разд. 1.2.5*. В установочном файле можно написать сценарий, создающий фреймовую структуру документа.

Отношения между фреймами и главным окном

При разбиении окна на два фрейма в объектной модели браузера возникает иерархия объектов, в которой главное окно браузера представляется в виде главного, родительского фрейма, а созданные два фрейма являются дочерними фреймами или, иначе, фреймами-потомками. При разбиении любого из этих фреймов на следующие два фрейма, последние являются дочерними для

исходного. Итак, между фреймами возникает отношение "родители — потомки" ("родители — дочки").

При запуске браузера объект главного окна `window` формируется автоматически. Если в это окно загружается документ с фреймовой структурой, то главное окно получает статус родительского фрейма по отношению ко всем остальным. С другой стороны, каждый тег `<FRAME>` внутри контейнера `<FRAMESET>` создает собственный объект `window`, в который загружается соответствующий документ. Каждому из фреймов соответствует свой объект `document`. С точки зрения объекта `document`, ему соответствует единственный контейнер — фрейм. Хотя родительский объект и не виден пользователю, он все равно присутствует в объектной модели.

Итак, в вершине иерархии находится окно браузера `window`. Оно поделено на два фрейма с именами, например, `frame1` и `frame2`. Объект `window` является родительским по отношению к дочерним `frame1` и `frame2` (рис. 3.5). Для ссылки на родительское окно используется ключевое слово `parent`.

Допустим, пользователь активизирует некоторую ссылку в первом фрейме, но соответствующий документ должен загрузиться не в этот же фрейм, а в другой. Для решения данной задачи необходимо рассмотреть три случая:

- главное окно получает доступ к фрейму-потомку;
- фрейм-потомок получает доступ к родительскому (главному) окну;
- фрейм-потомок получает доступ к другому фрейму-потомку.

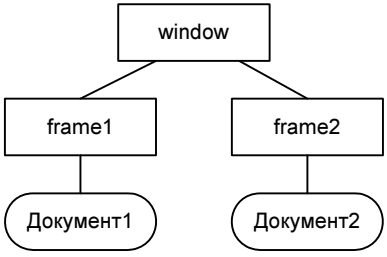


Рис. 3.5. Родительское окно и два фрейма

Между объектом `window` и объектами `frame1` и `frame2` существует прямая связь "родитель — потомок". Поэтому если вы пишете сценарий в установочном файле, создающем эти фреймы, то к фреймам можно обращаться по имени, как это показано на рис. 3.6.

Иногда может потребоваться получить доступ к родительскому окну. Например, это бывает необходимо, если вы хотите при следующем переходе убрать все фреймы. Удаление фреймов означает лишь загрузку нового документа

вместо содержащего фреймы — в рассматриваемом случае загрузку документа в родительское окно. Это можно сделать с помощью доступа к родительскому окну на основе свойства `parent` (рис. 3.7). Чтобы загрузить новый документ, следует использовать объект `location` из родительского окна (заметим, что каждый фрейм имеет собственный объект `location`): требуется внести в `location.href` родительского окна новый URL-адрес. Таким образом, чтобы загрузить новый документ в родительское окно, нужно записать в сценарии следующее:

```
parent.location.href="URL-адрес_документа"
```

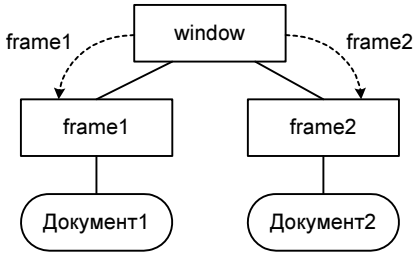


Рис. 3.6. Обращение к фреймам по имени

Наконец, довольно часто необходимо решать задачу обеспечения доступа из одного фрейма-потомка к другому (рис. 3.8). Например, находясь в одном фрейме-потомке, можно записать что-нибудь в другой фрейм-потомок. Однако между фреймами-потомками не существует прямой связи. Поэтому мы не можем просто вызвать фрейм `frame2` по имени, находясь во фрейме `frame1`. Фрейм `frame1` ничего не знает о `frame2`, но `frame2` существует с точки зрения родительского окна. По этой причине мы должны сначала обратиться к родительскому окну, затем — к `frame2` и лишь потом к объекту `document`, который разместился во втором фрейме:

```
parent.frame2.document.write("Привет от первого фрейма!")
```

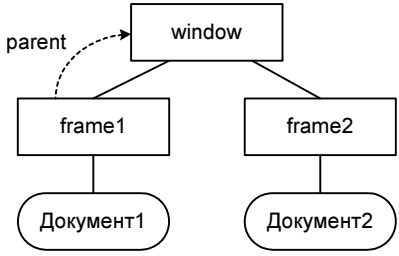


Рис. 3.7. Доступ к родительскому окну на основе свойства `parent`

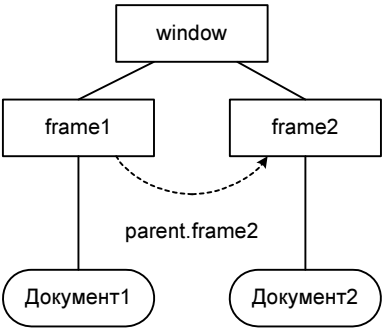


Рис. 3.8. Обеспечение доступа из одного фрейма-потомка к другому

Рассмотрим, как можно изменить элемент в одном фрейме из другого. При щелчке на тексте из правого фрейма в левом изменяется один из текстовых элементов. В листингах 3.32—3.34 показаны HTML-коды соответственно для установочного файла, левого и правого фреймов.

Листинг 3.32. Установочный файл

```
<HTML>
  <HEAD><TITLE>Фреймы</TITLE></HEAD>
  <FRAMESET COLS="40%,*">
    <frame src="документ1.htm" name="LEFT" scrolling=no>
    <frame src="документ2.htm" name="RIGHT">
  </FRAMESET>
</HTML>
```

Листинг 3.33. Документ в левом фрейме с именем `LEFT`

```
<HTML>
  Это левый фрейм
  <H1 ID="XXX">Привет</H1>
</HTML>
```

Листинг 3.34. Документ в правом фрейме с именем `RIGHT`

```
<HTML>
  <SCRIPT>
    function change(){
```

```
      parent.LEFT.document.all.XXX.innerText="Что угодно?"
    }
  </SCRIPT>
  Это правый фрейм
  <H1 onclick="change()">Щелкни здесь</H1>
</HTML>
```

В теле функции `change()`, которая делает нужные изменения, происходит обращение к левому фрейму с именем `LEFT` (задается в установочном HTML-файле) через `parent`. Далее, `document.all.XXX` обеспечивает доступ к элементу с идентификатором `"XXX"` (в примере это заголовок 1-го уровня). Здесь `all` — коллекция всех элементов документа. Собственно изменение элемента происходит за счет присвоения значения свойству `innerText` (в примере это "Что угодно?").

Обратите внимание, что изменения в одном фрейме по событию в другом происходят без перезагрузки HTML-документа. Для изменения элементов можно использовать, кроме `innerText`, свойства `outerText`, `innerHTML` и `outerHTML`.

Описанным выше способом вы можете организовать, например, такой сценарий: щелчок на миниатюре (маленьком графическом изображении) в одном фрейме выводит в нем же полномасштабную картинку, а в другом — краткое описание.

Фреймы удобно использовать при создании навигационных панелей. В одном фрейме располагаются ссылки, а второй предназначен для отображения документов, вызываемых при активизации соответствующих ссылок. При активизации ссылки документ загружается не в тот же фрейм, где находятся ссылки, а в другой.

Рассмотрим пример создания навигационной панели. Окно браузера разделено на два фрейма (листинг 3.35): первый (`menu`) играет роль навигационной панели, а второй (`main`) — окно для отображения документов.

Листинг 3.35. Установочный файл frames.htm

```
<HTML>
  <FRAMESET COLS="25%,75%">
    <FRAME SRC="menu.htm" NAME="menu">
    <FRAME SRC="start.htm" NAME="main">
  </FRAMESET>
</HTML>
```

Здесь start.htm — документ, который будет первоначально показан во фрейме `main`; menu.htm — навигационная панель (листинг 3.36).

Листинг 3.36. Документ menu.htm

```
<HTML>
  <SCRIPT >
    function load(url){
      parent.main.location.href=url;
    }
  </SCRIPT>
  <BODY>
    <A HREF="javascript:load('первый.htm')">Первый</A>
    <A HREF="второй.htm" TARGET="main">Второй </A>
    <A HREF="третий.htm" TARGET="top">Третий </A>
  </BODY>
</HTML>
```

Здесь показано несколько способов загрузки новой страницы во фрейм `main`. В первой ссылке используется функция `load()`. Вместо атрибута `TARGET` указания на фрейм выполняет функция. Функции `load()` в качестве параметра передается строка `'первый.htm'`, указывающая, какой файл следует загрузить. При этом место, куда он будет загружен, определяется самой функцией `load()`. Во второй ссылке используется атрибут `TARGET`. Третья ссылка показывает, как можно избавиться от фреймов. Чтобы удалить фреймы с помощью функции `load()`, достаточно написать в ней следующую строку:

```
parent.location.href=url
```

Атрибут `TARGET` в теге ссылки `<A HREF=...>` обычно применяется в тех случаях, когда требуется загрузить одну страницу в один фрейм. Язык сценариев используют, если необходимо при активизации ссылки выполнить несколько действий, например, загрузить несколько страниц в разные фреймы.

Для ссылок из родительского окна к объектам его дочерних фреймов можно воспользоваться коллекцией `frames` всех фреймов. Коллекция фреймов представляет собой массив объектов фреймов. Обратиться к конкретному фрейму из этой коллекции можно по индексу или по имени фрейма, указанному в качестве значения атрибута `NAME` в теге `<FRAME>`:

window.frames[*индекс*]

window.*имя_фрейма*

Заметим, что индекс 0 соответствует первому дочернему фрейму в порядке, определенном следованием тегов `<FRAME>` в контейнере `<FRAMESET>`.

Приведенные выше шаблоны ссылок на фреймы из родительского окна используются как префиксы в полных ссылках к объектам, содержащимся во фреймах. При этом следует помнить, что если нас интересуют объекты документа, загруженного во фрейм, то прежде чем обратиться к ним, нужно упомянуть объект `document`. Например:

```
window.frames(0).document.all.Myinput.value
window.LEFT.document.all.Myinput.value
```

Ссылка из дочернего фрейма непосредственно на родительский производится с помощью ключевого слова `parent`. Если имеется еще один фрейм более высокого уровня, то ссылка на него выглядит так: `parent.parent`. Аналогичным образом можно построить ссылку из дочернего фрейма до прапрадедушки. Чтобы сразу обратиться к родительскому окну, находящемуся на вершине иерархии, можно использовать ключевое слово `top`.

Совет

При использовании ссылки `top` следует учитывать то обстоятельство, что ваш сайт может быть загружен в один из фреймов другого сайта. В этом случае указанный вами объект `top` окажется совсем иным и ссылки, построенные с его использованием, не будут правильно работать. Поэтому рекомендуется использовать `parent` для ссылок на вышестоящий фрейм (окно).

Предотвращение использования фреймов

Ссылки `top` и `self` можно использовать для предотвращения отображения вашего сайта внутри фреймов другого сайта. Для этого необходимо, чтобы документ верхнего уровня проверял, в какое окно он загружен. Это должно быть самое верхнее (`top`) или родительское (`parent`) окно. Если это действительно так, ссылка на свойство `top` совпадает со ссылкой `self` на текущее окно. При несовпадении этих значений документ следует перезагрузить заново, но уже в окно верхнего уровня. Сценарий, выполняющий эту работу, необходимо разместить в начале документа. Вот его код:

```
<SCRIPT>
  if (top != self)
    top.location=location
</SCRIPT>
```

Проверка загрузки всех фреймов

При посещении вашего многофреймового сайта пользователь может сделать закладку (поместить в папку Избранное) только на один фрейм, в то время как все средства навигации по сайту находятся в другом фрейме. При следующем посещении в браузер загрузится усеченный вариант вашего сайта. А от этого может пострадать как пользователь, так и ваша репутация.

Следующий простой сценарий проверяет, загружается ли Web-страница в своем наборе фреймов, сравнивая URL-адреса окна `top` и текущего окна. Если они совпадают, то загружен лишь один фрейм, и необходимо загрузить весь набор фреймов, определенный в установочном файле, например, frameset.htm:

```
<SCRIPT>
  if (top.location.href == window.location.href)
    top.location.href = "frameset.htm"
</SCRIPT>
```

Однако при выполнении этого сценария в окне браузера отобразятся документы, заданные в установочном файле, и, возможно, среди них не окажется запрошенного пользователем. Поэтому для каждого HTML-документа, отображаемого во фрейме, следует предусмотреть отдельный установочный файл, в котором задана загрузка этого документа.

Допустим, документы, предназначенные для отображения во фрейме, хранятся в файлах file1.htm, file2.htm, ..., file*N*.htm. Тогда следует создать установочные файлы frameset1.htm, frameset2.htm, ..., frameset*N*.htm, в которых задана загрузка во фрейм соответствующего документа. Например, установочный файл frameset1.htm для двухфреймовой структуры может иметь следующий вид:

```
<FRAMESET rows = "128,*" border = 0>
  <FRAME SRC = "navpane.htm" NAME = "FRAME1" scrolling = no>
  <FRAME SRC = "file1.htm"   NAME = "FRAME2" scrolling = auto>
</FRAMESET>
```

Здесь HTML-документ из файла file1.htm предполагается загружать во второй фрейм. При этом файл file1.htm где-то в самом начале должен содержать такой сценарий:

```
<SCRIPT>
  if (top.location.href == window.location.href)
    top.location.href = "frameset1.htm"
</SCRIPT>
```

Аналогичным образом формируются остальные установочные и информационные файлы сайта.

3.5.3. Плавающие фреймы

Для вставки одного HTML-документа в тело другого средствами браузера пользователя служит контейнерный тег `<IFRAME>`:

```
<IFRAME SRC="адрес_документа"></IFRAME>
```

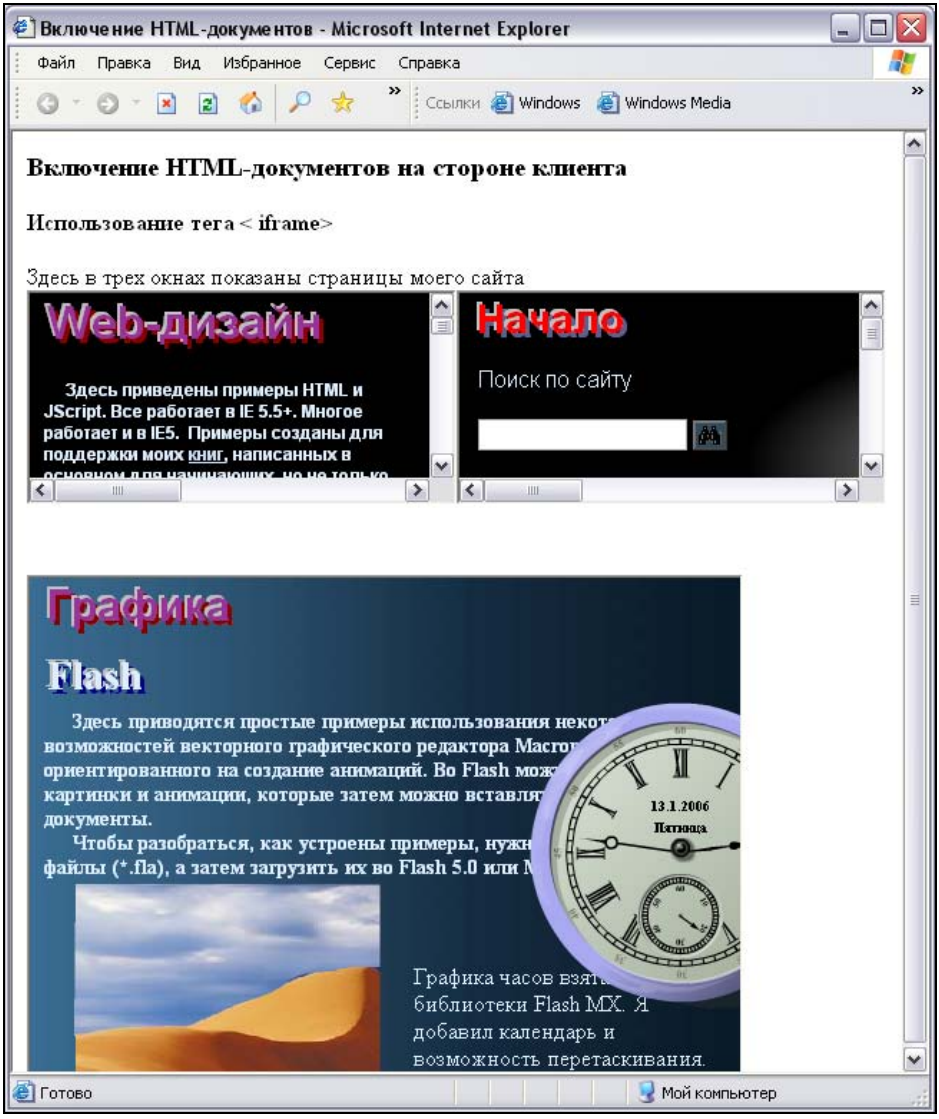


Рис. 3.9. Страница, содержащая три плавающих фрейма с документами из внешних файлов

Элемент, задаваемый этим тегом, можно позиционировать с помощью параметров таблицы стилей (тег `<STYLE>` или атрибут `STYLE`). Если его положение не указано явно, то он позиционируется в соответствии с положением `<IFRAME>` в HTML-коде. Внешне этот элемент выглядит как прямоугольная область (с полосами прокрутки или без них), в которой отображается документ из некоторого HTML-файла. Такие окна называют *плавающими фреймами*.

HTML-документы, загружаемые в плавающие фреймы, могут иметь сценарии и прочие средства, присущие любому HTML-документу. Плавающий фрейм ведет себя почти так же, как и обычный фрейм. Если он создан, то его можно найти в коллекции `frames`. Однако следует иметь в виду, что если определены события, одинаковые для фрейма и для загруженного в него документа, то на них будут откликаться обработчики документа, а не фрейма.

В листинге 3.37 приведен пример для загрузки трех различных HTML-файлов в плавающие фреймы (рис. 3.9). Это HTML-документы из сайта автора (**www.admiral.ru/~dunaev**). Для третьего фрейма без полос прокрутки определены обработчики событий `onmouseover` (наведение указателя мыши) и `onmouseout` (уход указателя мыши).

Примечание

Для документа во фрейме такие события не заданы и поэтому мы вполне можем рассчитывать на изменение размеров фрейма при наведении на него или уходе с него указателя мыши, как это определено в функциях-обработчиках указанных событий. Иначе говоря, документ не перехватит эти события.

Листинг 3.37. Три плавающих фрейма

```
<HTML >
  <HEAD><TITLE>Включение HTML-документов</TITLE></HEAD>
  <SCRIPT>
    function big(){       // увеличение размеров
      document.all.myfr.style.width=
        parseInt(document.all.myfr.style.width)+100
      document.all.myfr.style.height=
        parseInt(document.all.myfr.style.height)+100
    }
    function small(){     // уменьшение размеров
      document.all.myfr.style.width=
        parseInt(document.all.myfr.style.width)-100
      document.all.myfr.style.height=
```

```
        parseInt(document.all.myfr.style.height)-100
    }
  </SCRIPT>
  <H3>Включение HTML-документов на стороне клиента</H3>
  <H4>Использование тега < iframe></H4>
  Здесь в трех окнах показаны страницы моего сайта<br>
  <IFRAME SRC="examples.htm" ></IFRAME>
  <IFRAME SRC="i_is.htm" ></IFRAME >
  <P>
  <IFRAME ID=myfr SRC="graphic.htm" STYLE="position:absolute;
                                top:310; width:500; height:350"
          SCROLLING="no"
          onmouseover="big()"
          onmouseout="small()">
  </IFRAME >
</HTML >
```

3.5.4. Всплывающие окна

Всплывающее окно не имеет органов управления и располагается над документом, в котором оно было создано, в том числе и над диалоговыми окнами. Щелчок кнопкой мыши где-либо вне этого окна или вызов другого приложения приводит к его закрытию. Оно также не отображается на панели задач Windows.

В сценарии сначала создают всплывающее окно, а затем заполняют его содержимым. При этом необходимо следить, чтобы содержимое вписывалось в заданные границы окна, поскольку это окно не имеет полос прокрутки. Такие всплывающие окна удобно использовать в качестве больших всплывающих подсказок для вывода контекстной справочной информации. Вместе с тем можно придумать более интересное применение этого средства.

Всплывающим окнам соответствует объект `popup`. Он создается только в сценарии для браузера IE5.5+ с помощью метода `window.createPopup()`. Далее этому объекту назначаются требуемые свойства посредством объекта `style`. Так, например, для всплывающего окна можно назначить свойства окаймляющей рамки (границы — `border`), цвета границы (`color`), цвета фона (`background`), отступов (`padding`). В листинге 3.38 приведен пример типичной последовательности выражений сценария, создающего всплывающее окно, задающего его параметры и отображающего на экране.

Листинг 3.38. Всплывающее окно

```
<HTML>
  <SCRIPT>
    var mypopup=window.createPopup()    /* создание всплывающего окна */
    var popupBody=mypopup.document.body /* сохранили в переменной
                                           для краткости */
    /* Параметры окна */
    popupBody.style.border="solid 1px green"    // граница окна
    popupBody.style.padding="5px"               // отступ текста
    popupBody.style.color="green"               // цвет текста в окне
    popupBody.style.background="ffffd0"         // цвет фона окна

    /* Заполнение содержимым: */
    popupBody.innerHTML="<H3>Здесь какой-то текст</H3>"
    mypopup.show(200,100,400,70,document.body)    /* вывод всплывающего
                                                     окна */
  </SCRIPT>
</HTML>
```

Внимание

Всплывающее окно может появляться не только в клиентской области браузера.

Свойства объекта `popup`:

- `document` — свойство, имеющее в качестве значения ссылку на документ, содержащийся во всплывающем окне; через него можно задать ряд параметров как самого окна, так и его содержимого:

  ```
  mypopup.document.body.style.border="solid 4px black"   // границы окна
  mypopup.document.body.style.background="yellow"      // цвет фона окна
  mypopup.document.body.style.color="blue"         // цвет текста в окне
  ```

- `isOpen` — пока всплывающее окно отображается на экране, это свойство имеет значение `true`, в противном случае — `false`; применяется в выражениях сценария, которые выполняются при уже открытом окне.

Методы объекта `popup`:

- `show(`*`left`*`, `*`top`*`, `*`width`*`, `*`height`*`[, `*`позиционирование`*`])` — отображает всплывающее окно после создания объекта `popup` с помощью метода

`window.createPopup()` и заполнения его содержимым. Если окно исчезло из-за того, что пользователь щелкнул где-то в поле браузера, то для его повторного открытия следует заново выполнить метод `show()`. Первые четыре параметра метода задают координатное позиционирование и размеры окна. Параметры *`left`* и *`top`* определяют координаты верхнего левого угла окна относительно экрана монитора, а не окна браузера. Последний необязательный параметр позволяет задать другое координатное пространство как ссылку на элемент HTML-документа. Например, указав `document.body`, вы ограничиваете координатное пространство окном браузера.

- `hide()` — позволяет скрыть созданное и отображаемое всплывающее окно.

Заполнение всплывающего окна содержимым производится с помощью свойства `innerHTML`, принимающего в качестве значения строку, содержащую теги HTML *(см. разд. 3.3.4)*. Таким образом, в окно можно вставлять не только тексты, но и картинки, и другие элементы. Например:

```
mypopup.document.body.innerHTML="<IMG SRC='picture.jpg'>"
```

В листинге 3.39 приведен пример HTML-документа с двумя кнопками, щелчок на которых вызывает одну и ту же функцию `pop(xcontent)`, открывающую всплывающее окно и заполняющую его содержимым. Содержимое задано параметром `xcontent`. Разнообразие вносится только значением параметра функции `pop()`.

Листинг 3.39. HTML-документ с двумя кнопками *(вариант 1)*

```
<HTML>
  <H3>Всплывающие окна</H3>
  <BUTTON onclick =
     "pop('Во первых строках своего письма я рад вам сообщить')">Просто текст
  </BUTTON>
  <BR><BR>
  <BUTTON onclick="pop('<IMG SRC=picture.jpg>Это картинка')">
     Картинка с   текстом
  </BUTTON>
  <SCRIPT>
    var mypopup=window.createPopup()
    var popupBody=mypopup.document.body
    popupBody.style.border="solid 2px green"
    popupBody.style.padding="5px"
    popupBody.style.color="blue"
    popupBody.style.background="ffffd0"
```

```
    function pop(xcontent){
      popupBody.innerHTML="<p>" + xcontent + "</p>"
      mypopup.show(130,90,400,200,document.body)
    }
  </SCRIPT>
</HTML>
```

В рассмотренном выше примере при вызове функции `pop()` каждый раз создается одно и то же всплывающее окно, содержимое которого может изменяться. Поскольку позиционирование, размеры и цвет окна не изменяются, то можно написать более экономный код за счет вынесения выражений его создания и параметризации за пределы определения функции `pop()` — листинг 3.40.

Листинг 3.40. HTML-документ с двумя кнопками *(вариант 2)*

```
<HTML>
  <H3>Всплывающие окна</H3>
  <BUTTON onclick=
"pop('Во первых строках своего письма я рад вам сообщить')">
Просто текст
  </BUTTON>
  <BR><BR>
  <BUTTON onclick="pop('<IMG SRC=picture.jpg>Это картинка')">
     Картинка с   текстом
  </BUTTON>
  <SCRIPT>
    var mypopup=window.createPopup()
    var popupBody=mypopup.document.body
    popupBody.style.border="solid 2px green"
    popupBody.style.padding="5px"
    popupBody.style.color="blue"
    popupBody.style.background="ffffd0"

    function pop(xcontent){
      popupBody.innerHTML="<p>" + xcontent + "</p>"
      mypopup.show(130,90,400,200,document.body)
    }
  </SCRIPT>
</HTML>
```

Здесь использовался объект `style` для назначения параметров стиля, таких как граница, отступы и цвет.

3.6. Загрузка изображений

Большинство Web-страниц содержат графические элементы, которые используются не только для украшения страниц, но и в качестве информационного наполнения: иллюстрированные каталоги товаров, схемы, чертежи, географические карты, фотогалереи и т. п. Если файлы с графикой велики и/или их много, то загрузка такой страницы в браузер может потребовать слишком много времени. Для ускорения загрузки используют специальные приемы. Так, нередко сначала загружают изображения с низким разрешением из небольших по объему файлов. На их основе создаются ссылки на файлы с графикой полноценного разрешения, которые загружаются только при щелчке на ссылке. Можно также использовать в теге `<IMG>`, кроме атрибута `SRC`, атрибут `LOWSRC`, позволяющий загрузить сначала изображение с низким разрешением, а затем, по мере приема, заменить ее картинкой с большим разрешением. Здесь мы рассмотрим способ ускорения загрузки графики, предоставляемый JavaScript.

С помощью сценария можно организовать предварительную загрузку изображений в кэш-память браузера, не отображая их на экране. Это особенно эффективно при начальной загрузке страницы. Пока изображения загружаются в память, оставаясь невидимыми, пользователь может рассматривать текстовую информацию, его не раздражает медленное появление графических элементов. В результате загрузка страницы не вызывает у пользователя отрицательных эмоций.

Для предварительной загрузки изображения следует создать его объект в памяти браузера. Такой объект несколько отличается от объекта изображения, создаваемого с помощью тега `<IMG>`. Как и все объекты, создаваемые сценариями, объект изображения не выводится в окне браузера. Однако его наличие в коде документа уже обеспечивает загрузку самого изображения при загрузке документа. Чтобы создать в памяти объект изображения, необходимо в сценарии выполнить следующее выражение:

```
myimg=new Image(ширина, высота)
```

Параметры функции-конструктора объекта определяют размеры изображения и должны соответствовать значениям атрибутов `WIDTH` и `HEIGHT` тега `<IMG>`, который используется для отображения предварительно загруженной картинки.

Для созданного в памяти объекта изображения `myimg` можно задать имя или, в общем случае, URL-адрес графического файла. Это делается с помощью свойства `src`:

```
myimg.src="URL-адрес_изображения"
```

Данное выражение предписывает браузеру загрузить в кэш-память указанное изображение, но не выводить его. После загрузки в кэш-память всех изображений и загрузки всего документа можно сделать их видимыми. Для этого нужно свойству `src` элемента `<IMG>` присвоить значение этого же свойства объекта изображения в кэш-памяти. Например,

```
document.images[0].src=myimg.src
```

Здесь слева от оператора присваивания указано свойство `src` первого в документе элемента, соответствующего тегу `<IMG>`, а справа — свойство `src` объекта изображения в кэш-памяти.

Теперь рассмотрим в качестве примера HTML-документ, в котором отображается список названий графических элементов и одно исходное изображение. Щелчок на элементе списка приводит к выводу соответствующего изображения. Все графические элементы из этого списка предварительно загружаются в кэш-память и поэтому быстро отображаются при выборе.

Список создается с помощью контейнерного тега `<SELECT>`, содержащего теги `<OPTION>`. Соответствующий фрагмент HTML-кода мы сгенерируем с помощью сценария и запишем в текущий документ. В листинге 3.41 приведен код.

Листинг 3.41. Код HTML-документа, в котором отображается список названий графических элементов и одно исходное изображение

```
<HTML>
  <HEAD>
  <SCRIPT>
    var imgFile=new Array()       // массив имен графических файлов
    imgFile[0]="pict1.jpg"
    imgFile[1]="pict2.jpg"
    imgFile[2]="pict3.jpg"
    imgFile[3]="pict4.jpg"

    var imgName=new Array()       // массив названий картинок
    imgName[0]="Картинка1"
    imgName[1]="Картинка2"
```

```
    imgName[2]="Картинка3"
    imgName[3]="Картинка4"

    /* Создание объектов изображений и загрузка изображений в кэш */
    var imgObj=new Array()            // массив объектов изображений
    for(i=0; i < imgFile.length; i++){
      imgObj[i]=new Image(100, 200)   // создаем объект изображения
      imgObj[i].src=imgFile [i]       /* загрузка картинки в память
                                         без отображения */
    }

    function imgshow(list){  // отображение картинки при выборе из списка
      var x=list.options[list.selectedIndex].value   /* выбранный номер
                                                        картинки */
      document.all.img0.src=eval("imgObj["+ x + "].src")
    }

    /* Создание списка картинок */
    var clist="<SELECT onchange='imgshow(this)'>"
    for(i=0; i<imgFile.length; i++){
      clist+= "<OPTION VALUE=" + i + ">" + imgName[i]
    }
    clist+= "</SELECT>"
    document.write(clist)        // запись списка картинок в документ
  </SCRIPT>
  </HEAD>
  <! Исходная картинка>
  <IMG ID="img0" SRC="pict0.jpg" WIDTH=150 HEIGHT=100>
</HTML>
```

Обратите внимание, что для преобразования строки в настоящую ссылку на объект используется функция `eval()`. HTML-код, определяющий список картинок, сгенерированный сценарием, выглядит следующим образом:

```
<SELECT onchange='imgshow(this)'>
  <OPTION VALUE=0>Картинка1
  <OPTION VALUE=1>Картинка2
  <OPTION VALUE=2>Картинка3
  <OPTION VALUE=3>Картинка4
</SELECT>
```

3.7. Управление процессами во времени

В JavaScript имеется возможность периодически, через заданный интервал времени, запускать одну или несколько функций JavaScript. При этом создается эффект одновременного (параллельного) выполнения вычислительных процессов. Например, вы можете запустить несколько функций, перемещающих на экране различные видимые объекты. Эти объекты будут двигаться как бы одновременно. Иногда требуется организовать временную задержку перед выполнением какой-то функции, чтобы ранее начатый процесс успел завершиться. Все это относится к задачам управления вычислительными процессами во времени.

Для организации периодического (через заданный интервал времени) выполнения некоторого выражения или функции служит метод `setInterval()` объекта `window`. Этот метод имеет следующий синтаксис:

```
setInterval(выражение, период[, язык])
```

Первый параметр представляет собой строку, содержащую выражение (в частности, вызов функции). Второй параметр — целое число, указывающее временную задержку в миллисекундах перед последующими выполнениями выражения, заданного в первом параметре. Третий, необязательный параметр определяет язык, на котором написано выражение; по умолчанию — JavaScript.

Метод `setInterval()` возвращает некоторое целое число — идентификатор временного интервала, который может быть использован в дальнейшем для прекращения выполнения процесса, запущенного с помощью данного метода (см. далее метод `clearInterval()`).

Пусть, например, требуется, чтобы некоторая функция `myfunc()` выполнялась периодически через 0,5 с. Тогда в сценарии следует записать следующее выражение:

```
setInterval("myfunc()", 500)
```

Тот факт, что первый параметр метода `setInterval()` является строкой, обуславливает некоторые особенности передачи параметров периодически вызываемой функции. Если периодически вызываемая функция принимает параметры, то мы должны сначала сформировать строку, содержащую имя этой функции, круглые скобки, значения параметров и запятые между ними, а затем передать ее в качестве первого параметра методу `setInterval()`. В следующем примере показано, как передать методу `setInterval()` функцию

с двумя параметрами `param1` и `param2`, значения которых определены в другом месте сценария:

```
var xstr="myfunc(" + param1 + "," + param2 +")"
setInterval(xstr, 500)
```

Выражение, переданное методу `setInterval()`, будет периодически выполняться сколь угодно долго. Если это выражение осуществляет, например, приращение координат какого-нибудь видимого элемента документа, то этот элемент станет перемещаться в окне браузера.

Для остановки запущенного временного процесса служит метод

`clearInterval(`*идентификатор*`)`

который принимает в качестве параметра целочисленный идентификатор, возвращаемый соответствующим методом `setInterval()`, например:

```
var myproc=setInterval("myfunc(), 100")
if (confirm("Прервать процесс ?"))
  clearInterval(proc)
```

Содержательный пример использования метода `setInterval()` приведен в *разд. 3.14*.

Чтобы выполнить выражение с некоторой временной задержкой, используется метод `setTimeout()`. Этот метод объекта `window` имеет следующий синтаксис:

`setTimeout(`*выражение*`, `*задержка*`[, `*язык*`])`

Первый параметр представляет собой строку, содержащую выражение (в частности, вызов функции). Второй параметр — целое число, указывающее временную задержку в миллисекундах выполнения выражения, заданного в первом параметре. Третий, необязательный параметр указывает язык, на котором написано выражение; по умолчанию — JavaScript.

Метод `setTimeout()` возвращает некоторое целое число — идентификатор временного интервала, который может быть использован в дальнейшем для отмены задержки выполнения процесса, запущенного с помощью данного метода (см. ниже метод `clearTimeout()`).

Пусть, например, требуется, чтобы некоторая функция `myfunc()` выполнялась спустя 1 с. Тогда в сценарии следует записать следующее выражение:

```
setTimeout("myfunc()", 1000)
```

Внимание

Помните, что это выражение не задерживает выполнение всех последующих выражений сценария. Оно лишь задерживает выполнение функции `myfunc()`.

Для отмены задержки процесса, запущенного с помощью метода `setTimeout()`, используется метод

`clearTimeout(`*идентификатор*`)`

который принимает в качестве параметра целочисленный идентификатор, возвращаемый соответствующим методом `setTimeout`.

В следующем HTML-документе имеются две кнопки (листинг 3.42). Щелчок на кнопке **Пуск** открывает через 5 с новое окно и загружает в него документ mypage.htm. Однако это действие можно отменить с помощью кнопки **Отмена**, если щелкнуть на ней, пока окно еще не открыто.

Листинг 3.42. Использование временной задержки

```
<HTML>
  <BUTTON ID="start">Пуск</BUTTON>
  <BUTTON ID="stop">Отмена</BUTTON>
  <SCRIPT>
    var myproc
    function start.onclick(){
      myproc=setTimeout("window.open('mypage.htm')", 5000)
    }
    function stop.onclick(){
      clearTimeout(myproc)
    }
  </SCRIPT>
</HTML>
```

Внимание

При использовании методов `setInterval()` и `setTimeout()` следует иметь в виду, что их вторые параметры задают лишь приблизительные значения временных задержек.

3.8. Работа с cookie

3.8.1. Общие сведения

Для хранения небольших объемов информации на диске компьютера пользователя в браузере предусмотрен так называемый механизм cookie. Обычно он применяется для хранения имени пользователя и пароля, который вводится

в поле формы защищенного Web-сайта, а также информации о предыдущем посещении сайта. Например, можно сохранить на диске дату последнего посещения сайта данным пользователем. При загрузке сайта эта дата сравнивается с некоторой датой, установленной автором сайта в качестве даты обновления. Если вторая (авторская) дата более поздняя, чем первая, то на Web-странице появляется соответствующая отметка, например, текст или изображение с надписью "New" или "Обновлено!". Разумеется, сравнение дат и вывод на страницу отметки производится с помощью сценария. А вот еще один пример: сценарий проверяет, посетил ли сайт определенный пользователь; если это произошло, то на страницу выводится персональное сообщение.

По существу, cookies — это единственный способ сохранения данных на диске пользователя, безопасный для него. Как известно, Web-браузеры препятствуют свободному обращению к папкам и файлам на компьютере пользователя. В настройках браузера можно установить уровень конфиденциальности, определяющий, какого рода cookies будут блокироваться. Вместе с тем, следует помнить, что многие пользователи не любят cookies-записи и полностью их блокируют и уничтожают.

Записи cookies браузер Internet Explorer сохраняет в отдельных текстовых файлах в папке Cookies. Расположение этой папки в системе Windows зависит от версии. Имя файла для записи cookies образуется на основе имени пользователя и домена того сервера, на котором создавался cookie-файл. Например, Netscape Navigator 4 для Windows создает просто один файл cookie.txt. Вообще говоря, структура данных в cookies-файлах для различных браузеров не столь существенна, поскольку, во-первых, не рекомендуется открывать и модифицировать эти файлы в текстовых редакторах, а во-вторых, браузеры Internet Explorer и Netscape Navigator используют одинаковый синтаксис чтения и записи cookie-данных, основанный на свойстве `document.cookie`.

Итак, данные в cookies-файлах организованы в виде записей. Каждую такую запись можно представить себе в виде строки, содержащей следующие элементы:

- имя записи;
- содержание записи;
- срок хранения (годности) записи;
- домен сервера, который создал запись;
- сведения о необходимости установки безопасного HTTP-соединения для доступа к записям;
- расположение документов, которым разрешен доступ к записям.

Зависимость записей от домена обеспечивает безопасность хранения так называемых *невосстанавливаемых паролей* (пар вида *имя_пользователя-пароль*), поскольку запись, созданную сервером одного домена, не может прочитать сервер с другим доменом.

Срок хранения используется браузером для автоматического удаления просроченных записей, чтобы предотвратить чрезмерное разрастание объема cookies-файлов. Впрочем, IE4+ и NN4+ ограничили объем cookies-файлов 20 записями на каждый домен.

Для записи данных в cookies-файл с помощью JavaScript используется выражение присваивания строки, содержащей cookies-данные, свойству `document.cookie`. При этом важно соблюдать формат строки:

```
document.cookie="cookieName=данные
[; expires=строка_времени_GMT]
[; path=путь]
[; domain=домен]
[; secure]"
```

Здесь квадратные скобки указывают, что заключенное в них содержимое не является обязательным (может быть опущено).

Рассмотрим элементы cookie-строки.

- `cookieName`. Каждая cookie-запись должна иметь имя и строковое значение, которое может быть и пустой строкой. Например, если требуется сохранить слово "Вася" в cookie-записи с именем `User_Name`, то соответствующее выражение JavaScript будет иметь вид:

  ```
  document.cookie="User_Name=Вася"
  ```

  При выполнении этого выражения браузер попытается найти cookie-запись с таким именем. Если он не найдет ее в текущем домене, то создаст автоматически. Если запись с таким именем уже существует, то браузер заменит ее данные новыми. Данные в этом элементе cookie-записи не должны содержать точек с запятыми, запятых и пробелов. Чтобы заменить пробелы соответствующими символами (`%20`), строка с данными предварительно обрабатывается функцией `escape()`.

- `expires`. Дата и время хранения (годности, жизни) cookie-записи должны быть представлены строкой и содержать данные по Гринвичу (GMT). Например, вычислить дату истечения срока хранения записи, месяц спустя от текущей даты, можно следующим образом:

  ```
  var expdate=new Date()          // создаем объект даты
  var monthFromNow=expdate.getTime() + (30*24*60*60*1000)
  expdate.setTime(monthFromNow)  // устанавливаем значение даты
  ```

После этого полученную дату следует привести к строковому формату GMT:

```
document.cookie="User_Name=Вася; expires=" + expdate.toGMTString()
```

Cookie-запись можно удалить и до истечения заданного срока хранения, установив новый срок, заведомо уже прошедший:

```
expdate=Thu, 01-Jan-70 00:00:01 GMT
```

Отсутствие срока хранения означает для браузера, что данная cookie-запись является временной и не записывается в файл.

Об использовании методов объекта даты *см. разд. 2.11*.

- `path`. Cookies-записи, производимые компьютером пользователя, имеют путь, принятый по умолчанию (в текущей папке). Однако можно создать копию cookie в другой папке, указав путь к ней в качестве значения этого параметра.
- `domain`. Для синхронизации cookies-данных с определенным документом или/и группой документов браузер выясняет домен текущего документа и помещает в cookie-файл записи, соответствующие этому домену. Если пользователю требуется просмотреть список всех cookies-записей, содержащихся в свойстве `document.cookie`, то он должен просмотреть все пары "имя-значение", находящиеся в cookie-файле с именем домена текущего документа. Формат представления домена должен охватывать, по крайней мере, два уровня, например, **rambler.ru**.
- `secure`. Принимает логические значения (`true` или `false`). При создании cookies-записей на стороне клиента (компьютера пользователя) этот параметр опускается.

3.8.2. Функции чтения, записи данных и удаления cookie

Данные cookie, которые можно получить с помощью сценария на JavaScript, представляют собой единственную строку — значение свойства `document.cookie`. Выбор значений отдельных элементов (параметров) cookie производится на основе анализа содержимого этой строки методами объекта `String`. Кроме того, если две и более cookies-записи (до 20) соответствуют одному и тому же домену, то в JavaScript они все равно представляются одной строкой и разграничиваются точкой с запятой и пробелом.

Мы можем написать на JavaScript функции для чтения, записи и удаления данных cookie. Рассмотрим сначала функцию `readCookie(name)`, читающую

cookies-данные, соответствующие имени записи *name*, которое передается этой функции в качестве параметра (листинг 3.43).

Листинг 3.43. Функция чтения данных cookie

```
function readCookie(name) {   // чтение cookies-данных записи
  var xname=name + "="
  var xlen=xname.length
  var clen=document.cookie.length
  var i=0
  while(i < clen){
    var j=i + xlen
    if (document.cookie.substring(i, j) == xname)return getCookieVal(j)
    i=document.cookie.indexOf(" ",1) + 1
    if (i == 0) break
  }
  return null
}
```

Функция `readCookie(name)` возвращает значение cookie-записи с именем `name` или `null`, если такая запись не найдена. В теле этой функции использована еще одна, вспомогательная, функция `getCookieVal(j)`, возвращающая декодированное значение cookies-данных. Декодирование производится с помощью встроенной функции `unescape()`. Дело в том, что cookie-запись должна представлять собой кодированную строку, полученную путем обработки встроенной функцией `escape()`, чтобы, в частности, заменить пробелы специальными символами (`%20`). Код этой функции приведен в листинге 3.44.

Листинг 3.44. Функция `getCookieVal()`, вызываемая из функции `readCookie()`

```
function getCookieVal(n){   /* вспомогательная функция, вызываемая
                               из readCookie() */
  var endstr=document.cookie.indexOf(";", n)
  if (endstr == -1) endstr=document.cookie.length
  return unescape(document.cookie.substring(n, endstr))
}
```

Теперь рассмотрим функцию `writeCookie()`, позволяющую создать или обновить cookie-запись (листинг 3.45).

Эта функция принимает следующие параметры:

- `name` — строка, содержащая имя cookie-записи (обязательный параметр);
- `value` — строка, содержащая значение cookie (обязательный параметр);
- `expires` — объект даты (`Date`), содержащий срок хранения cookie-записи; если отсутствует, то после завершения работы браузера cookie-запись удаляется;
- `path` — строка, содержащая путь cookie-записи; если параметр не указан, то используется путь вызванного документа;
- `domain` — строка, содержащая домен нужной cookie-записи; если параметр не указан, то используется домен вызванного документа;
- `secure` — логическое значение (`true` или `false`), определяющее необходимость использования безопасного HTTP-соединения.

Листинг 3.45. Функция записи данных cookie

```
function writeCookie(name, value, expires, path, domain, secure) {
   /* запись cookie */
  document.cookie =
    name + "=" + escape(value) +
    ((expires) ? "; expires=" + expires.toGMTString(): "") +
    ((path) ? "; path=" + path : "") +
    ((domain) ? "; domain=" + domain : "") +
    ((secure) ? "; secure" : "")
}
```

Обратите внимание, что в теле функции `writeCookie()` происходит кодирование значения параметра `value` с помощью встроенной функции `escape()`.

Для удаления cookie-записи можно вызывать функцию, представленную в листинге 3.46.

Листинг 3.46. Функция удаления данных cookie

```
function deleteCookie(name, path, domain) {
  /* удаление cookie-записи */
  if (readCookie(name)) {
    document.cookie =
      name + "=" +
      ((path) ? "; path=" + path : "") +
```

```
      ((domain) ? "; domain="+domain : "") +
      "; expires=Thu, 01-Jan-70 00:00:01 GMT"
  }
}
```

Эта функция устанавливает дату срока хранения cookie-записи так, что запись будет удалена. Параметр `path` должен иметь такое же значение, которое использовалось при создании cookie-записи, или пустое значение (`null`), если при создании записи он не был определен. Таким же образом задается значение параметра `domain`.

Советую вам поэкспериментировать с чтением, созданием и удалением cookies-записей (листинг 3.47). При этом следует иметь в виду, что созданные или измененные cookies-записи будут записаны на диск только после закрытия браузера. До этого они существуют лишь в кэше (оперативной памяти). С другой стороны, cookies-записи загружаются в оперативную память и становятся доступны как значение свойства `document.cookie` только при запуске браузера.

Листинг 3.47. HTML-код для эксперимента со сценарием, записывающим и читающим cookie-запись

```
<HTML>
  <SCRIPT>
    function readCookie(name) {  // чтение cookie-данных записи
      ... // код
    }

    function getCookieVal(n){    // вспомогательная функция, вызываемая
                                 // из readCookie()
      ... // код
    }

    function writeCookie(name, value, expires, path, domain, secure) {
    // запись cookie
      ... // код
    }

    / * Срок хранения — 1 год от текущей даты: */
    var d1=new Date()
    var d2=d1.getTime() + (365*24*60*60*1000)
    d1.setTime(d2)
```

```
    /* Запись и чтение cookie: */
    writeCookie("myrecord","Привет",d1)
    alert(readCookie("myrecord"))
  </SCRIPT>
</HTML>
```

Примечание

Работа с cookie может поддерживаться и серверными сценариями на языке PHP *(см. разд. 5.2.5)*.

3.9. Перемещение элементов мышью

Рассмотрим задачу перемещения видимых элементов HTML-документа с помощью мыши. Такими элементами могут быть графические объекты, кнопки, текстовые области, таблицы, плавающие фреймы и др. Обсудим перемещение изображений, текстовых областей и плавающих фреймов.

3.9.1. Перемещение графических объектов

Рассмотрим один из способов перемещения изображения:

1. Пользователь пытается перетащить мышью изображение.
2. Затем он должен отпустить кнопку мыши и переместить указатель в нужное место (при этом он может удерживать или не удерживать кнопку мыши в нажатом положении).
3. Остановившись в нужном месте, пользователь отпускает кнопку мыши или щелкает ею, чтобы прекратить перемещение изображения.

В листинге 3.48 приведен код, реализующий этот алгоритм.

Листинг 3.48. Перемещение изображения мышью

```
<HTML>
  <HEAD><TITLE>Перемещение изображения</TITLE></HEAD>
  <BODY ID=mybody BGCOLOR=white>
    <IMG ID="myimg" SRC='pict.gif' ondragstart="drag()"
         STYLE="position:absolute; top:10; left:10">
  </BODY>
  <SCRIPT>
    flag=false                          // нельзя перемещать
    var id_img=""
```

```
    function drag(){
      flag=!flag
      id_img=event.srcElement.id
    }

    function mybody.onmousemove(){
      if (flag){                        // если можно перемещать
        document.all[id_img].style.top=event.clientY
        document.all[id_img].style.left=event.clientX
      }
    }

    function mybody.onmouseup(){
      flag=false                        // нельзя перемещать
    }
  </SCRIPT>
</HTML>
```

Здесь функция `drag()`, обрабатывающая событие `ondragstart` (попытка перетаскивания), устанавливает переменную-триггер `flag` и выясняет, кто инициатор события, записывая значение его `id` в переменную `id_img`. Значение переменной `flag` позволяет определить, можно или нельзя перемещать элемент. В данном примере инициатором события может быть только один элемент, но мы готовимся к более общему случаю нескольких перемещаемых элементов. Обратите внимание, что события `onmousemove` (перемещение указателя мыши) и `onmouseup` (кнопка мыши отпущена) получает не изображение, а объект тела документа `mybody`. Мы не использовали событие `onclick` для изображения, зарезервировав его для других целей (например, для перехода по ссылке, если нам это потребуется). Заметьте также, что в функции `mybody.onmousemove()` при обращении к объекту изображения его идентификатор передается через переменную `id_img`. Поэтому мы используем запись `document.all[id_img]`, а не `document.all.id_img`.

3.9.2. Перемещение текстовых областей

В листинге 3.49 приведен пример HTML-документа, в котором можно перемещать текстовые области, созданные с помощью тегов `<TEXTAREA>`. При этом размеры области и шрифта текста определяются в ней в зависимости от значения вертикальной координаты. Так создается *эффект перспективы* (ближе-дальше). Чем выше, тем дальше, и наоборот (рис. 3.10).

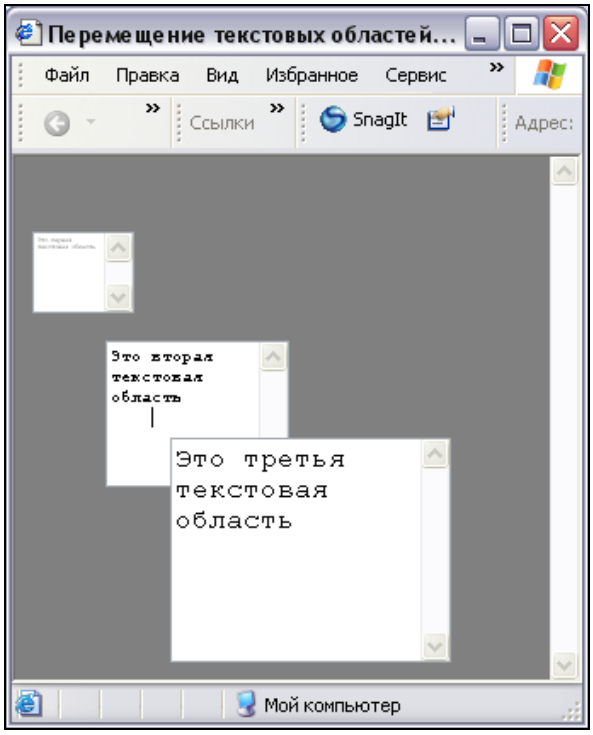


Рис. 3.10. Перемещаемые текстовые области

Чтобы приблизить к себе текстовую область, необходимо просто переместить ее вниз. Вы можете "сложить в стопку" текстовые области, отодвинуть их от себя или, наоборот, приблизить так, чтобы текст стал разборчивым. В отличие от примера с изображениями, здесь разрешение перемещения происходит по двойному щелчку на текстовой области. Обратите внимание на то, как в функции `resizetext()` определяется элемент, размеры которого следует изменить. В принципе, ничто не мешает вам создать более сложный алгоритм для функции `resizetext()`. Например, можно дополнительно варьировать цвет фона и шрифта.

Листинг 3.49. Перемещение текстовых областей

```
<HTML>
  <HEAD><TITLE>Перемещение текстовых областей</TITLE></HEAD>
  <BODY ID="mybody" BGCOLOR=#808080>
    <TEXTAREA ID="t1" ondblclick="drag()"
              STYLE="position:absolute; top:50; left:10;fon-size:large">
```

```
Это первая текстовая область
    </TEXTAREA>
    <TEXTAREA ID="t2" ondblclick="drag()"
              STYLE="position:absolute; top:100; left:150">
Это вторая текстовая область
    </TEXTAREA>
    <TEXTAREA ID="t3" ondblclick="drag()"
              STYLE="position:absolute; top:150; left:250">
Это третья текстовая область
    </TEXTAREA>
  </BODY>
  <SCRIPT>
    resizetext()      // установка размеров текстовых областей
    var flag=false
    var id_img=""

    function drag(){
      flag=!flag
      id_img=event.srcElement.id    /* id элемента, который надо
                                       перемещать */
    }
    function mybody.onmousemove(){
      if (flag){
        document.all[id_img].style.top=event.clientY
        document.all[id_img].style.left=event.clientX
        resizetext()   // установка размеров текстовых областей
      }
    }

    function mybody.onmouseup(){
      flag=false
    }

    function resizetext(){  /* установка размеров текстовых
                               областей */
      var y, size, idimg, idtext
      for (i=0; i < document.all.length; i++){
        if (document.all[i].tagName == 'TEXTAREA') {
```

```
          idtext=document.all[i].id
          y=parseInt(document.all[idtext].style.top)
          size=Math.min(y, 800)
          size=Math.max(size, 60)
          document.all[idtext].style.width=size
          document.all[idtext].style.height=0.8*size
          document.all[idtext].style.zIndex=y
          document.all[idtext].style.fontSize=Math.max(2, y/10)
        }
      }
    }
  </SCRIPT>
</HTML>
```

3.9.3. Перемещение плавающих фреймов

Небольшая модификация кода, приведенного в листинге 3.50, позволяет получить документ с множеством перемещаемых плавающих фреймов, в которые загружены другие HTML-документы (рис. 3.11). Однако в этом случае возникает небольшая трудность: как передать плавающему фрейму сигнал, что мы собираемся его перемещать? Дело в том, что сигналы о нажатии кнопки мыши, щелчках и попытках перетаскивания при наведении указателя мыши на фрейм получает не он, а элементы загруженного в него документа. Я выбрал следующий способ решения данной задачи:

1. Рядом с каждым фреймом поместим маленькое изображение, которое будет воспринимать событие, интерпретируемое как разрешение на перемещение фрейма. Я использовал изображение руки, расположенное около верхнего левого угла фрейма.
2. В функции изменения положения напишем код, который меняет координаты, размеры и индекс фрейма, а также графического объекта. Таким образом, изображение будет получать событие, а обработчик этого события будет изменять параметры и изображения, и фрейма.

Напомню, что плавающие фреймы создаются с помощью контейнерного тега `<IFRAME>`. В сценарии (листинг 3.50) мы сначала сформируем массив с адресами документов (файлов), загружаемых во фреймы. Затем сформируем строку тегов, описывающих фреймы и графические изображения, и запишем ее в HTML-документ с помощью метода `write()`. Функция, производящая вычисление положения и размеров фреймов и изображения, в этом примере называется `resizeframe()`.

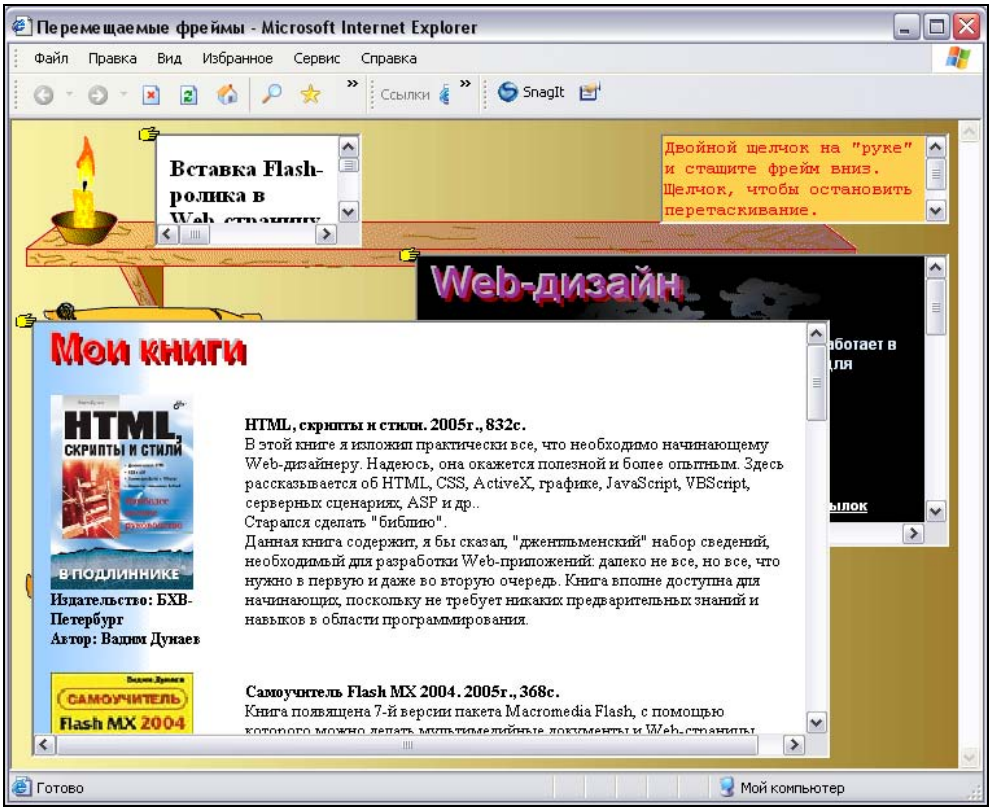


Рис. 3.11. В перемещаемые фреймы загружены страницы сайта

Листинг 3.50. Перемещение плавающих фреймов

```
<HTML>
  <HEAD><TITLE>Перемещаемые фреймы</TITLE></HEAD>
  <BODY ID="mybody" BACKGROUND="myfon.gif">
  </BODY>
  <SCRIPT>
    /* Массив адресов документов */
    var ahref=new Array()
    ahref[0]="flashx.htm"
    ahref[1]="examples.htm"
    ahref[2]="mybook.htm"
    /* Формирование строки с описанием фреймов и изображений */
```

```
var xstr=""
for (i=0; i<ahref.length;i++){
  /* Фреймы */
  xstr+= '<IFRAME width=200  ID="fr' + i + '" SRC="'+ahref[i] +
    '"  STYLE="position:absolute;  top:' + (5 + 25*i)+'; left:' +
    (10 + 30*i) + '"></iframe>'
  /* Изображения */
  xstr+= '<IMG ondblclick="drag()" width=16 height=14 SRC="рука.gif"
          id="im' + i + '" STYLE="position:absolute">'
}
document.write(xstr)                    // запись в документ

flag=false
var id_img=""
resizeframe()                           // установка размеров фреймов

function drag(){
  flag=!flag
  id_img=event.srcElement.id
  id_img="fr" + id_img.substr(2)
}

function mybody.onmousemove(){
  if (flag){
    document.all[id_img].style.top=event.clientY
    document.all[id_img].style.left=event.clientX
    resizeframe()
  }
}

function mybody.onmouseup(){
  flag=false
}

function resizeframe(){                 // установка размеров фреймов
  var y, size, idimg, idfr
  for (i=0; i < ahref.length; i++){
```

```
        /* Фреймы */
        idfr="fr" + i                        // id фрейма
        y=parseInt(document.all[idfr].style.top)
        size=Math.min(y,800)
        size=Math.max(60,size)
        size=size*3
        document.all[idfr].style.width=size
        document.all[idfr].style.height=0.8*size
        document.all[idfr].style.zIndex=y
        /* Изображения */
        idimg="im" + i                       // id изображения
        document.all[idimg].style.top=y-5
        document.all[idimg].style.left =
          parseInt(document.all[idfr].style.left) - 12
        document.all[idimg].style.zIndex=document.all[idfr].style.zIndex
      }
    }
  </SCRIPT>
</HTML>
```

Обратите внимание, что перемещение фрейма приостанавливается, когда указатель мыши находится над другим фреймом. Поэтому для перемещения фрейма указатель мыши должен двигаться над участками окна, не занятыми фреймами. Это небольшое неудобство вызвано тем, что фрейм маскирует тело (элемент `<BODY>`) документа исходного окна и обработчик `mybody.onmousemove()` не срабатывает. Маскирующий эффект возникает из-за того, что фрейм является разновидностью окна. В случае с текстовыми областями этот эффект не наблюдается, поскольку текстовая область — просто элемент документа текущего окна, а не отдельное окно.

3.10. Меню

Меню, как элемент навигации и управления, часто встречаются на Web-страницах. Они могут иметь различные внешний вид и внутреннюю конструкцию, но общим для них является необходимость обработки в сценарии выбора опции, произведенного пользователем. Мы рассмотрим несколько возможных способов создания меню.

3.10.1. Раскрывающийся список

Простейшее меню можно создать с помощью тегов `<SELECT>` и `<OPTION>` *(см. разд. 1.2.7)*. Обычно такие конструкции называют *раскрывающимися списками*. В листинге 3.51 приведен простейший пример использования раскрывающегося списка. В этом примере раскрывающийся список задается HTML-кодом, а обработка выбора из списка — сценарием. Задача сценария заключается просто в обработке номера выбранного элемента. В примере это соответствует выводу окна с сообщением о номере выбранного элемента. Выбор пользователя из раскрывающегося списка производится щелчком левой кнопкой мыши на элементе списка. При этом свойство `selectedIndex` объекта элемента документа, соответствующего тегу `<SELECT>`, приобретает в качестве своего значения номер выбранного элемента списка (нумерация начинается с 0). Для инициации обработки выбора пользователя здесь служит событие `onchange` (произошло изменение в выделении элемента списка). Обработка этого события осуществляется пользовательской функцией `myselection()`. Начальное выделение и отображение элемента в раскрывающемся списке задается атрибутом `SELECTED` тега `<OPTION>`. В рассматриваемом примере в списке виден элемент "Вторая опция".

Листинг 3.51. Выбор элемента из раскрывающегося списка

```
<HTML>
  Выберите что-нибудь:
  <SELECT NAME="TEST" onchange ="myselection()">
    <OPTION>Первая опция
    <OPTION SELECTED>Вторая опция
    <OPTION>Третья опция
    <OPTION>Четвертая опция
  </SELECT>
  <SCRIPT>
    function myselection() {
      alert("Вы выбрали опцию №: " + document.all.TEST.selectedIndex)
    }
  </SCRIPT>
</HTML>
```

3.10.2. Одноуровневое меню

Одноуровневое меню нетрудно сделать на основе таблицы. Оно может быть вертикальным или горизонтальным. В ячейках таблицы располагаются названия опций (элементов) меню. При наведении указателя мыши на элемент последний выделяется (подсвечивается) цветом, а при щелчке на элементе выполняется некоторое действие, определенное сценарием.

В листинге 3.52 приведен HTML-код, задающий вертикальное меню из трех элементов, и набор функций, которые обрабатывают события `onmouseover` (наведение), `onmouseout` (уход) и `onclick` (щелчок).

Листинг 3.52. Одноуровневое меню

```
<HTML>
<BODY>
  <! Здесь HTML-код вашей страницы >
  <TABLE ID="menu" border=1 bgcolor='#f0f0f0'
                   width=40 rules=none cellpadding=10 cellspacing=1>
  <TR><TD id="e1"
    onmouseover="onelement()" onmouseout="outelement()">Один
  </TD></TR>
  <TR><TD id="e2"
    onmouseover="onelement()" onmouseout="outelement()">Два
  </TD></TR>
  <TR><TD id="e3"
    onmouseover="onelement()" onmouseout="outelement()">Три
  </TD></TR>
  </TABLE>
</BODY>
<SCRIPT>
function onelement(){        // наведение на элемент
  xid=event.srcElement.id
  document.all[xid].style.color='white'
  document.all[xid].style.background='blue'
}
function outelement(){       // уход с элемента
  xid=event.srcElement.id
  document.all[xid].style.color='black'
  document.all[xid].style.background='#f0f0f0'
}
```

```
function document.all.menu.onclick(){ // щелчок на элементе
  alert(event.srcElement.id) /* вывод сообщения об id элемента,
                                на котором был щелчок */
}
</SCRIPT>
</HTML>
```

Обратите внимание, что метод (функция) `document.all.menu.onclick()`, а также другие функции сами определяют, на каком элементе меню был щелчок: свойство `event.srcElement.id` возвращает идентификатор этого элемента. Использование этого свойства освобождает нас от рутинного повторения кода для каждого элемента.

3.10.3. Простое двухуровневое меню

Двухуровневое меню, которое мы сейчас рассмотрим, состоит из главного горизонтального меню и нескольких вертикальных подменю. При наведении указателя мыши на элемент главного меню раскрывается соответствующее подменю, если, конечно, оно определено. При уходе указателя с подменю последнее скрывается. Кроме того, элемент меню выделяется цветом (подсвечивается) при наведении на него указателя мыши. Щелчок на элементе меню обрабатывается специальной функцией. Такое меню можно сделать на основе нескольких вертикальных меню, созданных с использованием таблиц и рассмотренных в предыдущем разделе.

В рассматриваемом здесь примере главное горизонтальное меню состоит из трех элементов, двум из которых соответствуют вертикальные раскрывающиеся подменю (листинг 3.53).

Листинг 3.53. Простое двухуровневое меню

```
<HTML>
<BODY>
<! Таблицы, из которых строится меню >
<TABLE id=menu1 onmouseover="document.all.tb1.style.display='" style="Z-
INDEX: 5; BACKGROUND: #e0e0e0; position: absolute; TOP: 60px; LEFT:
250px; WIDTH: 80px " onclick=menuclick() onmou-
seout="document.all.tb1.style.display='none'" cellSpacing=0 cellPad-
ding=10 rules=none border=1>
<TBODY>
<TR>
<TD id=menuzag1 onmouseover=onmousemenu() style="BACKGROUND: #00ff00"
onmouseout=outmousemenu()>Меню 1</TD></TR>
```

```
<TBODY id=tb1 style=" display: none">
<TR>
<TD id=e11 onmouseover=onelement() onmouseout=outelement()>Один</TD></TR>
<TR>
<TD id=e12 onmouseover=onelement() onmouseout=outelement()>Два </TD></TR>
<TR>
<TD id=e13 onmouseover=onelement() onmouseout=outelement()>Три</TD></TR>
</TBODY></TABLE>
<TABLE id=menu2 onmouseover="document.all.tb2.style.display='" style="Z-
INDEX: 5; BACKGROUND: #e0e0e0; POSITION: absolute; TOP: 60px; LEFT:
330px; WIDTH: 80px;" onclick=menuclick() onmou-
seout="document.all.tb2.style.display='none'" cellSpacing=0 cellPad-
ding=10 rules=none width=40 border=1>
<TBODY>
<TR>
<TD id=menuzag2 onmouseover=onmousemenu() style="BACKGROUND: #00ff00"
onmouseout=outmousemenu()>Меню 2</TD></TR>
<TBODY id=tb2 style=" display: none">
<TR>
<TD id=e21 onmouseover=onelement() onmou-
seout=outelement()>Первый</TD></TR>
<TR>
<TD id=e22 onmouseover=onelement() onmou-
seout=outelement()>Второй</TD></TR>
</TBODY></TABLE>
<TABLE id=menu3 onmouseover="document.all.tb2.style.display='" style="Z-
INDEX: 5; BACKGROUND: #e0e0e0; POSITION: absolute; TOP: 60px; LEFT:
410px; WIDTH: 80px " onclick=menuclick() onmou-
seout="document.all.tb2.style.display='none'" cellSpacing=0 cellPad-
ding=10 rules=none width=40 border=1>
<TBODY>
<TR>
<TD id=menuzag3 onmouseover=onmousemenu() style="BACKGROUND: #00ff00"
onmouseout=outmousemenu()>Меню 3</TD></TR>
</TBODY></TABLE>
<SCRIPT>
function onelement(){ // наезд на элемент
xid=event.srcElement.id
document.all[xid].style.color='white'
document.all[xid].style.background='#0000ff'
}
```

```
function outelement(){ // уход с элемента
xid=event.srcElement.id
document.all[xid].style.color='black'
document.all[xid].style.background='#e0e0e0'
}
function onmousemenu(){ // наезд на заголовок меню
xid=event.srcElement.id
document.all[xid].style.background='#00dd00'
}
function outmousemenu(){
xid=event.srcElement.id
document.all[xid].style.background='#00ff00'
}

function menuclick(){ // щелчок на элементе меню
alert("Элемент меню: "+event.srcElement.id) // вы можете написать здесь
свой код
/* Пример выбора первого элемента
if (event.srcElement.id=="e11"){window.open("http://www.yandex.ru")}
*/
}
</SCRIPT>
<BODY>
</HTML>
```

Обратите внимание, что значение стилевого параметра `z-index` для таблицы выбрано заведомо большим, чтобы гарантировать ее появление поверх всех других элементов документа.

3.10.4. Сложное двухуровневое меню

В данном разделе мы рассмотрим двухуровневое меню (рис. 3.12), которое полностью генерируется сценарием. Внешне оно выглядит так же, как и рассмотренное в предыдущем разделе, но имеет больше возможностей настройки параметров.

Код сценария, реализующего такое меню, мы сделаем, по возможности, более универсальным и оформим в виде двух файлов с расширением js.

- Файл menu_prm.js хранит параметры меню (листинг 3.54). При создании приложений его содержимое можно изменять в зависимости от конкрет-

ных задач. Например, в этом файле определяются названия опций и действия для них, цвет и другие параметры. Таким образом, это — переменная (настраиваемая) часть описания меню.

- Файл menu_bld.js содержит описание механизма построения и функционирования меню (листинг 3.55). В нем используются параметры, определенные в первом файле. Это постоянная часть описания меню. Разумеется, вы можете скорректировать содержимое данного файла по своему усмотрению.

Чтобы создать меню, в HTML-документе следует просто записать следующие строки:

```
<SCRIPT SRC="menu_prm.js"></SCRIPT>
<SCRIPT SRC="menu_bld.js"></SCRIPT>
<SCRIPT >buildMenu()</SCRIPT>
```

Здесь `buildMenu()` — функция, определение которой находится в файле menu_bld.js. Она выводит на экран меню в соответствии с параметрами, заданными в файле menu_prm.js.

Рассмотрим пример, в котором главное (горизонтальное) меню содержит три опции, первым двум из которых соответствуют вертикальные подменю. Последняя опция является терминальной. Названия опций выбраны так, чтобы было легко понять, что к чему относится (рис. 3.12).

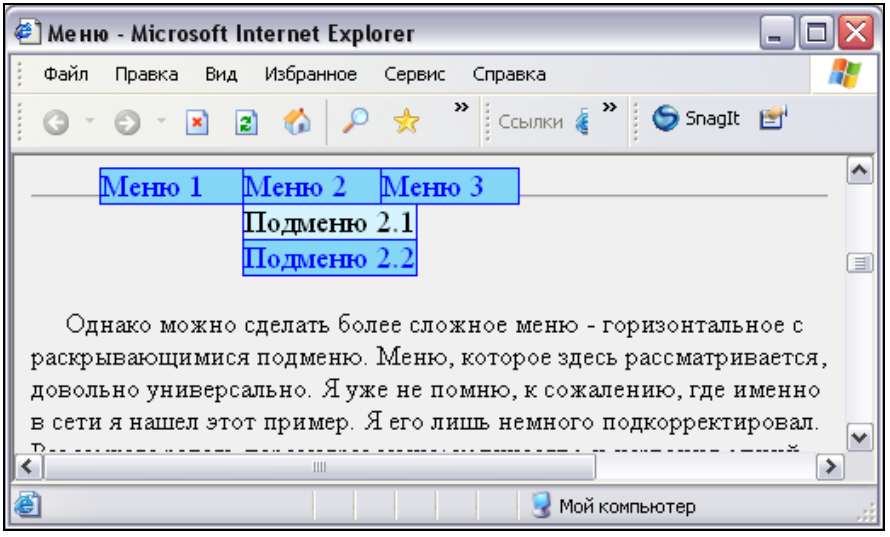


Рис. 3.12. Внешний вид меню

В первом файле мы задаем параметры цвета, шрифта, а также состав названий опций, действия и координаты. Названия опций меню, их позиционирование, действия и содержание статусной строки задаются с помощью массивов. В конце концов, структура меню определяется двумерным массивом. Обратите внимание, что если действие представляет собой не URL-адрес, а некоторый код на языке JavaScript, то он должен начинаться с префикса `"javascript:"`, за которым следуют выражения, разделенные точкой с запятой.

Во втором файле определен ряд функций, с помощью которых меню разворачивается и работает. Отображение меню происходит на основе использования HTML-тегов, определяющих таблицы. Фрагменты HTML-кода, из которых складываются генерируемые строки HTML-документа, определены в виде элементов массива. Позиционирование меню производится с помощью параметров `top` и `left` таблицы стилей, которая также генерируется сценарием. Среди множества функций, определенных в этом файле, имеется главная — `buildMenu()`. Вызов ее в сценарии, расположенном в HTML-документе, выводит меню в окно браузера вместе с другими элементами этого документа.

Листинг 3.54. Код файла menu_prm.js с определением параметров меню

```
/* Параметры меню */
// Цвета:
var clBorder='blue'          // цвет рамки
var clBgInact="#83d6f5"      // цвет фона
var clBgAct='#d6f4fe'        // цвет подсветки фона опции
var clFnInact='blue'         // цвет надписи обычной
var clFnAct='black'          // цвет надписи при подсветке опции

// Шрифт (параметры стиля):
var cFontSize='18'           // размер шрифта
var cFontFamily='times'      /* семейство шрифтов ('arial',
                                'courire', 'times' и т. п.) */

// Другие параметры:
var closeTimeout=500         /* время задержки свертывания
                                подменю, мс */
var selfPos=false            /* автоматическое позиционирование опций
меню и подменю (если true, то задаются координаты только первого меню) */
```

```
// Собственно меню:
var menu =new Array()
menu[0] =new Array()
/* Параметры главного меню:
     название опции;
     URL-адрес или строка с кодом JavaScript,
       начинающаяся с "javascript:";
     статусная строка;
     left (x-координата главного меню);
     top (y-координата главного меню);
     width (ширина главного меню);
     left (x-координата подменю)
     top (y-координата подменю);
     width (ширина подменю).
*/
menu[0][0]=new Array("Меню 1","","Выберите что-нибудь из подменю",
                     50, 10, 80, 50,1 20, 120)
menu[0][1]=new Array("Подменю 1.1", "javascript:history.go(-1)", "Назад")
menu[0][2]=new Array("Подменю 1.2", "http://www.yandex.ru", "Яндекс")
menu[0][3]=new Array("Подменю 1.3", "javascript: alert('Привет!')",
                     "Окно с приветствием")
menu[1] =new Array()
menu[1][0]=new Array("Меню 2", "", "", 130, 10, 80, 130, 120, 0)
menu[1][1]=new Array("Подменю 2.1", "", "")
menu[1][2]=new Array("Подменю 2.2", "", "")
menu[2] =new Array()
menu[2][0]=new Array("Меню 3", "http://www.chat.ru",
                     "CHAT.RU", 210, 10, 80, 0, 0, 0)
```

Листинг 3.55. Код файла menu_bld.js

```
var ie=document.all ? true : false /* Есть ли что-нибудь в документе? */

var overBox=''
var timerID

// Заготовки элементов HTML-кода
var barHtml=new Array()
```

```
barHtml[0]='<DIV ID="divbarpos'
barHtml[1]='" STYLE="position:absolute;left:'
barHtml[2]='px;top:'
barHtml[3]='px;" onmouseover="openbox('
barHtml[4]=')" onmouseout="closebox('
barHtml[5]=')" onclick="clickbox('
barHtml[6]=')"><TABLE CELLPADDING=0 CELLSPACING=0><TR><TD BGCOLOR="'
barHtml[7]='"><TABLE CELLPADDING="0" CELLSPACING="1" BORDER="0"> <TR><TD
ID="mnubarpos'
barHtml[8]='" WIDTH="'
barHtml[9]='" BGCOLOR="'
barHtml[10]='" STYLE="color:'
barHtml[11]=';font-size:'+cFontSize+';font-family:'+cFontFamily+';">'
barHtml[12]='</TD></TR></TABLE></TD></TR></TABLE></DIV>'
var boxHtml=new Array()
boxHtml[0]='<DIV ID="divbox'
boxHtml[1]='" STYLE="position:absolute;visibility:hidden;left:'
boxHtml[2]='px;top:'
boxHtml[3]='px;font-family:'+cFontFamily+'" onmouseout="closebox('
boxHtml[4]=')"><TABLE CELLPADDING=0 CELLSPACING=0><TR><TD BGCOLOR="'
boxHtml[5]='"><TABLE ID="mnubox'
boxHtml[6]='" CELLPADDING="0" CELLSPACING="1" BORDER="0" >'
boxHtml[7]='<SPAN ID="divboxpos'
boxHtml[8]='" onmouseover="openboxpos('
boxHtml[9]=')" onmouseout="closeboxpos('
boxHtml[10]=')" onclick="clickboxpos('
boxHtml[11]=')"><TR><TD ID="mnuboxpos'
boxHtml[12]='" WIDTH="'
boxHtml[13]='" BGCOLOR="'
boxHtml[14]='" STYLE="color:'
boxHtml[15]=';font-size:'+cFontSize+'">'
boxHtml[16]='</TD></TR></SPAN>'
boxHtml[17]='</TABLE></TD></TR></TABLE></DIV>'

function buildMenu() {       // построение меню (главная функция)
  if (ie) {
     buildMenuBar();
     buildSubMenu();
```

```
     if (selfPos) PosMenu();
       ResizeSubMenu();
  }
}

function buildMenuBar() {    // построение горизонтального меню
  for (i=0; i < menu.length; i++){
    var s=barHtml[0] + i + barHtml[1] + menu[i][0][3] + barHtml[2] +
        menu[i][0][4] + barHtml[3] + i + barHtml[4] + i + barHtml[5] +
        i + barHtml[6] + clBorder + barHtml[7] + i + barHtml[8] +
        menu[i][0][5] + barHtml[9] + clBgInact + barHtml[10] +
        clFnInact + barHtml[11] + menu[i][0][0] + barHtml[12];
    document.writeln(s);
  }
}

function buildSubMenu() {                          // построение подменю
  for (i=0; i < menu.length; i++){
    if (menu[i].length > 1) {
      var s=boxHtml[0] + i + boxHtml[1] + menu[i][0][6] +
            boxHtml[2] + menu[i][0][7] + boxHtml[3] + i +
            boxHtml[4] + clBorder + boxHtml[5] + i + boxHtml[6]
      for (j=1; j < menu[i].length; j++) {
        var s1=i + ',' + j
        var s2=i + '_' + j
        s +=  boxHtml[7] + s2 + boxHtml[8] + s1 + boxHtml[9] + s1 +
              boxHtml[10] + s1 + boxHtml[11] + s2 + boxHtml[12] +
              menu[i][0][8] + boxHtml[13] + clBgInact + boxHtml[14] +
              clFnInact + boxHtml[15] + menu[i][j][0] + boxHtml[16]
      }
      s += boxHtml[17]
      document.writeln(s)
    }
  }
}

function PosMenu() {
  for (i=0; i < menu.length; i++) {
    barCurr=document.all['divbarpos' + i]
```

```
    if (i > 0) {
      barPrev=document.all['divbarpos' + (i - 1)]
      barCurr.style.left=barPrev.offsetLeft + barPrev.offsetWidth - 1
    }
    if (menu[i].length > 1) {
      boxCurr=document.all['divbox' + i]
      boxCurr.style.pixelTop=barCurr.offsetTop +
                              barCurr.offsetHeight - 1
      boxCurr.style.pixelLeft=barCurr.offsetLeft
    }
  }
}

function ResizeSubMenu() {
  for (i=0; i < menu.length; i++) {
    if (menu[i].length > 1 && menu[i][0][8] <= 0) {
      el=document.all['mnubox' + i]
      w=document.all['divbarpos' + i].offsetWidth
      if (el.offsetWidth < w) el.style.width=w
    }
  }
}

function openbox(x) {
  paintLayer('mnubarpos' + x, active=true)
  showLayer('divbox' + x)
  for (i=0; i < menu.length; i++) if (i != x) closebox(i, 0)
  overBox='divbox' + x
  window.status=menu[x][0][2]
}

function closebox(x, timeout) {
  paintLayer('mnubarpos' + x, active=false)
  clearTimeout(timerID)
  if (timeout == 0) hideLayer('divbox' + x)
  else timerID=setTimeout('hideLayer("divbox' + x + '")', closeTimeout)
  overBox=''
  window.status=defaultStatus
}
```

```
function clickbox(x) {
  clickMenu(menu[x][0][1])
}

function openboxpos(x, y) {
  paintLayer('mnuboxpos' + x + '_' + y, active=true)
  overBox='divbox' + x
  window.status=menu[x][y][2]
}

function closeboxpos(x, y) {
  window.status=defaultStatus
  paintLayer('mnuboxpos' + x + '_' + y, active=false)
}

function clickboxpos(x, y) {
  clickMenu(menu[x][y][1])
}

function clickMenu(s) {
  if (s.indexOf('javascript:') == 0) eval(s);
  else if (s != '') window.location.href=s;
}

function paintLayer(layerID, active) {
  if (layer=document.all[layerID]) {
    if (active) { clBg=clBgAct; clFn=clFnAct }
    else { clBg=clBgInact; clFn=clFnInact }
    layer.style.backgroundColor=clBg
    layer.style.color=clFn
  }
}

function showLayer(layerID) {
  if (layer=document.all[layerID]) layer.style.visibility='visible'
}
```

```
function hideLayer(layerID) {
  if (document.all[layerID] && (overBox != layerID))
  document.all[layerID].style.visibility='hidden'
}
```

Примечание

Позиционирование элементов меню и подменю может происходить независимо друг от друга. Это позволяет визуально представить меню на экране в виде пространственно разнесенных частей. Таким образом, чтобы сделать несколько меню, не обязательно для каждого из них создавать и загружать специальные файлы с описаниями.

3.10.5. Древовидный список

Древовидный раскрывающийся список можно сделать на основе элемента управления ActiveX `TreeView`. Его можно использовать, например, для представления оглавлений документов (книг), структуры папок с файлами и т. п.

Элемент управления `TreeView` вставляется в HTML-код с помощью тега

```
<OBJECT ID='myTreeView'
       CLASSID="clsid:C74190B6-8589-11D1-B16A-00C0F0283628">
</OBJECT>
```

Если содержимое списка не помещается в область, заданную атрибутами `WIDTH` и `HEIGHT`, то автоматически появляются полосы прокрутки. В тег `<OBJECT>` для элемента `TreeView` можно вставить теги `<PARAM>`, определяющие дополнительные параметры. Их список можно узнать с помощью выражения `alert(myTreeView.innerHTML)`. На внешний вид списка влияют следующие параметры:

```
<PARAM NAME='LineStyle'   VALUE='число'>
<PARAM NAME='Style'       VALUE='число'>
<PARAM NAME='BorderStyle' VALUE='число'>
```

Каждый узел иерархического списка представляется объектом `Node`. Множество всех узлов списка хранится в коллекции `Nodes`. Каждый узел может быть корневым (не иметь родителя), конечным (не иметь потомков) или промежуточным (иметь родителя и потомков). Корневых узлов может быть несколько, и тогда список состоит из нескольких древовидных списков.

Объект `Node` (узел) имеет ряд свойств, с помощью которых описывается местоположение узла в списке: `parent` (узел-родитель), `child` (первый потомок), `children` (количество потомков) и `root` (корень).

Список создается путем добавления узлов в коллекцию `Nodes` посредством метода `Add`, который имеет следующие параметры.

- Первый параметр определяет объект, относительно которого создается новый узел. Это либо индекс, либо уникальный идентификатор существующего объекта. Если данный параметр не указан, то создается корневой узел.
- Второй параметр определяет отношение к уже существующим узлам. Возможны следующие значения:
  - 0 — новый узел создается перед всеми узлами того же уровня, что и узел, определенный первым параметром;
  - 1 — новый узел создается после всех узлов того же уровня, что и узел, определенный первым параметром;
  - 2 — новый узел создается после узла, определенного первым параметром;
  - 3 — новый узел создается перед узлом, определенным первым параметром;
  - 4 — новый узел создается как потомок узла, определенного первым параметром.
- Третий параметр — идентификатор — строка символов; по идентификатору можно ссылаться на создаваемый узел.
- Четвертый параметр — текст, отображаемый рядом с меткой узла (обязательный параметр).
- Пятый параметр определяет изображение, соответствующее узлу.
- Шестой параметр определяет изображение, соответствующее узлу, когда он выбран щелчком или с помощью навигационных клавиш.

Только параметр, определяющий подписи узлов, является обязательным. Вместо пропускаемых параметров в вызове метода `Add` следует ставить `null`.

В листинге 3.56 приведен код HTML-документа, содержащего элемент управления `TreeView`, а также плавающий фрейм (тег `<IFRAME>`). Вид этого документа в окне браузера показан на рис. 3.13. В сценарии формируется древовидный список и определены процедуры, обрабатывающие события, связанные со списком. В результате их выполнения список можно развернуть, свернуть и загрузить в плавающий фрейм некоторый HTML-документ. В этом примере загрузка будет происходить при щелчке на концевом узле. При этом адрес загружаемого документа формируется из подписи концевого узла путем приписывания к ней точки и расширения файла htm. Разумеется,

возможны и другие алгоритмы формирования URL-адреса отображаемого документа.

В обработчике щелчка на элементах списка используется подобъект `SelectedItem` объекта `TreeView`, который содержит сведения о выделенном узле. Так, свойство `Text` содержит подпись выделенного узла, а `children` — количество его дочерних узлов.

Листинг 3.56. Древовидный список на основе ActiveX-элемента управления `TreeView`

```
<HTML>
  <HEAD><TITLE>Древовидный список</TITLE></HEAD>
  <BODY BGCOLOR=#e0e0e0>
    <TABLE WiDTH=100% HEIGHT=100%>
      <TR><TD WiDTH=200 HEIGHT=100%>
      <DIV ID=mydiv><FONT SIZE=4>
        <I ID=xxx></I>
        <OBJECT ID=myTreeView
                classid="clsid:C74190B6-8589-11D1-B16A-00C0F0283628"
                WIDTH="200" HEIGHT=100%">
          <PARAM NAME="Style" VALUE=7>
          <PARAM NAME="LineStyle" VALUE=1>
          <PARAM NAME="BorderStyle" VALUE=7>
        </OBJECT>
      </FONT>
      </DIV>
      </TD><TD WiDTH="*">
      <IFRAME id="myFrame" WIDTH=100% HEIGHT=100%></IFRAME>
      </TD></TR>
    </TABLE>
  </BODY>

  <SCRIPT>
    // Создание узлов
    myTreeView.Nodes.Add(null,null,"Ch0","Введение")
    myTreeView.Nodes.Add(null,null,"Ch1","Глава1")
    myTreeView.Nodes.Add("Ch1",4,"Ch11","Раздел 1.1")
    myTreeView.Nodes.Add("Ch11",4,"Ch111","1.1.1")
```

```
    myTreeView.Nodes.Add("Ch11",4,"Ch112","1.1.2")
    myTreeView.Nodes.Add("Ch1",4,"Ch12","Раздел 1.2")
    myTreeView.Nodes.Add("Ch12",4,"Ch121","1.2.1")
    myTreeView.Nodes.Add("Ch12",4,"Ch122","1.2.2")
    myTreeView.Nodes.Add(null,null,"Ch2","Глава2")
    myTreeView.Nodes.Add("Ch2",4,"Ch21","2.1")

    document.all.xxx.innerHTML=document.all.xxx.innerHTML //"Передернули"

   function mydiv.onclick() {                    // Щелчок
      if (!myTreeView.SelectedItem) return      // Если нет объекта
      if (myTreeView.SelectedItem.Children == 0)
   myFrame.location.href=myTreeView.SelectedItem.Text + ".htm"
    }
  </SCRIPT>
</HTML>
```

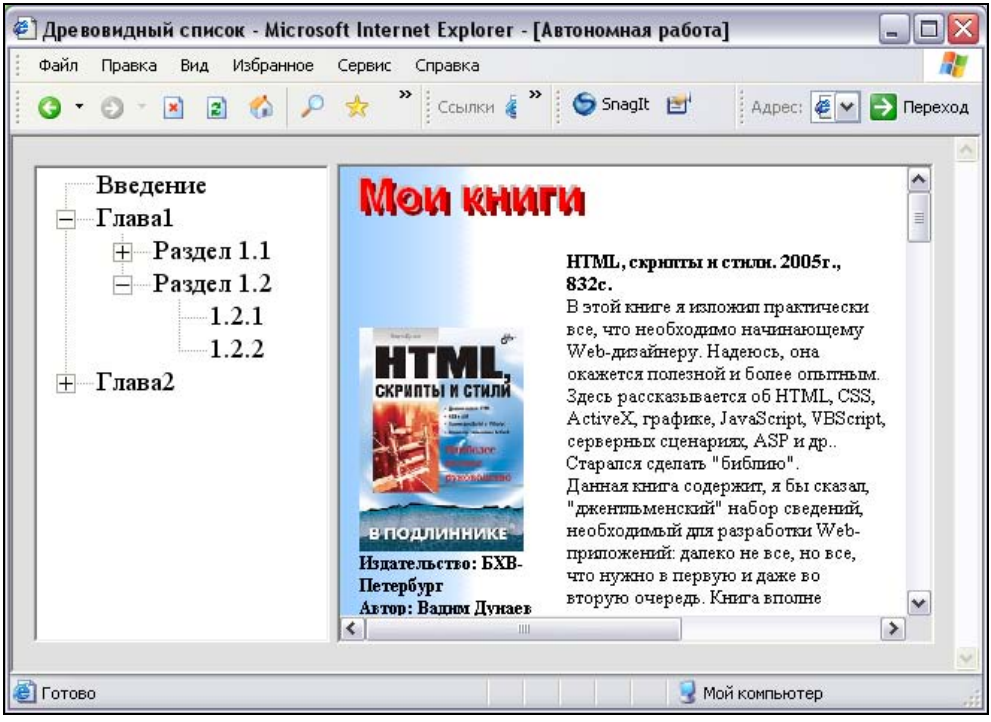


Рис. 3.13. Древовидный список

Примечание

Возможности элемента управления `TreeView` можно реализовать в полной мере, если сценарий написать на языке VBScript. Так, вы можете передать графические изображения для меток узлов с помощью функции `LoadPicture()` в дополнительный объект `ListImages`. Кроме того, сценарию на VBScript доступны события `dblclick` (двойной щелчок), `expand` (раскрытие узла) и `collapse` (свертывание узла). В сценарии на JavaScript эти возможности, к сожалению, не доступны. Более того, мне пришлось прибегнуть к некоторым ухищрениям, чтобы `TreeView` воспринимал одинарный щелчок. Для этого пришлось вставить ничего не содержащий тег `<I ID=xxx></I>`, а в сценарии написать строку

```
document.all.xxx.innerHTML=document.all.xxx.innerHTML
```

Это позволило инициализировать объект `SelectedItem` сразу же после загрузки `TreeView` и до щелчка пользователя на нем.

3.11. Таблицы

3.11.1. Доступ к элементам таблицы

Таблица, как объект документа, имеет две коллекции, посредством которых осуществляется доступ к ее содержимому. Первая из них — коллекция строк `rows`, а вторая — коллекция ячеек `cells`. Коллекция `rows` содержит все строки таблицы, включая разделы, соответствующие тегам `<THEAD>` и `<TFOOT>`. Коллекция `cells` содержит все элементы таблицы, созданные с помощью тегов `<TH>` и `<TD>`. Доступ к элементам коллекций осуществляется либо по индексу, либо по значению атрибута `ID` в соответствующем теге. Так, для доступа к строке таблицы можно брать значение `ID` в теге `<TR>`, а для доступа к ячейке — значение `ID` в теге `<TH>` или `<TD>`. При использовании индекса (номера) следует иметь в виду, что нумерация начинается с 0. При этом ячейки таблицы нумеруются слева направо и сверху вниз.

Рассмотрим пример простой таблицы, содержащей три столбца и четыре строки (листинг 3.57).

Листинг 3.57. Пример таблицы из трех столбцов и четырех строк

```
<HTML>
  <HEAD><TITLE>Таблица</TITLE></HEAD>
  <TABLE ID="mytab" BORDER=1>
```

```
    <THEAD>Список сотрудников</THEAD>
    <TH>Фамилия</TH><TH>Имя</TH><TH>Должность</TH>
    <TR ID="r1">
      <TD>Иванов</TD><TD>Иван</TD><TD>Директор</TD>
    </TR>
    <TR ID="r2">
      <TD>Петров</TD><TD>Петр</TD><TD>Заместитель директора</TD>
    </TR>
    <TR ID="r3">
      <TD>Сидоров</TD><TD>Сидор</TD><TD>Сторож</TD>
    </TR>
    <TR ID="r4">
      <TD>Федоров</TD><TD>Федор</TD><TD>Водитель</TD>
    </TR>
  </TABLE>
</HTML>
```

Ниже приведено несколько примеров ссылок на элементы таблицы из сценария:

```
document.all.mytab.rows[0]      /* ссылка на строку заголовков
                                   столбцов */
document.all.mytab.rows[1]      /* ссылка на первую строку данных */
document.all.mytab.rows["r1"]   /* другой способ ссылки на первую
                                   строку данных */

document.all.mytab.cells[0]      /* ссылка на ячейку, содержащую
                                    "Фамилия" */
document.all.mytab.cells[3]      /* ссылка на ячейку, содержащую
                                    "Иванов" */
document.all.mytab.cells[4]      /* ссылка на ячейку, содержащую
                                    "Иван" */
document.all.mytab.cells[6]      /* ссылка на ячейку, содержащую
                                    "Петров" */
```

Чтобы обратиться к строке или ячейке таблицы, следует учесть иерархию объектов: сначала мы обращаемся к коллекции `all` всех элементов документа, затем к таблице и только после этого — к элементу таблицы. Обратите внимание, что эти ссылочные выражения возвращают не содержимое

элементов таблицы (строк или ячеек), а лишь ссылки на элементы как на объекты.

Каждая строка таблицы (элемент коллекции `rows`), являясь объектом, имеет собственную коллекцию ячеек, которая называется `cells`, так же как и коллекция всех ячеек таблицы. Однако нумерация ячеек в этой коллекции происходит в пределах одной строки, начиная с 0. Например, в случае трехстолбцовой таблицы ячейки из коллекции `cells` для одной строки имеют индексы 0, 1 и 2. Ниже приведено несколько примеров:

```
document.all.mytab.rows[0].cells[0]     /* ссылка на ячейку,
                                           содержащую "Фамилия" */
document.all.mytab.rows[1].cells[0]     /* ссылка на ячейку,
                                           содержащую "Иванов" */
document.all.mytab.rows[1].cells[1]     /* ссылка на ячейку,
                                           содержащую "Иван"*/
document.all.mytab.rows[2].cells[0]     /* ссылка на ячейку,
                                           содержащую "Петров" */
document.all.mytab.rows["r2"].cells[1]  /* ссылка на ячейку,
                                           содержащую "Петр" */
```

Ранее мы рассмотрели создание ссылок на элементы таблицы. Эти ссылки применяются для чтения и изменения содержимого таблицы. В IE4+ для этой цели можно использовать свойства `innerText` и `innerHTML`. С помощью свойства `innerText` читается и изменяется обычное текстовое содержимое ячеек. В случае, когда в ячейке находится HTML-код, то для его изменения необходимо аналогичным образом воспользоваться свойством `innerHTML`. Впрочем, если содержимое ячейки является обычным текстом, то вполне можно использовать и `innerHTML`. Таким образом, свойство `innerHTML` универсальнее, чем `innerText`. Если таблица выводится в окно браузера (т. е. видима), изменение содержимого ее ячеек отобразится автоматически.

```
document.all.mytab.rows[1].innerText            // ИвановИванДиректор
document.all.mytab.cells[3].innerText           // Иванов
document.all.mytab.rows[1].cells[2].innerText   // Директор

/* Изменяем текстовое содержимое ячейки: */
document.all.mytab.rows[1].cells[2].innerText="Сторож"
/* Заменяем текстовое содержимое ячейки на изображение: */
document.all.mytab.rows[1].cells[2].innerHTML="<IMG SRC='pict.jpg'>"
```

Следует иметь в виду, что изменять содержимое таблицы необходимо по ячейкам, а не по строкам. Иначе говоря, попытка использовать выражение вида:

```
document.all.mytab.rows[индекс].innerText=новое_значение
```

приведет к удалению строки и, возможно, к ошибкам выполнения сценария.

Коллекции строк и ячеек, как и любой массив, обладает свойством `length`, значением которого является количество элементов в коллекции. Например,

```
document.all.mytab.rows.length           // количество всех строк
document.all.mytab.cells.length          // количество всех ячеек
document.all.mytab.rows[2].cells.length  /* количество ячеек в
                                            строке с индексом 2 */
```

3.11.2. Добавление и удаление строк таблицы

Добавление новой строки в таблицу производится с помощью метода `insertRow()`. Он возвращает ссылку на вновь созданную строку, которая затем используется для вставки ячеек. Ячейки вставляются в строку с помощью метода `insertCell(`*индекс_ячейки*`)`. Данный метод возвращает ссылку на созданную ячейку, которая затем используется для задания содержимого ячейки. Ячейки вставляются в строку по порядку без пропусков, начиная с нулевой. Так, нельзя создать ячейку с индексом 3, если ячейки с индексами 0, 1 и 2 еще не созданы. С другой стороны, вы можете последовательно создать любое количество ячеек в строке.

Таким образом, новая строка создается в три этапа: сначала к таблице добавляется новая строка, затем в нее вставляются ячейки, после чего они заполняются значениями. Далее приведен пример добавления строки к существующей таблице и заполнения ее ячеек значениями:

```
newrow=document.all.mytab.insertRow()   // добавляем новую строку
newcell=newrow.insertCell(0)            // вставляем ячейку с индексом 0
newcell.innerText="Привет"              // заполняем ячейку значением
newcell=newrow.insertCell(1)
newcell.innerHTML="<IMG SRC='pict.jpg'>"
newcell=newrow.insertCell(2)
newcell.innerHTML="<B>Охранник</B>'>"
```

Для удаления строки таблицы служит метод `deleteRow(`*индекс_строки*`)`. Параметр указывает номер удаляемой строки. Например:

```
document.all.mytab.deleteRow(2)
```

3.12. Простая база данных в текстовом файле

3.12.1. Элемент управления табличными данными

Достаточно эффективный способ формирования таблиц — использование специального элемента управления ActiveX, называемого Simple Tabular Data (STD, простые табличные данные) или Text Data Service (TDS, обслуживание текстовых данных). Этот элемент, встроенный в браузер IE4+, позволяет работать с табличными данными, содержащимися в обычных текстовых файлах.

Данные с табличной организацией имеют в качестве разделителей полей (столбцов) и строк различные символы, не используемые в представлении собственно данных. Файлы с табличными данными еще называют CSV-файлами (comma separated values, значения, разделенные запятой), хотя в качестве разделителя полей может использоваться не только запятая.

Создавать такие файлы можно вручную с помощью текстового редактора (например, Блокнота Windows). Кроме того, в большинстве систем работы с базами данных возможно преобразование специфических файлов в текстовые с заданными разделителями.

В HTML-документ элемент управления STD вставляется с помощью тега `<object>`, как в следующем примере:

```
<object classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        id="mydbcontrol">
  <param name="FieldDelim" value="|">
  <param name="DataURL" value="mydb.txt">
  <param name="UseHeader" value=true>
</object>
```

Здесь с помощью тегов `<param>` задаются параметры:

- `FieldDelim` — разделитель полей (ячеек); в примере значением этого параметра является вертикальная черта (`|`);
- `DataURL` — URL-адрес текстового файла с данными; в примере мы указали просто имя файла mydb.txt;
- `UseHeader` — определяет, содержит ли первая строка в текстовом файле имена полей.

В качестве разделителя строк обычно используется невидимый символ перевода строки (клавиша <Enter>). Однако можно задать любой символ в качестве разделителя строк с помощью параметра:

```
<param name="RowDelim" value="&#ASCII-код;">
```

По умолчанию значением атрибута `value` является `
`, т. е. ASCII-код символа перевода строки. Кроме перечисленных выше, элемент управления имеет и другие параметры, некоторое из них будут рассмотрены далее.

В первой строке текстового файла (источника данных) записываются через символьный разделитель имена столбцов. Они не обязательно должны совпадать с теми заголовками столбцов, которые вы хотите видеть на экране. Главное их назначение в том, чтобы обеспечить доступ к данным. Это так называемые *имена полей базы данных* или, другими словами, *идентификаторы полей*. Правила их образования такие же, как и для имен переменных. В листинге 3.58 показано содержимое текстового файла, хранящего сведения о некоторых моих книгах, в котором первая строка хранит имена полей, а остальные — собственно данные (значения полей). Разделителем полей здесь является вертикальная черта (`|`).

Листинг 3.58. Текстовый файл с табличными данными

```
Наименование|Издательство|Год|Обложка
HTML, скрипты и стили|БХВ-Петербург|2005|<img src=book1.png>
Самоучитель JavaScript|Питер|2005|<img src=book2.png>
Самоучитель Flash MX 2004|Питер|2005|<img src=book3.png>
```

Отображать данные на Web-странице должны элементы, определяемые HTML-тегами. Это могут быть теги таблицы, поля ввода данных, кнопки и др. Чтобы это стало возможным, необходимо привязать данные к элементам HTML-документа.

Для привязки данных к элементам HTML-документа необходимо в соответствующих тегах атрибуту `datasrc` присвоить значение, совпадающее со значением атрибута `id` тега `<object>`, установив перед ним символ `#`. Тем самым мы укажем, у какого объекта-поставщика следует взять данные. С помощью атрибутов `datafld` задаются имена полей из числа тех, что определены в первой строке текстового файла. В текстовом файле может содержаться не только обычная текстовая информация, но и HTML-коды. Чтобы эти коды интерпретировались, а не просто отображались, следует использовать атрибут `dataformatas=html`. Это позволяет, например, вставлять в ячейки таблицы изображения, гиперссылки, кнопки и другие элементы. Например, чтобы ото-

бразить в окне браузера данные из текстового файла mydb.txt (см. листинг 3.58) в виде таблицы, можно использовать HTML-код листинга 3.59. Вид этого документа в окне браузера показан на рис. 3.14.

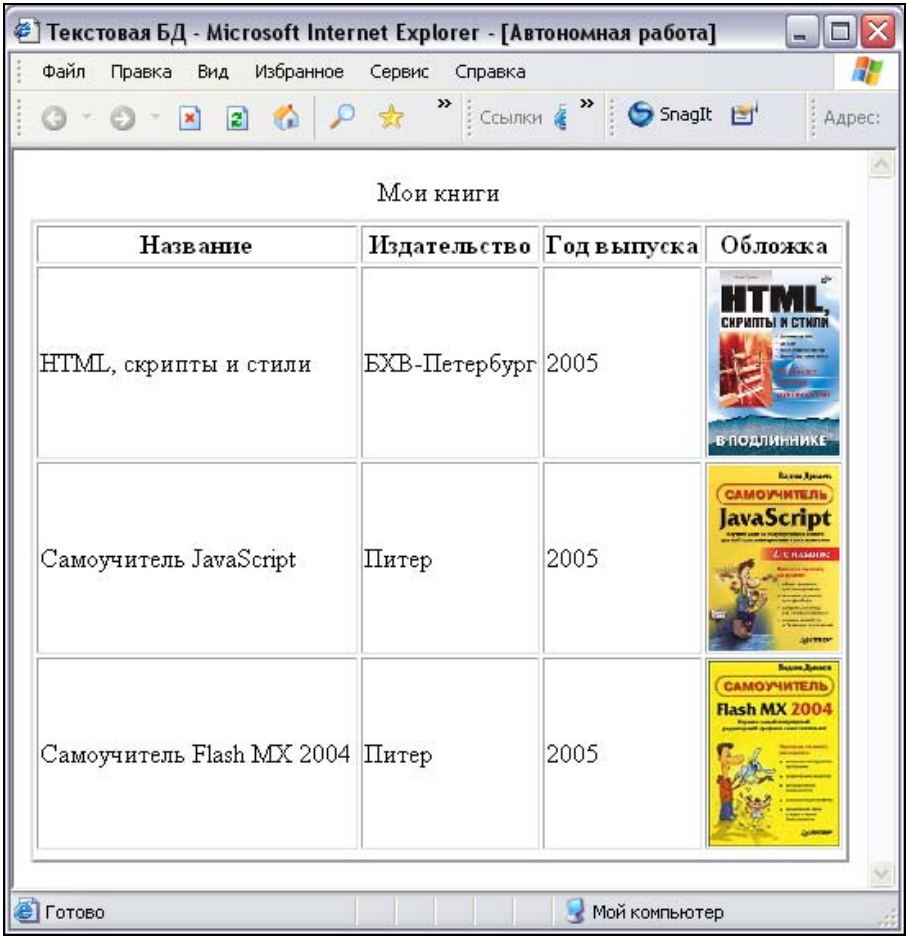


Рис. 3.14. Пример отображения табличных данных из текстового файла

Листинг 3.59. Отображение на Web-странице данных из текстового файла

```
<html>
  <head><title>Текстовая БД</title></head>
  <object classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
          id="mydbcontrol">
```

```
    <param name=FieldDelim value="|">
    <param name=DataURL    value=mydb.txt>
    <param name=UseHeader  value=true>
  </object>
  <table datasrc="#mydbcontrol" border= 5 width=500>
    <caption>Мои книги</caption>
    <thead>
      <th>Название</th>
      <th>Издательство</th>
      <th>Год выпуска</th>
      <th>Обложка</th>
    </thead>
    <tr>
      <td><span datafld=Наименование></span></td>
      <td><span datafld=Издательство></span></td>
      <td><span datafld=Год></span></td>
      <td><span datafld=Обложка dataformatas=html></span></td>
    </tr>
  </table>
<html>
```

Здесь в теге `<table>` атрибут `datasrc` в качестве значения имеет ссылку на элемент STD (значение атрибута `id` в теге `<object>`). Внутри тегов данных `<td>`, в контейнерах `<span>`, значениями атрибутов `datafld` являются имена полей, как они определены в текстовом файле. Написание этих имен должно в точности соответствовать данным, указанным в первой строке текстового файла. Однако порядок следования и количество выводимых на экран полей может быть произвольным. Например, можно вывести не все поля или, наоборот, продублировать некоторые из них. Кроме того, можно изменить порядок отображения полей. Этот порядок, а также состав отображаемых полей (столбцов) определяется в теге `<table>`.

Таким образом, при использовании STD организуется взаимодействие трех компонентов: элемент управления STD устанавливает связь с источником данных (текстовым файлом), элемент `<table>` связывается с STD и источником данных, текстовый файл содержит собственно данные.

Элемент управления STD имеет и другие параметры. В частности, есть параметры, с помощью которых можно установить фильтр (выборку) и сортировку данных.

Для установки фильтра предусмотрены следующие три параметра:

```
<param name="FilterColumn"    value="имя_поля">
<param name="FilterCriterion" value="оператор_сравнения">
<param name="FilterValue"     value=образец>
```

С помощью указанных параметров задается логическое условие (критерий) фильтра следующего вида:

имя_поля оператор_сравнения образец

Например, если требуется отфильтровать из таблицы все строки, в которых поле `Издательство` имеет значение `"БХВ-Петербург"`, то необходимо использовать следующие теги параметров:

```
<param name="FilterColumn"    value="Издательство">
<param name="FilterCriterion" value="==">
<param name="FilterValue"     value="БХВ-Петербург">
```

В качестве оператора сравнения можно использовать `=` (неточное равенство), `==` (точное равенство), `!=` (неравенство), `>` (больше), `<` (меньше), `>=` (не меньше), `<=` (не больше).

Чтобы выключить (сбросить) фильтр и сделать доступными все строки таблицы, достаточно задать условие, которому заведомо удовлетворяют все строки таблицы. Например, следующие параметры сбрасывают текущий фильтр, поскольку пустая строка содержится в любой строке:

```
<param name="FilterCriterion" value="=">
<param name="FilterValue"     value="">
```

По умолчанию фильтрация данных чувствительна к регистру, в котором набран образец. Управлять зависимостью от регистра можно с помощью параметра `CaseSensitive`:

```
<param name="CaseSensitive" value=значение>
```

Атрибут `value` может принимать значения `0` или `false` (не зависит) и `1` или `true` (зависит).

Для сортировки (упорядочения) данных предназначен следующий параметр:

```
<param name="SortColumn" value ="имя_поля">
```

Сортировка строк производится в соответствии с ASCII-кодом значений заданного поля базы данных. Например, если мы хотим, чтобы названия книг располагались по алфавиту, необходимо записать:

```
<param name="SortColumn" value="Наименование">
```

По умолчанию сортировка осуществляется по увеличению значения ASCII-кода символов. Однако порядок можно изменить на противоположный, если использовать следующий параметр:

```
<param name="SortAscending" value=0>
```

Отобразить данные, находящиеся во внешнем текстовом файле, могут и другие элементы HTML-документа. Так, текстовую информацию можно показать в элементах, соответствующих тегам `<input type=text>` и `<textarea>`. Кнопки `<input type=button>` и `<button>` отображают текстовые данные в виде надписей. Если данные во внешнем файле представляют собой адрес графического файла, то его можно показать с помощью тега `<img>`. Во всех перечисленных выше случаях теги должны содержать атрибуты `datasrc` и `datafld`. При этом в указанных элементах будет отображаться информация из первой строки данных текстового файла. Чтобы показать данные из других строк, необходимо осуществить перемещение по файлу к требуемой строке. Для этого элемент управления STD предоставляет свои методы и свойства, которые можно использовать в сценариях, но об этом будет рассказано в *разд. 3.12.2*.

В листинге 3.60 показано, как отобразить данные из файла mydb.txt (см. листинг 3.58) в поле ввода и на кнопке.

Листинг 3.60. Отображение данных из файла mydb.txt в поле ввода и на кнопке

```
<html>
  <head><title>Текстовая БД</title></head>
  <object classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
          id="mydbcontrol">
    <param name=FieldDelim value="|">
    <param name=DataURL value="mydb.txt">
    <param name=UseHeader  value=true>
  </object>
  <! Поле ввода >
  <input type=text datasrc="#mydbcontrol" datafld=Наименование size=30>
  <p>
  <! Кнопка >
  <input type=button datasrc="#mydbcontrol" datafld=Наименование>
  <p>
  <! Кнопка >
  <button datasrc="#mydbcontrol" datafld=Издательство></button>
<html>
```

Внимание

Изменения данных в таблице сохраняются только в оперативной памяти компьютера и никак не влияют на содержимое текстового файла-источника данных. Чтобы обновить содержимое таблицы, необходимо отредактировать текстовый файл и перекачать его на сервер, как это обычно делается при модификации страниц сайта.

3.12.2. Перемещение по записям простых баз данных

Элемент управления STD, рассмотренный в *разд. 3.12.1*, предоставляет объект `recordset`, который поддерживает методы перемещения по строкам таблицы (по записям базы данных). Синтаксис следующий:

id_объекта`.recordset.`*метод*

Например, `mydb.recordset.moveNext()` перемещает указатель текущей записи на следующую запись.

Методы перемещения по записям базы данных:

- `movePrevios()` — к предыдущей записи;
- `moveNext()` — к следующей записи;
- `moveFirst()` — к первой записи;
- `moveLast()` — к последней записи.

Перед использованием методов `moveNext()` и `movePrevios()` следует проверять значения свойств `eof` (конец файла данных) и `bof` (начало файла данных). Так, если текущей является последняя строка таблицы, то нельзя вызывать метод `moveNext()`. Аналогично, если текущей является первая строка, то нельзя использовать `movePrevios()`. Разумеется, методы перемещения применяются в сценариях обработки событий, таких как, например, щелчок на кнопке. Это необходимо, если на экран выводятся не все записи (строки) таблицы, а только одна запись (например, с помощью полей ввода). Перемещение по строкам таблицы осуществляется при поиске данных в соответствии с некоторым критерием и при их обработке.

Если использовать поля ввода (скажем, при создании формы для просмотра и коррекции данных), то привязку данных к ним можно выполнить с помощью следующего фрагмента HTML-кода:

```
<TR><TD>
  <INPUT TYPE="text" DATASRC=#mydbcontrol DATAFLD="Наименование">
</TD></TR>
```

Здесь mydbcontrol — значение атрибута ID тега <OBJECT>, с помощью которого вставлен элемент управления STD.

Обратите внимание, что атрибуты привязки данных DATASRC и DATAFLD совместно используются в одном и том же теге.

Рассмотрим следующий пример. Пусть данные находятся в текстовом файле mydb.txt (см. листинг 3.58). Создадим страницу, в которой отображаются данные только из одной записи (строки) таблицы. Для инициации перемещения по записям будут служить кнопки, к которым привяжем соответствующие обработчики щелчка. В листинге 3.61 приводится пример HTML-кода, который выводит на страницу одну запись базы данных с кнопками навигации.

Листинг 3.61. Форма с данными из одной записи и кнопками навигации

```
<HTML>
  <HEAD><TITLE>Пример формы БД</TITLE></HEAD>
  <BODY BGCOLOR=#e0e0e0>
    <OBJECT ID="mydbcontrol"
            CLASSID="CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
      <PARAM NAME="FieldDelim" VALUE="|">
      <PARAM NAME="DataURL"    VALUE="mydb.txt">
      <PARAM NAME="UseHeader"  VALUE=true>
    </OBJECT>

    <!-- Поля с данными -->
    <TABLE WIDTH=75%>
      <TR><TH>Наименование</TH>
      <TD>
        <INPUT TYPE=text
               DATASRC=#mydbcontrol DATAFLD="Наименование" >
      </TD></TR>
      <TR><TH>Издательство</TH>
        <TD><INPUT TYPE=text
                   DATASRC=#mydbcontrol DATAFLD="Издательство">
        </TD></TR>
      <TR><TH>Год</TH>
        <TD><INPUT TYPE=text
                   DATASRC=#mydbcontrol DATAFLD="Год">
        </TD></TR>
      <TR><TH>Обложка</TH>
```

```
        <TD><SPAN DATASRC=#mydbcontrol DATAFLD="Обложка"
                  DATAFORMATAS="html"></SPAN>
        </TD></TR>
    </TABLE>
    <P>
    <!-- Кнопки перемещения по записям -->
    <INPUT NAME=cmdFirst TYPE=BUTTON VALUE="<<"
           onclick="First()">
    <INPUT NAME=cmdPrevious TYPE=BUTTON VALUE="<"
           onclick ="Previous()">
    <INPUT NAME=cmdNext TYPE=BUTTON VALUE=">"
           onclick="Next()">
    <INPUT NAME=cmdLast TYPE=BUTTON VALUE=">>"
           onclick="Last()">
  </BODY>
  <SCRIPT>
    function First() {                        // к первой записи
      mydbcontrol.recordset.moveFirst()
    }

    function Previous() {                     // к предыдущей записи
      if (! mydbcontrol.recordset.bof)
        mydbcontrol.recordset.movePrevious()
    }

    function Next() {                         // к следующей записи
      if (! mydbcontrol.recordset.eof)
        mydbcontrol.recordset.moveNext()
    }

    function Last() {                         // к последней записи
      mydbcontrol.recordset.moveLast()
    }
  </SCRIPT>
</HTML>
```

Обратите внимание, что четвертое поле с именем `Обложка` содержит изображение, и поэтому мы показываем его не с помощью элемента `<INPUT>`, а с помощью контейнера `<SPAN>` с атрибутом `DATAFORMATAS="html"`.

3.12.3. Сортировка данных таблицы

Рассмотрим сортировку строк таблицы по значениям того или иного столбца. В нашем примере для сортировки необходимо просто щелкнуть на заголовке нужного столбца таблицы. Для этого, кроме элемента управления STD и таблицы, нам понадобится сценарий, содержащий функцию, обрабатывающую щелчок путем модификации тега `<OBJECT>` (листинг 3.62). Здесь использован следующий прием. Сначала база данных показывается без всякой сортировки. Мы сохраняем блок тегов от `<OBJECT>` до `</OBJECT>`, за исключением последнего, в переменной `obj`. При щелчке на заголовке таблицы вызывается функция сортировки `sort(field)`, которой передается имя поля. Эта функция дописывает к значению переменной `obj` строку, содержащую теги параметров, отвечающие за сортировку. Далее функция `sort()` заменяет в HTML-коде теги `<OBJECT>` на те, которые содержатся в переменной `obj`. Это делается с помощью свойства `outerHTML`. Элемент управления STD заново инициализируется с новыми параметрами. В результате таблица перерисовывается в соответствии с новым порядком следования строк.

Листинг 3.62. Сортировка данных таблицы

```
<HTML>
  <OBJECT ID="mydbcontrol"
          CLASSID="CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME="FieldDelim"    VALUE="|">
    <PARAM NAME="DataURL"       VALUE="mydb.txt">
    <PARAM NAME="UseHeader"     VALUE=true>
    <PARAM NAME="SortColumn"    VALUE="Год">
    <PARAM NAME="SortAscending" VALUE=1>
  </OBJECT>

  <TABLE DATASRC=#mydbcontrol BORDER=5>
    <THEAD>
      <TH onclick="sort('Наименование')">Наименование</TH>
      <TH onclick="sort('Издательство')">Издательство</TH>
      <TH onclick="sort('Год')">Год</TH>
      <TH>Обложка</TH>
    </THEAD>
    <TR>
      <TD><SPAN DATAFLD="Наименование"></SPAN></TD>
      <TD><SPAN DATAFLD="Издательство"></SPAN></TD>
```

```
      <TD><SPAN DATAFLD="Год"></SPAN></TD>
      <TD><SPAN DATAFLD="Обложка" DATAFORMATAS="html"> </SPAN></TD>
    </TR>
  </TABLE>

  <SCRIPT>
    // Сохраняем теги объекта STD в переменной
    var x=document.all.mydbcontrol.innerHTML  /* теги, вложенные
                                                 в <object> */
    var obj='<OBJECT ID="mydbcontrol"
        CLASSID="CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">' + x

    function sort(field) {     /* сортировка по значениям столбца
                                  field */
      var y=document.all.mydbcontrol
      y.outerHTML=obj + '<PARAM NAME="SortColumn" VALUE="' +
                  field + '"></OBJECT>'
    }
  </SCRIPT>
</HTML>
```

В представленном примере сортировать данные можно только по первым трем текстовым полям. Вы можете в качестве упражнения усовершенствовать этот код. Например, сделайте так, чтобы при двойном щелчке на заголовке столбца таблицы сортировка происходила в противоположном порядке. Для этого необходимо не устанавливать параметр `<PARAM NAME="SortAscending" VALUE=0>`.

3.12.4. Фильтрация данных таблицы

Для организации фильтрации данных из таблицы по заданному критерию используется подход, аналогичный рассмотренному в *разд. 3.12.3*. В приведенном далее примере (листинг 3.63) в окно браузера выводится таблица с данными и элементы, с помощью которых можно задать поле (столбец) и значение (образец) для формирования простого условия фильтра вида: *имя_поля=значение*.

Имя поля выбирается из раскрывающегося списка, а значение вводится с клавиатуры. Для установки фильтра следует щелкнуть на кнопке **Применить**. Если поле ввода значения пусто, то при щелчке на этой кнопке фильтр выключится, а в таблице будут отображены все имеющиеся записи. Заметим,

что результат фильтрации в данном примере не зависит от регистра, в котором пользователь ввел значение.

Так же, как и при сортировке, мы сохраняем в переменной часть информации о параметрах элемента STD, а затем модифицируем ее, чтобы учесть условие фильтра. Для установки фильтра используется свойство `outerHTML`.

Листинг 3.63. Фильтрация данных таблицы

```
<HTML>
  <HEAD><TITLE>Пример фильтрации</TITLE></HEAD>
  <BODY BGCOLOR=#e0e0e0>
    <H3>Фильтр:</H3>
    Поле
    <! Раскрывающийся список имен полей >
    <SELECT NAME="FLD">
      <OPTION value="Наименование">Наименование
      <OPTION value="Издательство">Издательство
      <OPTION value="Год">Год
    </SELECT>
    <BR>
    Значение
    <! Поле ввода значения для фильтра и кнопка >
    <INPUT NAME="INP" VALUE ="" TYPE="text">
    <P>
    <BUTTON onclick="filter()">Применить</BUTTON>
    <HR>

    <! Элемент управления STD >
    <OBJECT ID="mydbcontrol"
            CLASSID="CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83">
      <PARAM NAME ="FieldDelim" VALUE="|">
      <PARAM NAME="DataURL"     VALUE="mydb.txt">
      <PARAM NAME="UseHeader"   VALUE=true>
    </OBJECT>

    <! Таблица для вывода данных >
    <TABLE DATASRC=#mydbcontrol border=5>
      <THEAD>
        <TH>Наименование</TH>
```

```
        <TH>Издательство</TH>
        <TH>Год</TH>
        <TH>Обложка</TH>
      </THEAD>
      <TR>
        <TD><SPAN DATAFLD="Наименование"></SPAN></TD>
        <TD><SPAN DATAFLD="Издательство"></SPAN></TD>
        <TD><SPAN DATAFLD="Год"></SPAN></TD>
        <TD><SPAN DATAFLD="Обложка" DATAFORMATAS="html"> </SPAN></TD>
      </TR>
    </TABLE>
  </BODY>
  <SCRIPT>
    /* сохраняем основные параметры элемента STD в переменной obj */
    var obj="<OBJECT ID='mydbcontrol' "
    obj+= "CLASSID='CLSID:333C7BC4-460F-11D0-BC04-0080C7055A83'>"
    obj+="<PARAM NAME='FieldDelim' Value='|'>"
    obj+= "<PARAM NAME='DataURL' Value='mydb.txt'>"
    obj+= "<PARAM NAME='UseHeader' value=true>"

    function filter() {        // установка фильтра
      var cpar=""              /* переменная для хранения
                                  параметров фильтра */
      if (INP.value){          // если введено значение
        cpar="<param name='FilterColumn' value='"+FLD.value+"'>"
        cpar+= "<param name='FilterValue' value='"+INP.value+"'>"
        cpar+= "<param name='FilterCriterion' value='=' >"
        cpar+="<PARAM NAME='CaseSensitive' VALUE=false>"
      }
      document.all.mydbcontrol.outerHTML=obj + cpar + "</OBJECT>"
    }
  </SCRIPT>
</HTML>
```

Совет

Для тренировки усовершенствуйте рассмотренный выше код таким образом, чтобы можно было выбирать оператор сравнения из раскрывающегося списка, а также устанавливать режим зависимости или независимости от регистра.

3.13. Поиск в тексте

Тексты большого объема, расположенные на Web-странице, обычно снабжают поисковой системой. Она может содержать поле ввода поискового образа (набора символов, который требуется найти в тексте) и кнопку, щелчок на которой запускает процедуру поиска. В простейшем варианте эта процедура прокручивает текст в окне так, чтобы найденный поисковый образ оказался видимым и выделенным. Если поиск оказался неудачным, то положение текста в окне остается неизменным и, возможно, появляется соответствующее сообщение.

Для решения задачи поиска в тексте используется объект `TextRange`. Этот объект браузера IE4+ обеспечивает доступ к текстовой информации, находящейся в объектах, которые соответствуют тегам `<BODY>`, `<TEXTAREA>`, `<BUTTON>` и `<INPUT TYPE=text>`.

В листинге 3.64 организуется поиск в тексте, находящемся в теле HTML-документа, т. е. в контейнерном теге `<BODY>`. В этот же контейнер помещены поле ввода поискового образа и кнопка для инициации поиска. Сценарий содержит описание функции `myfind()`, вызываемой при щелчке на кнопке. Функция `myfind()` с помощью метода `createTextRange()` создает текстовую область. Это не копия текста, содержащегося в теле документа, а просто ссылка на него. Далее, с помощью метода `findText()` производится поиск в текстовой области строки символов, которую ввел пользователь в поле с именем `WORD`. Наконец, с помощью метода `scrollIntoView()` содержимое окна прокручивается так, чтобы найденная строка символов оказалась видимой. С помощью метода `select()` найденная строка в тексте выделяется цветом (подсвечивается).

Листинг 3.64. Поиск в текстовой области документа

```
<HTML>
  <BODY>
    <INPUT TYPE="text" NAME="WORD" VALUE="" SIZE=20 >
    <BUTTON onclick="myfind()">Поиск</BUTTON>
    <P>

    <! Здесь расположен текст, в котором производится поиск >

  </BODY>
  <SCRIPT>
```

```
    function myfind(){
      if (!WORD.value) return   // если поисковый образ пуст, выходим
      obj=document.body.createTextRange() // создаем текстовую область
      obj.findText(WORD.value)            // производим поиск
      obj.scrollIntoView()                /* прокручиваем текстовую
                                             область в окне */
      obj.select()                        // выделяем найденное
    }
  </SCRIPT>
</HTML>
```

Совет

В качестве упражнения измените рассмотренные выше функции поиска таким образом, чтобы в случае неудачного поиска в тексте на экране появилось сообщение "Не удалось найти". Для вывода сообщения воспользуйтесь стандартным методом `alert()`.

Если требуется осуществлять поиск не по всему документу, а только в текстовой области, заданной тегом `<TEXTAREA ID=mytextarea>`, то для создания объекта `TextRange` следует воспользоваться выражением:

```
obj=document.mytextarea.createTextRange()
```

Рассмотренный выше код обеспечивает одноразовый поиск, который останавливается, как только будет найдено первое вхождение поискового образа или достигнут конец текстовой области.

Теперь рассмотрим итеративный поиск, т. е. поиск всех вхождений поискового образа. Для этого следует воспользоваться методом `collapse()` объекта `TextRange`. Значением `false` параметр для этого метода перемещает точку начала поиска в конец найденного фрагмента текста, а последующее применение метода `findText()` инициирует поиск с этой, а не начальной точки. Разумеется, метод `findText()` необходимо применять в цикле. Чтобы выделить все найденные фрагменты, достаточно воспользоваться методом `execCommand("BackColor", false, цвет)`. Определение функции для итеративного поиска текстовой строки, введенной в поле с именем `WORD`, представлено в листинге 3.65.

Листинг 3.65. Функция для итеративного поиска

```
function myfind(){
if (WORD.value=="") return
```

```
  obj=document.body.createTextRange()
  while (obj.findText(WORD.value)){
    obj.select()
    obj.execCommand("backcolor","false","yellow")
    obj.collapse(false)
    obj.scrollIntoView()
}
}
```

Примечание

Метод `exeCommand()` объекта `TextRange` позволяет выполнить множество команд. Например, команды **Copy** и **Cut** копируют текстовую область в буфер обмена соответственно без и с удалением ее.

3.14. Движение элементов по заданной траектории

Анимационные эффекты на Web-странице можно создать с помощью сценариев и графических изображений. Эффект может быть усилен, если графическое изображение уже является анимационным. Такие изображения сохраняются в анимационных gif-файлах. С помощью сценария вы также можете перемещать вставленные Flash-ролики.

Схема сценария, осуществляющего непрерывное перемещение видимого элемента документа, имеет следующий вид:

```
function curvemove(){       // инициализация движения
  ...                       // подготовка к запуску функции move()
  setInterval("move()", задержка)
}
function move(){
  ...                       /* изменение координат top и left стиля
                               перемещаемого элемента */
}
curvemove()                 // вызов функции для перемещения элемента
```

Таким образом, мы создаем две функции, имена которых могут быть произвольными. Первая функция, `curvemove()`, осуществляет подготовку исход-

ных данных и вызывает метод `setInterval()` с указанием в качестве первого параметра имени второй функции `move()` в кавычках. Вообще говоря, первый параметр метода `setInterva()` является строкой, содержащей выражение, которое должно выполняться периодически во времени. Вторая функция, `move()`, изменяет координаты элемента. Поскольку метод `setInterval()` вызывает функцию `move()` периодически через заданное количество миллисекунд, то координаты элемента изменяются постоянно. При этом создается эффект движения. Скорость и плавность движения зависят от величин приращения координат (в функции `move()`) и временной задержки (второго параметра метода `setInterval()`). Чтобы начать перемещение элемента, необходимо просто вызвать первую функцию, `curvemove()`.

Чтобы иметь возможность прекратить движение элемента, следует сохранить значение, возвращаемое методом `setInterval()`, в глобальной переменной, а затем использовать его в качестве параметра метода `clearInterval()` в теле функции `move()`.

Рассмотрим задачу организации движения видимого элемента по произвольной кривой, заданной выражениями с одной переменной. Первое выражение описывает изменение вертикальной координаты элемента, а второе — горизонтальной. Эти выражения будут содержать одну переменную, которую мы обозначим через `x` — строчной латинской буквой. Переменную `x` можно интерпретировать как независимый параметр движения (например, время). С помощью встроенной функции `eval()` можно вычислить значения этих выражений при конкретном значении переменной `x` и присвоить их стилевым параметрам `left` и `top` перемещаемого элемента. Функция (пусть это будет `move()`), которая все это выполняет, передается в качестве первого параметра методу `setInterval()`, периодически вызывающему ее через заданный интервал времени.

Функция инициализации движения `curvemove()` будет принимать три строковых и один числовой параметр. Строковые параметры содержат соответственно значение идентификатора `ID` перемещаемого элемента, выражение для вертикальной координаты и выражение для горизонтальной координаты. Числовой параметр определяет период времени, через который координаты элемента пересчитываются. Вид кривой зависит от выражений, задающих характер изменения вертикальной и горизонтальной координат.

В листинге 3.66 приводятся определения функций `curvemove()` и `move()`.

Листинг 3.66. Функции движения по произвольной кривой

```
function curvemove(xid, yexpr, xexpr, ztime) {
  /* Движение по произвольной кривой. Вариант 1.
     Параметры:
     xid — id движущегося объекта, строка
     yexpr — выражение для вертикальной координаты
     xexpr — выражение для горизонтальной координаты
     ztime — интервал времени между вызовами функции move(), мс
  */
  if (!xid) return null
  if (!yexpr) yexpr="x"
  if (!xexpr) xexpr="x"
  if (!ztime) ztime=100                // интервал времени, мс
  x=0     /* глобальная переменная, входящая в выражения yexpr и xexpr */
  setInterval("move('" + xid + "', '" + yexpr + "', '" + xexpr + "')",
              ztime)
}

function move(xid, yexpr, xexpr) {
  x++
  document.all[xid].style.top=eval(yexpr)
  document.all[xid].style.left=eval(xexpr)
}
```

Обратите внимание, что переменная `x` в функции `curvemove()` является глобальной и поэтому доступна в функции `move()`. Чтобы сделать переменную `x` глобальной, мы просто не использовали ключевое слово `var` перед первым ее появлением в коде.

Исходное позиционирование перемещаемого элемента с помощью таблицы стилей не играет особой роли, поскольку при вызове функции `curvemove()` элемент помещается в точку с координатами, равными значениям выражений `xexpr` и `yexpr`, вычисленным при `x=0`. Поэтому начальное позиционирование следует задавать с помощью соответствующего выбора этих выражений.

Как уже отмечалось, вид траектории движения определяется выражениями `xexpr` и `yexpr`, аргументом которых является переменная `x`.

Далее приведены несколько частных случаев.

- Прямая линия:

  ```
  yexpr="a*x + y0";
  xexpr="b*x + x0";
  ```

  где `a`, `y0`, `b`, `x0` — числовые коэффициенты.

- Синусоида (волнистая линия):

  ```
  yexpr="A*Math.sin(w*x + f)";
  xexpr="x";
  ```

  где `A`, `w`, `f` — числовые коэффициенты.

- Эллипс (овал):

  ```
  yexpr="A*Math.sin(w*x + f)";
  xexpr="B*Math.cos(w*x + f)";
  ```

  где `A`, `B`, `w`, `f` — числовые коэффициенты.

  При `A=B` эллипс превращается в круг.

В листинге 3.67 приведен пример HTML-документа с движущимся изображением.

Листинг 3.67. Движущееся изображение

```
<HTML>
  <IMG ID="myimg" SRC="picture.gif" STYLE="position:absolute">
  <SCRIPT>
    function curvemove(xid, yexpr, xexpr, ztime) {
      ...          // код определения функции (см. листинг 3.66)
    }

    function move(xid, yexpr, xexpr) {
      ...           // код определения функции (см. листинг 3.66)
    }
    /* Вызов функции движения */
    curvemove("myimg", "100 + 50*Math.sin(0.03*x)", "50+x", 100)
  </SCRIPT>
</HTML>
```

В этом примере изображение из файла picture.gif будет перемещаться по синусоиде с амплитудой 50 пикселов и горизонтальной скоростью 10 пикселов

в секунду. Начальные координаты графического объекта равны 100 и 50 пикселов по вертикали и горизонтали соответственно.

3.15. Рисование линий

Если вам потребуется отобразить в окне браузера прямоугольник или горизонтальную линию, то для этого можно воспользоваться, например, HTML-тегами `<TABLE>` и `<HR>` соответственно. Однако иногда необходимо нарисовать наклонную прямую, овал или построить график какой-нибудь зависимости. Это можно сделать с помощью сценария.

Алгоритм работы сценария, рисующего линии, довольно прост. Нужно вывести на экран изображение размером 1×1 пиксел, залитое цветом, отличающимся от цвета фона. Это изображение следует разместить несколько раз в соответствии с координатами, которые задаются параметрами позиционирования `top` и `left` атрибута `STYLE` тега `<IMG>`. С помощью сценария можно сформировать строку, содержащую теги `<IMG>` с необходимыми атрибутами, а затем записать ее в документ методом `write()`. В остальном алгоритм решения данной задачи очень похож на алгоритм перемещения элементов *(см. разд. 3.14)*.

Точки, из которых состоит линия, можно задать посредством HTML-кода вида:

```
<IMG SRC="point.bmp" STYLE="position:absolute;top:y;left:x"
     WIDTH=n HEIGHT=n>
```

Здесь point.bmp — имя графического файла, содержащего один пиксел; `y`, `x` — числа, указывающие положение графического файла в пикселах. Одинаковые значения атрибутов `WIDTH` (ширина) и `HEIGHT` (высота) задают представление точки в виде квадрата размером $n \times n$ пикселов. При этом точка с исходными размерами 1×1 пиксел просто растягивается. Таким образом, мы можем задать отображаемые размеры (масштаб) одной точки, а следовательно, и определить толщину линии.

Вообще говоря, в качестве точки не обязательно использовать картинку размером 1×1 пиксел. Это может быть любое изображение.

Кривую будем задавать посредством двух выражений с аргументом `x`. Первое выражение определяет изменение вертикальной координаты, а второе — горизонтальной. Это так называемое параметрическое задание кривой. Поскольку такая кривая может продолжаться сколь угодно далеко, следует предусмотреть ограничение ее длины. Например, можно задать количество точек, из которых будет состоять кривая. Кроме того, неплохо предусмотреть

возможность рисования и прерывистых (штриховых линий). Подбирая графические файлы, вид кривой и длину штриха, можно получить интересные визуальные эффекты.

Функция, обеспечивающая рисование линий, представлена в листинге 3.68.

Листинг 3.68. Функция рисования кривых

```
function curve(pict_file, yexpr, xexpr, x0, y0, t, n, s){
  /* pict_file — имя графического файла
     yexpr — выражение с переменной y для вертикальной координаты
     xexpr — выражение с переменной x для горизонтальной координаты
     x0, y0 — координаты начала кривой
     t — количество точек кривой (значений переменной x)
     n — толщина линии
     s — длина штриха и паузы
  */
  if (!yexpr) return null
  if (!xexpr) xexpr="x"
  if (!pict_file) pict_file="point.bmp"
  if (!s) s=0
  if (!t) t=100
  var clinewidth=""
  if (n) clinewidth= 'WIDTH=' + n + 'HEIGHT=' + n
  var x
  xstr0='<IMG SRC="' + pict_file + '" ' + clinewidth +
        ' STYLE="position:absolute; top:'
  xstr=""
  var i=0, draw=true
  for(x=0; x <  t; x++){
    if (draw)
      xstr += xstr0 + (y0 + eval(yexpr)) + '; left:' +
              (x0 + eval(xexpr)) + '">'
    if (i > s&&s > 0) {
      draw=!draw
      i=0
    }
    i++
  }
  document.write(xstr)            // запись в документ
}
```

Обратите внимание, если имя файла с изображением точки кривой не задано, то подразумевается point.bmp.

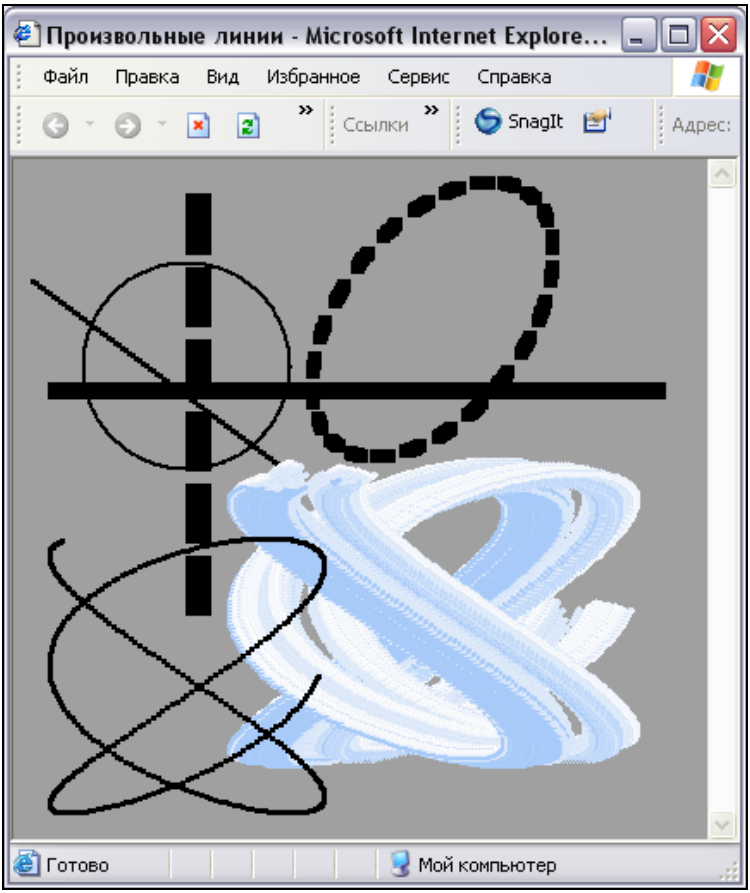


Рис. 3.15. Линии, нарисованные с помощью функции `curve()`

Ниже приводятся несколько примеров (рис. 3.15):

- вертикальная штриховая прямая

```
curve("","x","100",0,20,250,15,20)
```

- горизонтальная прямая

```
curve("","100","x",20,30,350,10)
```

- наклонная прямая

```
curve("", "1.5*x+20", "2*x+10",0,50,150,3)
```

- окружность

```
curve("","60*Math.sin(6/25*x)","60*Math.cos(6/25*x)",
      100,120,300,2,0)
```

- эллипс штриховой

```
curve("","80*Math.sin(6/25*x)","70*Math.cos(6/25*x+0.5)",
      240,90,300,8,5)
```

- фигура Лиссажу

```
curve("", "80*Math.sin(6/250*x)",
      "80*Math.cos(6/170*x+0.3)",100,300,450,3,0)
```

- фигура Лиссажу с изображением точки в виде облака

```
curve("obl.gif", "80*Math.sin(6/250*x)", "80*Math.cos(6/170*x+0.3)",
       200,250,450,90,0)
```

Примечание

Рассмотренная функция `curve()` сама отображает кривую в документе. Однако можно сделать так, чтобы она возвращала строку тегов `xstr`, не отображая ее. Тогда рисование кривой можно выполнить где-то в другом месте сценария.

3.16. Обработка данных форм

В HTML форма создается с помощью контейнерного тега `<FORM>`, внутри которого располагаются теги элементов пользовательского интерфейса — поля ввода данных, текстовые области, переключатели и флажки, раскрывающиеся списки и кнопки *(см. разд. 1.2.7)*. В объектной модели документа каждой форме соответствует свой объект, входящий в коллекцию `forms`. Форма предназначена главным образом для организации отправки на сервер всех данных, имеющихся в элементах этой формы, — введенных пользователем и/или сгенерированных сценарием. Для отправки данных на сервер сценарий не обязателен. Чтобы отправить данные, достаточно в теге `<FORM>` атрибуту `ACTION` присвоить надлежащее значение — URL-адрес получателя, а в самой форме разместить кнопку типа **Отправить** (`submit`). Щелчок на этой кнопке инициализирует отправку данных. Если атрибут `ACTION` не указан или его значение пусто, данные формы не будут отправлены, даже если вы щелкнете на кнопке типа **Отправить** (`submit`).

Внимание

Вместо обычной кнопки типа `submit` (`<input type=submit>`) можно использовать графическое изображение (`<input type=image src="`*адрес_файла*`">`).

Итак, для отправки данных формы атрибут ACTION должен иметь некоторое значение. В общем случае это URL-адрес файла или программы, которая получает и обрабатывает отправленные данные. Например:

```
ACTION="http://www.myserver/cgi/myprogram.php"
```

Если вы хотите отправлять данные формы по электронной почте, то значением ACTION является строка вида:

```
mailto:адрес_e-mail
```

Можно также указать тему (subject) сообщения:

```
mailto:адрес_e-mail?subject=тема_сообщения
```

Кроме атрибута ACTION в теге <FORM> следует указать еще два атрибута: METHOD и ENCTYPE. Атрибут METHOD имеет значение POST или GET. При использовании метода GET передаваемые данные видны в адресной строке браузера и следуют за адресом. Обычно этот метод применяют для передачи небольшого объема данных. Если у вас нет особых причин задумываться об этом, выбирайте значение POST. Атрибуту ENCTYPE присвойте значение "text/plain". В этом случае отправляемое сообщение будет представлять собой последовательность пар вида *имя_элемента=значение*. Здесь *имя_элемента* — значение атрибута NAME в теге элемента, содержащегося в форме, а *значение* — значение атрибута VALUE в этом же теге. Если не указать атрибут ENCTYPE, то сообщение будет представлено в неудобочитаемом (закодированном) виде.

Если перед отправкой данных формы требуется предварительно их проверить или еще что-нибудь сделать, то для этого необходим сценарий. В примере из листинга 3.69 проверяется, имеется ли символ @ в поле ввода адреса электронной почты получателя и не пусто ли поле ввода сообщения. Если символа @ в адресе нет или поле сообщения пусто, то отправка не производится. Сценарий обрабатывает событие onsubmit, возникающее при щелчке на кнопке типа **Отправить** (submit).

Листинг 3.69. Проверка наличия символа @ в поле ввода адреса электронной почты

```
<HTML>
  <HEAD><TITLE>Отправка данных по E-mail</TITLE></HEAD>
  <FORM ID="myform" METHOD=POST ACTION=""
        ENCTYPE="text/plain" style="background:'e0e0e0'">
```

```
    Кому:
    <INPUT NAME="email_to" TYPE="text" VALUE="">
    <P>
    От кого:
    <INPUT NAME="email_from" TYPE="text" VALUE="">
    <P>
    Сообщение:<BR>
    <TEXTAREA NAME="Сообщение" TYPE="text" VALUE=""></TEXTAREA>
    <P>
    <! Кнопка типа Submit >
    <INPUT NAME="Отправить" TYPE="submit" VALUE="Отправить">
  </FORM>
  <SCRIPT>
    function myform.onsubmit(){
      var noemail=myform.email_to.value.indexOf('@') == — 1
           /* присваиваем значение равенства */
      var notext=!myform.Сообщение.value
      var xtext="\nПисьмо не отправлено"
      if (noemail || notext){
        event.returnValue=false                    // отменить отправку
        if (noemail)
          alert("Неправильный адрес получателя" + xtext)
        else
          alert("Нет текста сообщения" + xtext)
      }else
        myform.action="mailto:" + myform.email_to.value// значение ACTION
    }
  </SCRIPT>
</HTML>
```

Если отправить данные рассмотренной ранее формы, то адресат получит сообщение в следующем виде:

```
email_to=qwerty@xxx.ru
email_from=myname@wwww.ru
Сообщение=Желаю счастья
Отправить=Отправить
```

Если данные не удовлетворяют нашему критерию правильности, то следует предотвратить их отправку. Это можно сделать двумя способами.

- Присвоить свойству `event.returnValue` значение `false` (как в нашем примере). Тогда отменяется стандартная реакция на событие, в данном случае — реакция на событие `onsubmit` (отправка сообщения).
- Присвоить свойству `action` объекта формы пустого значения. В нашем примере значение `action` пусто по умолчанию, поэтому выражение `event.returnValue=false` является излишним, но зато делает код сценария более понятным. В контексте нашего HTML-кода сценарий мог бы выглядеть и следующим образом:

```
function myform.onsubmit(){
  var noemail=myform.email_to.value.indexOf('@') == -1
  var notext=!myform.Сообщение.value
  var xtext="\nПисьмо не отправлено"
  if (noemail || notext){
   if (noemail)
     alert("Неправильный адрес получателя" + xtext)
   else
     alert("Нет текста сообщения" + xtext)
  }else
   myform.action="mailto:" + myform.email_to.value // значение ACTION
}
```

На Web-страницах нередко размещают ссылку или кнопку, открывающую форму отправки сообщения по электронной почте автору страницы. В этом случае не нужно вводить почтовый адрес получателя. Очевидно, что этот адрес уже присвоен в качестве значения атрибуту `ACTION` в теге `<FORM>`. Если вы хотите затруднить его выявление отправителем сообщения или, что важнее, различными программами-роботами, сканирующими Web-страницы с целью выявления адресов электронной почты, то следует предпринять некоторые меры. Самый простой рецепт — хранить отдельные компоненты адреса в различных переменных и собирать их с помощью выражения конкатенации (склейки), которое присваивается свойству `action`. В результате программа-робот не найдет в HTML-документе строки символов, имеющей структуру адреса электронной почты. В листинге 3.70 приведен пример HTML-кода, воплощающий эту нехитрую идею.

Листинг 3.70. Соединение адреса электронной почты *(вариант 1)*

```
<HTML>
  <FORM ID="myform" METHOD=POST ACTION="" ENCTYPE="text/plain"
        style="background:'e0e0e0'">
    <H2>Отправка почты автору</H2>
    От кого: <INPUT NAME="email_from" TYPE="text" VALUE="">
    <P>
    Сообщение:<BR>
    <TEXTAREA NAME="Сообщение" TYPE="text" VALUE=""></TEXTAREA>
    <P>
    <! Кнопка типа submit >
    <INPUT NAME="Отправить" TYPE="submit" VALUE="Отправить">
  </FORM>
  <SCRIPT>
    var first="mumu", second="gerasim"
    function myform.onsubmit(){
      if (!myform.Сообщение.value){
        event.returnValue=false           // отменить отправку
        alert("Нет текста сообщения \nПисьмо не отправлено")
      }else
        /* значение ACTION */
        myform.action="mailto:" + first + "@" + second + ".ru"
    }
  </SCRIPT>
</HTML>
```

Приведем еще один вариант привязки сценария обработки данных формы (листинг 3.71). Функционально он аналогичен рассмотренному выше.

Листинг 3.71. Соединение адреса электронной почты *(вариант 2)*

```
<HTML>
  <FORM ID="myform" METHOD=POST ACTION="" ENCTYPE="text/plain"
        onsubmit="return validator()" style ="background:'e0e0e0'">
    <H2>Отправка почты автору</H2>
    От кого:
    <INPUT NAME="email_from" TYPE="text" VALUE="">
    <P>
```

```
    Сообщение:<BR>
    <TEXTAREA NAME="Сообщение" TYPE="text" VALUE=""></TEXTAREA>
    <P>
    <! Кнопка типа submit >
    <INPUT NAME="Отправить" TYPE="submit" VALUE="Отправить">
  </FORM>
  <SCRIPT>
    var first="mumu", second="gerasim"
    function validatort(){
      if (!myform.Сообщение.value){
        alert("Нет текста сообщения \nПисьмо не отправлено")
        return false         // отменить отправку
      }else
       myform.action="mailto:" + first + "@" + second + ".ru" /* значение
                                                                ACTION */
      return=true            // разрешить отправку
    }
  </SCRIPT>
</HTML>
```

В этом примере функция `validator()` является обработчиком события `onsubmit` и возвращает либо `true`, либо `false` в зависимости от того, можно отправлять данные или нет. В теге `<FORM>` к событию `onsubmit` привязывается выражение `return validator()`, что эквивалентно присвоению значения свойству `event.returnValue` в теле функции-обработчика события.

В формах кроме элементов, в которые данные вводятся пользователем, можно разместить скрытые элементы (`<INPUT TYPE=HIDDEN>`). Эти элементы невидимы, и, следовательно, данные в них (значения атрибута `VALUE`) не могут быть изменены пользователем. Однако с помощью сценария можно сформировать такие данные и присвоить их свойству `value` скрытого элемента. При щелчке на кнопке типа `submit` эти данные в числе других будут также отправлены на сервер.

В листинге 3.72 приведен HTML-код, содержащий форму, в которой видимой является только картинка, играющая роль кнопки типа `submit`. Щелчок на этой кнопке отправляет на сервер **www.myserver.ru** программе anyprogram.php название браузера пользователя (свойство `appName` объекта `navigator`).

Листинг 3.72. Отправка на сервер данных из скрытого элемента формы

```
<HTML>
  <FORM NAME=myform
        ACTION=http://www.myserver.ru/anyprogram.php METHOD=POST>
    <INPUT TYPE=hidden NAME="mydata">
    <INPUT TYPE=image SRC="grbutton.png">
  </FORM>
  <SCRIPT>
    function myform.onsubmit(){
      myform.mydata.value=navigator.appName
    }
  </SCRIPT>
</HTML>
```

Таким образом, если форма содержит только скрытые элементы, то посетитель Web-страницы может и не догадаться, что, щелкнув на картинке или кнопке, он отправляет какие-то данные, сформированные программно.

Форма может содержать и большое количество элементов, например, полей ввода, расположенных в ячейках таблицы. В подобных случаях удобно задавать имена элементов как массивы. Тогда и передаваемые на сервер данные будут представлены в виде массива. В листинге 3.73 представлен пример, в котором отправляемые данные сгруппированы в массив `myText`. Указание квадратных скобок рядом с именем переменной определяет ее как массив. Следует иметь в виду, что это — специфический способ, применяемый только при передаче данных на сервер. Он не пригоден для доступа к элементам формы по их именам (значениям атрибутов `NAME`) в клиентских сценариях.

Листинг 3.73. Представление передаваемых данных в виде массивов

```
<HTML>
  <FORM NAME=myform
        ACTION=http://www.myserver.ru/anyprogram.php METHOD=POST>
    <INPUT TYPE=text NAME="myText[]">
    <INPUT TYPE=text NAME="myText[]">
    <INPUT TYPE=text NAME="myText[]">
    <INPUT TYPE=text NAME="myText[]">
    <INPUT TYPE=submit VALUE="Отправить">
  </FORM>
</HTML>
```

Глава 4

Основы PHP

4.1. Подготовка к программированию

Для изучения языка PHP очень важно самому выполнить на компьютере рассматриваемые далее примеры. Для этого кроме браузера и обычного текстового редактора (например, Блокнота Windows) понадобится программное обеспечение Web-сервера и модуль PHP. Эксперименты по программированию на языке PHP разумно проводить не с помощью удаленного Web-сервера, к которому необходимо интернет-подключение, а с помощью Web-сервера, установленного на одном из компьютеров локальной сети или даже на том же компьютере, на котором работаете вы.

В качестве Web-сервера можно использовать Apache или Internet Information Services (IIS). IIS для Windows NT/200/XP поставляется вместе с дистрибутивом операционной системы, но не устанавливается автоматически. Модуль PHP можно бесплатно получить по адресу **www.php.net**. Как установить и настроить PHP и IIS для Windows, рассказывается в *приложениях 2* и *3*. В данной книге при рассмотрении примеров предполагается работа с Web-сервером IIS.

Предположим, вы установили Web-сервер и модуль PHP на своем компьютере. Для получения уверенности в том, что можно начинать работу, выполните следующие действия:

1. Откройте текстовый редактор, например Блокнот Windows, и введите какой-нибудь текст (например, `Привет от Web-сервера!`). Это просто текст, а не код на языке PHP. Начиная со следующей строки, введите PHP-код:

   ```
   <?php echo "Это тест PHP" ?>
   ```

2. Сохраните введенный текст в обычном текстовом файле с расширением php (например, proba.php) в домашнем каталоге Web-сервера. По умолчанию таким каталогом для IIS является C:\Inetpub\wwwroot.

3. Откройте Web-браузер, например, Internet Explorer и в адресной строке введите **http://localhost/proba.php** (префикс **http://** можно не указывать). В результате в клиентской области браузера должен отобразиться результат интерпретации файла proba.php (рис. 4.1). Вместо **localhost** можно ввести сетевое имя вашего компьютера либо его десятичный IP-адрес. Если Web-сервер установлен на другом компьютере в локальной сети, к которой вы подключены, то следует использовать сетевое имя или IP-адрес того компьютера, а не вашего.

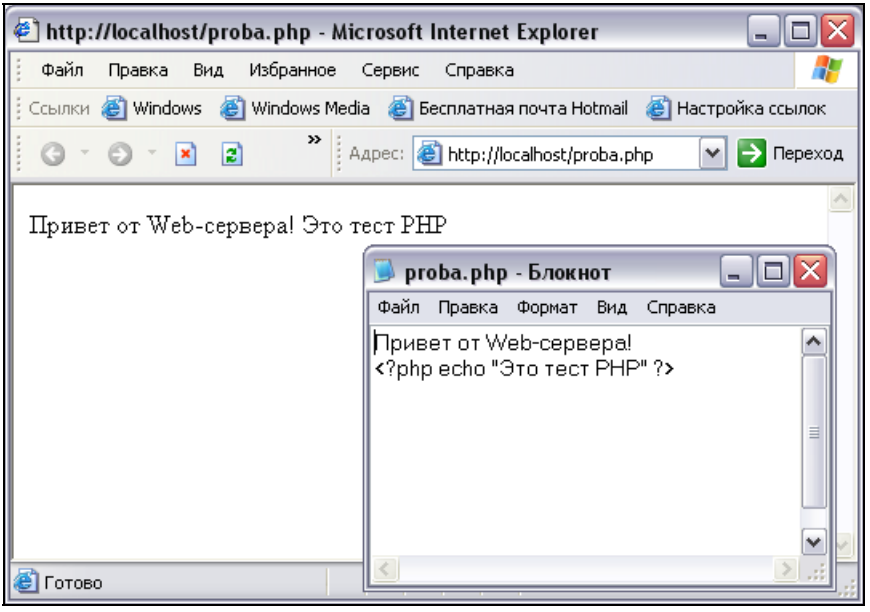


Рис. 4.1. Вид тестового примера в текстовом редакторе и браузере

Если вам потребуется изменить содержимое текущего PHP-файла (например, proba.php), то не следует выполнять команду меню браузера **Вид | Просмотр HTML-кода**, т. к. в этом случае вы увидите не исходное содержимое файла, а результат его интерпретации. Сценария на языке PHP там уже нет. В рассматриваемом примере это будет такой текст:

```
Привет от Web-сервера!Это тест PHP
```

Другими словами, помните, что содержимое исходного php-файла, хранящегося на сервере, может существенно отличаться от того, что загружено в браузер. Поэтому для коррекции открывайте в текстовом редакторе именно исходный файл. Чтобы изменения вступили в силу, сохраните файл под тем

же именем и в том же месте, а затем обновите содержимое окна браузера (перезагрузите страницу).

Содержимое php-файла может быть каким угодно: обычные тексты, HTML-теги, а также коды на языке PHP. Код на языке PHP должен быть заключен между дескрипторами `<?php` и `?>`. Он может занимать одну и более строк. В одном и том же php-файле может располагаться несколько фрагментов (секций) PHP-кода, перемежающихся блоками обычного текста и/или HTML-кода. Тогда каждый фрагмент PHP-кода должен окаймляться дескрипторами `<?php` и `?>`. Все содержимое этих дескрипторов будет выполнено PHP-интерпретатором на стороне сервера, а все, находящееся за их пределами, будет без изменений передано сервером браузеру клиента.

Некоторые программы на PHP могут иметь универсальный характер, позволяющий использовать их неоднократно и в различных проектах. В подобных случаях возможно сохранять фрагменты кода (определения функций, классов и др.) в отдельных файлах, а в конкретный сценарий включать их с помощью встроенной функции `include(имя_файла)`. В параметре указывается имя (возможно, с указанием пути или URL-адрес файла), содержимое которого необходимо вставить в место вызова данной функции `include()`. Вставляемый файл содержит PHP-код, который должен быть заключен в дескрипторы `<?php` и `?>`, если мы хотим, чтобы он выполнился, а не просто отобразился в браузере клиента. Имя этого файла может иметь произвольное расширение, но обычно используется inc, чтобы напомнить разработчику, что это — включаемый файл. Например, `include("mylib.inc")`.

Внимание

Файл, содержащий PHP-код, может иметь любое расширение, например, php, htm, html, inc и т. п. Так, большинство кодов Web-страниц сохраняются в файлах с расширением htm или html, но если они содержат внутри PHP-код, то он будет выполнен на стороне сервера.

Нередко требуется принудительно прекратить выполнение сценария (выйти из программы). Для этого можно использовать функцию `exit()` или `die()`, которая является псевдонимом для первой. При вызове функции `exit()` (или `die()`) допустимо задать параметр: строку сообщения, выводимую перед завершением сценария или целое число от 0 до 254 — так называемый *статус завершения*.

При разработке сценария можно использовать встроенные глобальные переменные и константы, доступные из любых сценариев, а также настройки модуля PHP, заданные в виде директив (*переменная=значение*) в конфигурационном файле php.ini. Работа с директивами файла php.ini рассмотрена в *приложении 2*.

Подробные сведения о PHP и, в частности, о его встроенных функциях можно найти в Интернете по адресу: **http://ru.php.net/download-docs.php**. Я настоятельно рекомендую вам загрузить на свой компьютер эту достаточно удобную справочную систему по PHP. Для пользователей Windows она представлена в CHM-формате (компилированном HTML-коде) и, следовательно, отображается в диалоговом окне с инструментами поиска и печати. Там вы найдете не только недостающие сведения по функциям и другим конструкциям языка PHP, но и по особенностям различных его версий. Будучи открытым продуктом и кодом, PHP постоянно и довольно быстро обновляется. Поэтому важно следить за обновлениями, по крайней мере, 3 раза в год. С этой целью посещайте сайт **www.php.net**.

4.2. Вывод данных

Начиная программировать на любом языке, важно уметь получать не только удручающие системные сообщения об ошибках, но и запланированные нами же сообщения, которые уведомляют, что программный код как-то понимается интерпретатором языка и даже выполняется. Поэтому важно сперва научиться выводить сообщения на экран монитора.

Вывод данных в окно (клиентскую область Web-браузера) с помощью PHP можно выполнить посредством оператора `echo`. Этот оператор позволяет вывести данные различных типов (числа, символьные строки и т. д.). Вы можете использовать его для вывода обычных текстовых строк, строк HTML-кода, а также данных других типов (чисел, массивов и др.). Посредством этого оператора можно выводить в окно браузера как сообщения, предусмотренные вашим приложением, так и отладочные (значения переменных, возвращаемые значения выражений).

Синтаксис оператора вывода `echo`:

`echo` *`элемент1`*`,` *`элемент2`*`, ...,` *`элементN`*

Данные (*`элемент1`*, *`элемент2`*, ..., *`элементN`*), подлежащие выводу, могут быть различных типов *(см. разд. 4.3)*. Например, это могут быть числа, строки различных символов и др. Все данные, перечисленные в списке, выводятся в окне браузера друг за другом, без пробелов и запятых.

Особо следует отметить вывод строковых данных, которые заключаются в двойные или одинарные кавычки. Если вы хотите вывести произвольную последовательность символов с помощью оператора `echo`, то ее следует заключить в кавычки. Внутри этих кавычек могут быть любые символы (буквы, цифры и т. п.), в том числе и имена переменных *(см. разд. 4.4)*. В случае

двойных кавычек строка символов будет интерпретироваться PHP. Если, например, в ней найдутся имена переменных, то в отображаемой строке эти имена переменных будут заменены их значениями. Более того, если в этой строке найдутся теги HTML (дескрипторы, заключенные в угловые скобки), то браузер отобразит этот HTML-код так, как он и должен это делать при восприятии HTML-документов. Если же вы хотите отобразить строку символов так, как вы ее написали, без какой бы ни было интерпретации, то ее следует заключить в одинарные кавычки. В листинге 4.1 приведен тестовый PHP-код, результат которого отображается в браузере, как показано на рис. 4.2.

Листинг 4.1. Пример использования оператора echo

```
<?php
  $x=2006;   // присвоение значения переменной
  echo "Привет", " всем!";
  echo "<br>";
  echo "На дворе", "$x год.","<br>Желаю вам счастья!";
  echo "<h1>Пример вывода в PHP</h1>";
  echo 'Использована переменная $x';
?>
```

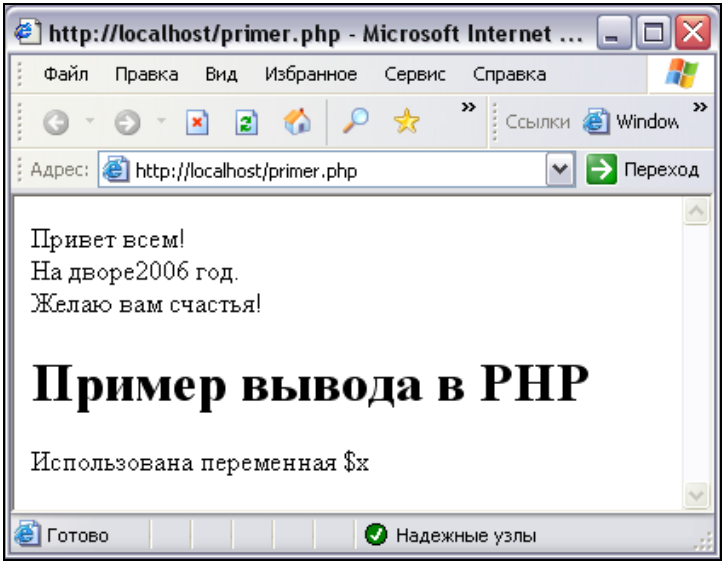


Рис. 4.2. Пример вывода с помощью оператора `echo`

Здесь использован оператор присваивания значения переменной `$x=2006`. Подробнее о переменных и присваивании им значений рассказано в *разд. 4.4*. Первое вхождение в программу оператора `echo` выводит две символьные строки: вторая следом (без пробелов) за первой. Второй оператор `echo` выводит HTML-тег `<br>`, который в браузере проявляется переводом строки. Третий оператор `echo` выводит подряд три строки, подставляя во второй из них значение переменной `$x`, а в третьей — тег перевода строки `<br>` и текст "Желаю вам счастья!". Четвертый оператор `echo` выводит строку символов, заключенных в теги заголовка 1-го уровня. Наконец, пятый оператор `echo` выводит строку символов, заключенную в одинарные кавычки. Именно поэтому все содержимое этой строки не интерпретируется, а выводится как есть (вместо переменной `$x` не подставляется значение, а просто выводится ее имя).

Внимание

Способ отображения символьных строк существенно зависит от того, в двойные или одинарные кавычки заключены символы строки.

4.3. Типы данных

PHP относится к языкам с так называемыми *свободными типами данных*. Это означает, что допустимо применение операций к данным различных типов, а также присвоение некоторой переменной данных то одного, то другого типа. Этой возможностью следует пользоваться осторожно во избежание недоразумений.

Тип данных — некоторое ограничение на множество возможных значений. Если мы каким-либо образом выделяем (ограничиваем) некоторое множество значений, то тем самым определяем тип данных.

В табл. 4.1 приведены основные (примитивные) типы данных, которые определены в PHP.

Внимание

Ключевые слова `true`, `false` и `null` могут использоваться в выражениях PHP в любом регистре.

В выражениях с операторами могут использоваться данные различных типов. Например, в арифметическом выражении могут оказаться данные не только числового, но и строкового или логического типа. В таких случаях интерпретатор PHP автоматически приводит данные к нужному типу.

Таблица 4.1. *Типы данных в PHP*

Тип данных		Примеры	Описание значений

Строковый или символьный (`String`)		`"Привет всем!"`
		`"д.т. 123-4567"`
		`'Сегодня 24.01.2006г.'`
			Последовательность символов, заключенная в кавычки, двойные или одинарные

Числовой	Целочисленный (`integer`)	`-567`
		`25`
		`0`
			Число, последовательность цифр, перед которой может быть указан знак числа (`+` или `-`); перед положительными числами не обязательно ставить знак `+`. Число записывается без кавычек

	С плавающей точкой (`float`)	`5.47`
		`12.5E2`
		`35E-3`
			Число с дробной частью. Целая и дробная части чисел разделяются точкой.
			В экспоненциальной форме символ `E` используется для обозначения 10, за которым следует число, указывающее степень. Например, запись `3E2` означает 3×10^2, т. е. 300

Логический (булевый, `Boolean`)		`true` `false`	Этот тип имеет два значения: `true` (истина, да) или `false` (ложь, нет)
`Null`		`null`	Этот тип данных имеет одно значение — `null`, указывающее на отсутствие какого бы то ни было значения
Массив (`Array`)			Этот тип данных имеет одно множество значений, которые могут быть различных типов
Объект (`Object`)			Программный объект, определяемый своими свойствами
Ресурс (`Resourse`)			Например, открытый файл, графические данные и др.

Например, в выражении с арифметическим оператором сложения `+` операнды могут быть как строковыми, так и числовыми:

```
5+3              // результат: 8
"5"+"3"          // результат: 8
```

```
"5"+3            // результат: 8
"5рублей"+3      // результат: 8
"Доход 5"+3      // результат: 3
"Привет"+3       // результат: 3
"Привет"+"всем"  // результат: 0
```

Если строка не содержит в качестве своих первых символов цифры (точнее, число, перед которым может быть указан знак), то в арифметических выражениях она преобразуется в число 0.

Примечание

В JavaScript оператор `+` применительно к числам является оператором сложения, а применительно к строкам — оператором конкатенации (склейки). В выражениях с этим оператором и с данными строкового и числового типа последние в JavaScript приводятся к строковому типу. Так, выражение `"5"+3` даст в результате `"53"`, а не 8, как в PHP.

Данные различных типов могут участвовать в логических операциях и операциях сравнения, которые возвращают логическое значение `true` или `false`. В таких случаях интерпретатор PHP автоматически приводит данные нелогического типа к логическому. При этом следующие значения преобразуются в `false`:

- строка `"FALSE"`, независимо от регистра;
- пустая строка `""` и строка `"0"`;
- число 0, целое или с плавающей точкой;
- `Null`;
- пустой массив (количество элементов равно 0);
- пустой объект (количество элементов равно 0).

Другие значения преобразуются в `true`.

Значения логического типа могут участвовать в арифметических операциях. В таких случаях значение `false` преобразуется в 0, а `true` — в 1. Например, выражение `true + true` возвращает 2, а `true + false` — 1.

Иногда требуется данные некоторого типа привести к определенному типу, чтобы гарантированно обеспечить корректность последующих операций над ними, не полагаясь на возможности автоматического преобразования типов. Для приведения данных к заданному типу служат специальные операторы — имя типа в круглых скобках, записываемое перед данными, которые необходимо преобразовать:

- `(int)` или `(integer)` — приведение к целочисленному типу;
- `(float)`, `(double)` или `(real)` — приведение к числовому типу с плавающей точкой;

- `(string)` — приведение к строковому типу;
- `(bool)` или `(boolean)` — приведение к логическому (булевому) типу;
- `(array)` — приведение к типу "массив";
- `(object)` — приведение к типу "объект".

Примеры:

```
(int) 12.65            // результат: 12
(float) "12.65 руб."   // результат: 12.65
(bool) "12.65 руб."    // результат: true
```

Перечисленные выше операторы преобразования типов применимы и к переменным, и к вызовам функций, о чем будет рассказано в следующих разделах.

4.4. Переменные и оператор присваивания

4.4.1. Имена переменных

Переменная является контейнером для хранения данных. Данные, сохраняемые в переменной, называются *значениями* этой переменной. Переменная имеет имя (идентификатор) — последовательность букв, цифр, символов подчеркивания без пробелов и знаков препинания, начинающаяся обязательно с символа доллара (`$`), за которым должна следовать буква или символ подчеркивания. Иначе говоря, в имени переменной первым символом является `$`, а вторым — не цифры.

Примеры правильных имен переменных: `$myName`, `$my_adress`, `$_x`, `$tel123_4567`, `$var`, `$Сумма`.

Примеры неправильных имен переменных: `$512group`, `$my adress`, `tel:412 3456`, `$Сумма$`.

По существу префикс `$` в имени переменной указывает интерпретатору, что он имеет дело с переменной, а не чем-либо иным. Поэтому в имени переменной можно использовать и ключевые слова языка, такие как `if`, `else`, `while` и т. д.

Внимание

PHP является регистрозависимым языком относительно имен переменных и констант. Это означает, что изменение регистра символов (с прописных на строчные и наоборот) в имени переменной приводит к другой переменной. Например, `$variable`, `$Variable` и `$vaRiabLe` — различные переменные. Однако ключевые слова языка могут использоваться в любом регистре. В отличие от PHP, язык JavaScript является полностью регистрозависимым.

Примечание

В JavaScript символ $ может встречаться в имени переменной несколько раз и в любом месте или не использоваться совсем. В именах переменных PHP символ $ обязан быть первым. Повторное его использование возможно, но только сразу же после первого. Однако в этом случае он выполняет особую роль, которая будет рассмотрена в *разд. 4.4.3*.

4.4.2. Создание переменных

Переменная создается при первом ее упоминании в программном коде с использованием присваивания ей некоторого значения. Оператор присваивания обозначается знаком равенства (=). Слева от него указывается имя переменной, а справа — выражение языка, возвращающее в результате вычисления некоторое значение. В простейшем случае справа от символа оператора присваивания указывается простое значение — число, строка символов в кавычках, ключевое слово, обозначающее специальное значение некоторых типов (true, false, null).

Создать переменную можно с помощью оператора присваивания (=):

```
$var="Привет всем";   /* переменной $var присвоено строковое значение
                         "Привет всем" */
$x=25.2E2;    // переменной $x присвоено числовое значение 2520
$x=true;      // переменной $x присвоено логическое значение true
```

Одной и той же переменной можно присвоить то одно, то другое значение. Тип переменной определяется типом значения, которое она в данный момент имеет (содержит). Так, в рассмотренном выше примере переменная $x сначала имела значение числового типа, а затем — логического.

Не следует путать оператор присваивания с отношением равенства и соответствующей операцией сравнения. Выражение с оператором = интерпретатор вычисляет следующим образом: переменной слева от него присваивается значение, расположенное справа от него.

Значение одной переменной можно присвоить другой переменной:

```
$x=5;
$y=$x;    // переменная $y имеет значение 5
```

Переменные могут участвовать в выражениях с операторами:

```
$x=5;
$y=$x+2;  // переменная $y имеет значение 7
```

Когда мы пишем $y=$x, то тем самым выражаем следующее: значение переменной $x присвоить переменной $y. Это так называемое *присвоение по значению*,

при котором значение одной переменной копируется в другую переменную. Однако кроме этого способа в PHP возможно организовать *присвоение по ссылке*, при котором копирование значения не происходит, а новая переменная становится псевдонимом (еще одним именем) для той переменной, на которую она ссылается. Для присвоения значения по ссылке используется символ `&` (амперсанд). Так, в выражении `$x=&$y` переменная `$x` ссылается на переменную `$y`. При этом обращение к переменной `$x` возвращает значение переменной `$y`, а изменение значения `$x` отражается на значении `$y`, как и, наоборот, изменение значения `$y` вызывает изменение значения `$x`:

```
<?php
  $x="Саша";
  $y="Маша";
  $x=&$y;       // значение $x равно "Маша"
  $x="Петя";    // значения $x и &y равны "Петя"
  /* Вывод значений */
  echo '$x=',$x;
  echo "<br>";
  echo '$y=',$y;
?>
```

Примечание

Присвоение значений переменным по ссылке происходит быстрее, чем по значению, и это становится заметным, например, при установке значений элементам больших массивов с помощью оператора цикла.

У существующей переменной можно удалить текущее значение, т. е. сделать ее без какого бы то ни было значения (очистка переменной и занимаемой ею памяти). При этом переменная продолжает существовать. С этой целью используется функция `unset(`*имя_переменной*`)`:

```
$x=5;
unset($x);    // удаление значения переменной $x
```

В действительности в результате выполнения функции `unset($x)` переменная `$x` приобретает значение `null`.

Примечание

В данной главе мы еще не рассматривали функции как таковые. Пока вы можете считать, что функция (точнее ее вызов в программе) задается именем, за которым следует пара круглых скобок. В скобках, если это требуется для данной функции, указываются параметры — значения, передаваемые функции из внешнего программного кода. Функция обычно (но не обязательно) что-то воз-

вращает и в этом случае ее можно использовать в выражениях языка. Если вы уже знакомы с каким-либо языком программирования, например JavaScript *(см. главу 2)*, то данное примечание для вас излишне.

Как уже отмечалось, тип переменной определяется типом значения, которое она имеет в текущий момент времени. Однако в PHP существуют операторы приведения типов *(см. разд. 4.3)*. Обычно они используются, чтобы обеспечить надлежащее преобразование данных, в результате которого можно рассчитывать на корректное применение тех или иных операций. Напомню, что и без этих операторов интерпретатор PHP, анализируя выражение, преобразует данные к некоторому подобающему в данной ситуации типу. Однако его "интеллектуальные потуги" иногда могут не соответствовать ожиданиям автора программы, и тогда автор может форсировать приведение типов по своему усмотрению. Вот пример:

```
$x="5.3 рублей";   // строка символов "5.3 рублей"
$y=(int)$x +7;     // целое число 12
$y=(float)$x+7;    // число с плавающей точкой 12.3
$z=(bool)$y;       // логическое значение true
```

4.4.3. Отображение значений переменных

Значения переменных можно вывести на экран с помощью оператора `echo`. Если вы работаете с Web-браузером, то этот оператор направляет вывод значений в клиентскую область браузера. При выполнении примеров и других экспериментальных кодов на языке PHP, а также при отладке ваших реальных сценариев очень полезно организовать вывод промежуточных и окончательных результатов на экран. Для этого мы используем оператор `echo` *(см. разд. 4.2)*. Вывод текущих значений переменных можно сделать несколькими способами. Приведу лишь наиболее употребительные:

```
echo  $var1, $var2, ..., $varN;  // вывод значений нескольких переменных
echo "Переменные: $var1, $var2, ..., $varN";
echo 'Переменные: $var1, $var2, ..., $varN';
```

Рассмотрим эти конструкции отдельно.

- В первом случае в клиентскую область браузера выводятся значения переменных, перечисленные в операторе `echo` через запятую. При отображении значения переменных следуют друг за другом без пробелов (запятые между именами переменных не отображаются).
- Во втором случае выводится символьная строка (заключенная в двойные кавычки), в которой упомянуты переменные. Интерпретатор PHP выведет строку, заменив эти переменные их значениями.

- В третьем случае выводится строка так, как есть. Иначе говоря, выводится все, что указано внутри одинарных кавычек без подстановок значений переменных.

Замечу попутно, что в строку символов, подлежащую выводу в окно браузера, можно вставлять теги HTML, которые позволяют отформатировать выводимый текст. В листинге 4.2 представлен PHP-код, который иллюстрирует приведенные выше замечания, а результат его выполнения, отображенный в браузере Internet Explorer, показан на рис. 4.3.

Листинг 4.2. Пример вывода переменных в окно браузера

```
<?php
  $var1="Привет";
  $var2=" всем!";
  echo  $var1, $var2; // вывод значений нескольких переменных
  echo "<br>Переменные: $var1, $var2<br>";
  echo 'Переменные: $var1, $var2';
?>
```

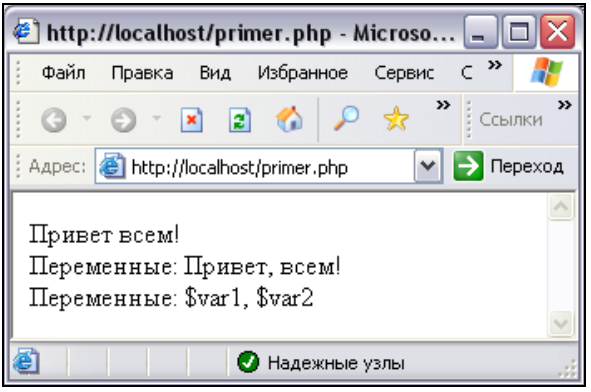


Рис. 4.3. Пример отображения значений переменных

Внимание

Вывод с помощью оператора `echo` строки, в которой указаны переменные, существенно зависит от того, в какие кавычки заключена эта строка символов — двойные или одинарные. В случае двойных кавычек переменные вычисляются, а вместо их имен в строку подставляются значения. В случае одинарных кавычек содержимое строки не анализируется и не вычисляется интерпретатором PHP, а выводится как есть.

Значения некоторых типов отображаются оператором `echo` не всегда так, как они были заданы в коде программы. Так, логическое значение `true` отображается как 1, а `false` — не отображается вовсе; числовые значения отображаются в экспоненциальной или неэкспоненциальной форме в зависимости от их величины (рис. 4.4). Вместе с тем вы можете отформатировать выводимые сообщения по своему усмотрению, но это будет рассказано в *разд. 4.7*.

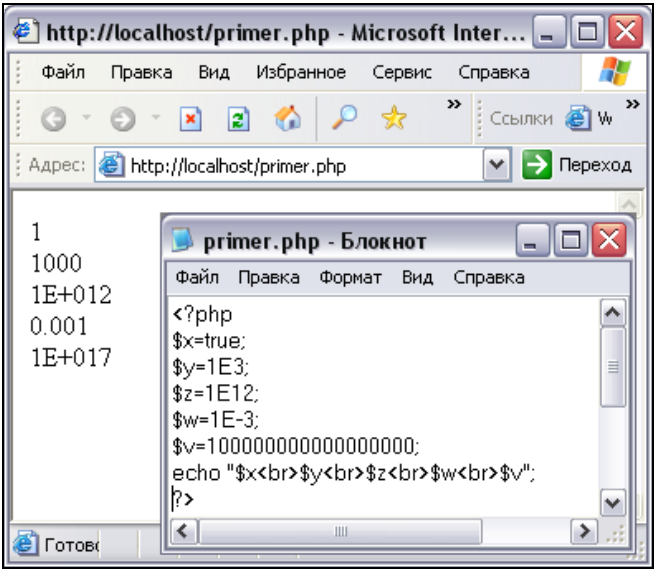


Рис. 4.4. Отображение логического и числовых значений с помощью оператора `echo`

Вывод значений элементов массива в составе текстовых строк имеет некоторую особенность: элементы массива необходимо заключать в фигурные скобки (подробнее об этом *см. в разд. 4.10.3*).

Для вывода более подробной информации о переменных, которая может понадобиться при отладке программы, служит функция `var_dump(`*`список_переменных`*`)`. В списке переменных указывается одно или несколько имен переменных, разделенных запятыми. Эта функция ничего не возвращает. На рис. 4.5 показан пример использования `var_dump()` для переменных различных типов. Так, в случае строковой переменной выводится ее тип, длина значения и само значение, для числовых переменных выводится тип и значение. Данная функция особенно полезна при выводе информации о массивах. В этом случае выводится тип переменной (`array`), количество элементов массива и список сведений об элементах вида *`индекс`*`=>` *`элемент`*. Индекс может быть как

числовым, так и символьным в зависимости от того, как он был задан в определении массива. Подробнее о массивах *см. в разд. 4.10*.

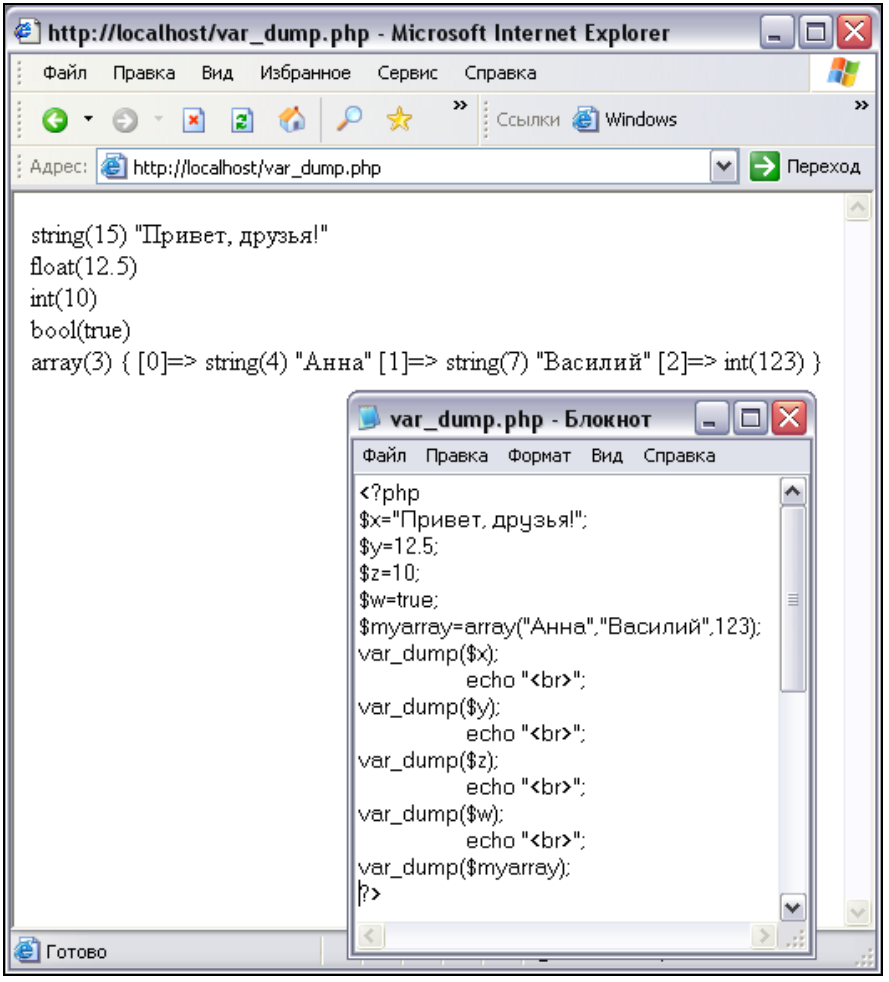


Рис. 4.5. Вывод информации о переменных с помощью функции `var_dump()`

Менее информативной, чем `var_dump()`, функцией вывода сведений о переменных является `print_r(`*`список_переменных`*`)`. Для переменных типа "массив" эта функция выводит список вида *`индекс=> элемент`* без указания типов и других характеристик элементов. Для переменных других типов выводятся только их значения. Пример использования этой функции можно найти в *разд. 4.7.3*.

4.4.4. Переменные переменные

Иногда может понадобиться интерпретировать строковое значение как имя переменной. Другими словами, мы хотим присвоить некоторой переменной в качестве значения имя другой переменной. Так можно создавать переменные динамически, т. е. программно задавать переменные, которые необходимы в данный момент вычислительного процесса. Другими словами, переменная может содержать другую переменную, называемую "переменной переменной". Для создания переменной переменной используется двойной символ `$`. Так, запись `$$x` означает переменную, имя которой указано в качестве значения переменной `$x`.

На рис. 4.6 показан код программы в окне текстового редактора и результат его выполнения в окне браузера. Здесь переменная `$x` содержит строковое значение `"Девочка"`. Выражение `$$x="Маша"` означает, что переменной, имя которой содержится в переменной `$x`, присваивается значение `"Маша"`. Таким образом, создается переменная `$Девочка` и ей присваивается значение `"Маша"`. С помощью оператора `echo` выводится строка, содержащая значение переменной `$x` и динамически созданной переменной `$Девочка`.

Оператор `echo "$$x"` выведет строку `"$Девочка"`, оператор `echo "${$x}"` — строку `"Маша"`. Здесь использовались фигурные скобки, чтобы указать интерпретатору, что требуется вывести, а именно значение переменной `$Девочка`: значение переменной {значение переменной `x`}.

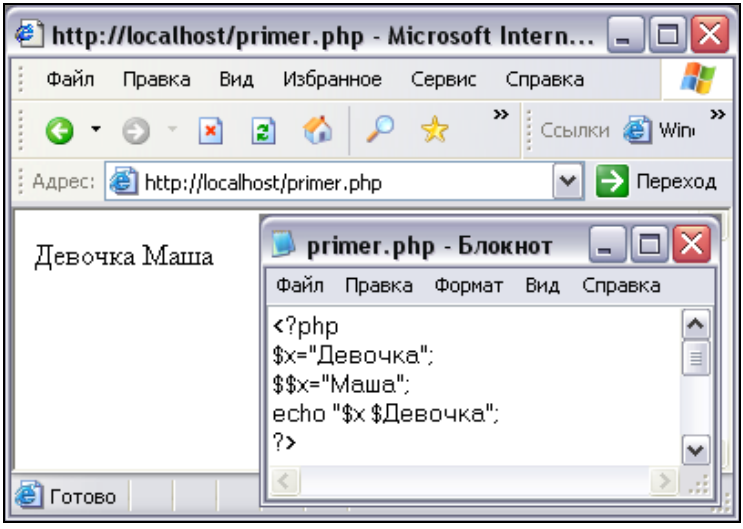


Рис. 4.6. Пример использования переменных переменных

4.4.5. Область действия переменных

Областью действия (видимости, доступности) переменной является тот программный код из одного и того же файла, в котором она была определена. При этом она также видна и в PHP-кодах, включаемых в данный из внешнего файла с помощью функции `include()`. Однако внутри кода функций такие переменные не видны, если не предпринять специальных мер.

В следующем примере переменная `$x` видна во всем программном коде, включая и код из файла mylib.php. Если код из этого файла не изменяет значение переменной `$x`, то выведенное значение будет равно 9.

```
<?php
  $x=5;
?>
```

Далее выводим переменную `$x`:

```
<?php
  include("mylib.php"); // включение файла
  $x=$x+4;
  echo "$x";            // вывод значения переменной $x
?>
```

Таким образом, переменные в PHP были бы глобальными в полном смысле этого слова, если бы не одно обстоятельство. Дело в том, что программный код может содержать определения и вызовы функций. Если созданная во внешнем коде переменная используется еще и в коде функции, то там она будет не глобальной, как ожидается, а локальной. Это означает, что значение этой переменной, заданное во внешнем коде, не видно внутри кода функции. С другой стороны, изменения значений этой переменной внутри функции не отразятся на этой же переменной во внешнем коде. Чтобы сделать переменные, созданные во внешнем коде, доступными и внутри кода функций (т. е. обеспечить им настоящую глобальность), они должны быть объявлены глобальными в коде функций с помощью ключевого слова `global`. Кроме того, переменные внутри функций могут быть объявлены как статические (`static`). Статическая переменная является локальной, но сохраняет свое значение и при следующем вызове функции, где она была определена, данное значение может быть использовано. Примером такой функции является `strtok()` — *см. разд. 4.7.3*.

Примечание

В языках JavaScript и C глобальные переменные автоматически доступны и внутри функций, если только они не были определены там как локальные.

То есть глобальные переменные видны везде. Например, в JavaScript локализация переменной в функции задается посредством ключевого слова `var`. Если, например, переменная `myvar` является глобальной, то чтобы в коде функции использовать это же имя для локальной переменной, достаточно там написать выражение `var myvar`.

Примечание

В PHP имеется множество предопределенных переменных типа "массив". Эти переменные глобальны в полном смысле этого слова и содержат сведения о Web-сервере, окружении и данных, которые передал серверу пользователь.

Примечание

Базовый оператор присвоения значений обозначается знаком равенства. Однако допускаются и специальные операторы присвоения значений, которые в ряде случаев оказываются более удобными. О них рассказывается в *разд. 4.6.4*.

4.4.6. Проверка существования и типов переменных

Иногда требуется проверить, определена ли та или иная переменная. Для этого служит функция

`isset(`*имя_переменной1*`, `*имя_переменной2*`, ..., `*имя_переменнойN*`)`

Если все переменные в списке существуют, то функция возвращает `true`, в противном случае — `false`.

Чтобы выяснить, является ли значение переменной пустым, используется функция

`empty(`*имя_переменной*`)`

Эта функция возвращает `true`, если значение переменной равно 0 (целое число), `""` (пустая строка), `"0"` (строка с 0), `false`, `null`, переменная не объявлена или является пустым массивом. В противном случае возвращается `false`.

Примечание

Начиная с PHP 5, объект без свойств не считается пустым с точки зрения функции `empty()`.

Чтобы проверить тип переменной, используются следующие функции:

- `is_string(`*имя_переменной*`)` — возвращает `true`, если переменная является строковой, и `false` — в противном случае;

- `is_int(`*имя_переменной*`)` — возвращает `true`, если переменная является целочисленной, и `false` — в противном случае;
- `is_float(`*имя_переменной*`)` — возвращает `true`, если переменная является числовой с плавающей точкой, и `false` — в противном случае;
- `is_numeric(`*имя_переменной*`)` — возвращает `true`, если переменная является числовой (типа `integer` или `float`), а также строкой, содержащей только числа. В противном случае возвращается `false`;
- `is_null(`*имя_переменной*`)` — возвращает `true`, если переменная имеет значение `null`, и `false` — в противном случае;
- `is_array(`*имя_переменной*`)` — возвращает `true`, если переменная является массивом (имеет тип `array`), и `false` — в противном случае.

4.5. Константы

Константы похожи на переменные, поскольку имеют имена (идентификаторы) и значения. Значения присваиваются константам однажды в коде программы и после не могут быть изменены. Константы имеют следующие особенности:

- имя константы образуется так же, как и имя переменной, за исключением того, что первым символом не должен быть `$`. Этот символ указывает, что за ним следует имя переменной, а мы сейчас имеем дело с константой. Поскольку имени константы не должен предшествовать символ `$`, в качестве имени константы нельзя использовать ключевые слова PHP, такие как `if`, `else`, `while`, `echo`, `default` и т. п.;
- значения констант могут принадлежать только типам `string`, `integer`, `float` и `boolean`;
- создание констант (объявление имени и присвоение значений) обеспечивается специальной функцией `define("`*имя_константы*`", `*значение*`)`. Например, `define("MYNAME","Дунаев Вадим Вячеславович")`. Обычный для переменных оператор = присваивания значений здесь не годится.

Константы полезны, если вы хотите обезопасить себя от случайного изменения в программе данных, которые должны быть постоянными. Если же вам все же потребуется их изменить, то это придется делать в одном месте, а именно там, где вы их однажды определили с помощью функции `define()`. Кроме того, вы можете создавать один и тот же программный код, который должен работать с различными наборами исходных данных, адаптированных, например, к конкретному пользователю, среде выполнения, варианту реализации и т. п. В этом случае использование констант поможет улучшить

технологичность вашего проекта. На рис. 4.7 показан пример определения и вывода констант.

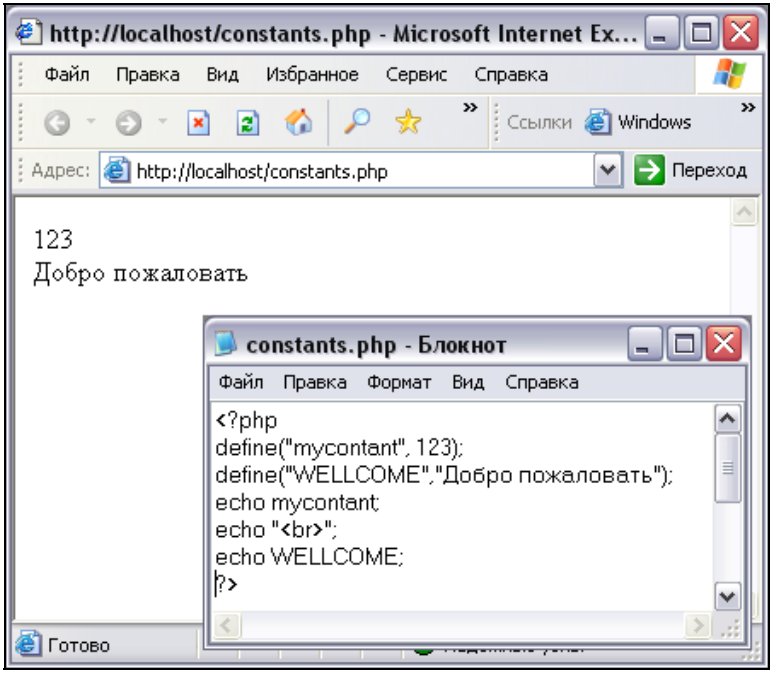


Рис. 4.7. Пример определения и вывода констант

Примечание

В PHP имеется множество предопределенных констант. Например, константа `__FILE__` (окаймляется двойными символами подчеркивания) содержит полное имя текущего PHP-файла, а константа `__LINE__` содержит номер текущей строки в PHP-файле. Математические константы приведены в *разд. 4.8.2*.

Иногда требуется проверить, определена ли та или иная константа. Для этого существует функция

`defined(`*имя_константы1*`, `*имя_ константы2*`, ..., `*имя_ константыN*`)`

Если все константы из списка существуют, то функция возвращает `true`, в противном случае — `false`.

Примечание

Аналогичная функция для проверки существования переменных — `isset()`.

4.6. Операторы

Операторы предназначены для составления выражений. Оператор применяется к одному или двум данным, которые в этом случае называются *операндами*. Например, оператор сложения применяется к двум операндам, а оператор логического отрицания — к одному операнду. *Элементарное выражение*, состоящее из операндов и оператора, выполняется интерпретатором и может иметь некоторое значение. В этом случае говорят, что оператор возвращает значение. Оператор имеет тип, совпадающий с типом возвращаемого им значения. Элементарное выражение с оператором и операндами, возвращающее значение, можно присвоить переменной.

В предыдущем разделе мы рассмотрели оператор присваивания. Сейчас мы обратимся к другим операторам, предназначенным для работы с данными различных типов.

4.6.1. Комментарии

Операторы комментария позволяют выделить фрагмент кода, который не выполняется интерпретатором, а служит лишь для пояснений содержания программы.

В PHP допустимы три вида оператора комментария:

- `//` — одна строка символов, расположенная справа от этого оператора, считается комментарием;
- `#` — действует так же, как и оператор `//`;
- `/*...*/` — все, что заключено между `/*` и `*/`, считается комментарием; с помощью этого оператора можно выделить несколько строк в качестве комментария.

4.6.2. Арифметические операторы

Арифметические операторы, такие как сложение, умножение и т. д., в PHP предназначены для манипулирования числовыми данными, хотя могут применяться к данным любых типов. В табл. 4.2 приведены арифметические операторы.

Таблица 4.2. *Арифметические операторы*

Оператор	Название	Пример
+	Сложение	`$X+$Y`
-	Вычитание	`$X-$Y`
*	Умножение	`$X*$Y`

Таблица 4.2 *(окончание)*

Оператор	Название	Пример
/	Деление	$X/$Y
%	Деление по модулю	$X%$Y
++	Увеличение на 1	$X++ ++$X
--	Уменьшение на 1	$Y-- --$X

Операторы ++ и -- называют соответственно *инкрементом* и *декрементом*.

Символ - используется не только как бинарный (для двух операндов) оператор вычитания, но и как унарный (для одного операнда) оператор отрицания для указания того, что число является отрицательным.

Однако формально ничто не мешает нам применить эти операторы к данным нечисловых типов. При этом все нечисловые операнды автоматически преобразуются в числа. Например, в случае строковых данных интерпретатор пытается привести их к числовому типу. Если префиксом строки является число, то эта строка будет преобразована в соответствующее число, в противном случае она будет преобразована в число 0. Логические значения преобразуются в 1 и 0 соответственно для `true` и `false`. Подробнее об этом *см. в разд. 4.3*.

Примеры:

```
5+3              // результат: 8
"5"+"3"          // результат: 8
"5"+3            // результат: 8
"5рублей"+3      // результат: 8
"Доход 5"+3      // результат: 3
"Привет"+3       // результат: 3
"Привет"+"всем"  // результат: 0
$x=8;
$y=3;
$x/$y;           // результат: 2.66666666667
$x%$y;           // результат: 2
$x++;  /* результат: значение переменной $x становится  равным 9, но выражение возвращает 8 */
++$x;  /* результат: значение переменной $x становится равным 9, и выражение возвращает 9 */
```

Примечание

В JavaScript в выражениях с оператором сложения + данных строкового и числового типа последние приводятся к строковому, а не числовому типу. Поэтому выражение "Вася"+5 даст в результате "Вася5", а не 5, как в PHP. С другой стороны, в PHP выражение "5"+4 вернет 9, а в JavaScript — "54".

4.6.3. Строковый оператор

Две строки можно объединить (склеить) с помощью операции конкатенации, обозначаемой точкой. Применение этой операции к двум строкам дает в результате строку, полученную путем приписывания справа второй строки к первой. Например, выражение "Саша"."Маша" возвращает в результате "СашаМаша".

Пример:

```
$x="Привет";
$y="всем";
echo $x." ".$y;
```

Здесь склеиваются три строки: значение переменной $x, строка с пробелом и значение переменной $y.

Примечание

В JavaScript оператор конкатенации обозначается так же, как и оператор сложения чисел (+).

4.6.4. Дополнительные операторы присваивания

Кроме обычного оператора присваивания = имеются еще шесть дополнительных операторов, сочетающих в себе действия обычного оператора присваивания и арифметических операторов (табл. 4.3).

Таблица 4.3. *Операторы присваивания*

Оператор	Пример	Эквивалентное выражение
+=	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

Пример:

```
$x="Вася"
$x.=", привет"   // "Вася, привет"
$x=5; $x*=7      // 35
```

4.6.5. Операторы сравнения

Результатом вычисления элементарного выражения, содержащего оператор сравнения и операнды (сравниваемые данные), является логическое значение, т. е. `true` или `false`. Операторы сравнения приведены в табл. 4.4

Таблица 4.4. *Операторы сравнения*

Оператор	Название	Пример
==	Равно	$X==$Y
===	Тождественно равно	$X===$Y
!=	Не равно	$X!=$Y
<>	Не равно	$X<>$Y
!===	Не равно тождественно	$X!===$Y
>	Больше, чем	$X>$Y
>=	Больше или равно (не меньше)	$X>=$Y
<	Меньше, чем	$X<$Y
<=	Меньше или равно (не больше)	$X<=$Y

Внимание

Оператор "равно" записывается с помощью двух символов == без пробелов между ними.

Сравнивать можно данные различных типов. Что касается оператора равенства (неравенства), то допустимы две его разновидности:

- гибкий оператор равенства (==). Выражение возвращает `true`, если операнды равны. В противном случае возвращается `false`. При этом типы сравниваемых данных могут различаться;
- жесткий оператор равенства (===). Выражение возвращает `true`, если операнды не только равны, но имеют еще и одинаковые типы. В противном случае возвращается `false`.

Примеры:

```
"5.3" == 5.3            // возвращает true
"5.3" === 5.3           // возвращает false
"1" == true             // возвращает true
1 == true               // возвращает true
"1" == "true"           // возвращает false
1 == "true"             // возвращает false
"0" == "false"          // возвращает false
"0" == false            // возвращает true
0 == "false"            // возвращает true
0 == false              // возвращает true
0 === "false"           // возвращает false
null==null              // возвращает true
null===null             // возвращает true
"abcd" > " abcd"        // возвращает true
"abc" < "abcd"          // возвращает true
"235ab" < "abcdxyz"     // возвращает true
"235xyz" < "abc"        // возвращает true
```

Внимание

При практическом программировании следует обратить особое внимание на сравнение логических значений `true` и `false` с данными других типов, таких как пустые строки, `null`, `"0"`, `0` и т. д. Многие функции возвращают `false` тогда, когда что-то не так (например, возникла ошибка преобразования данных, произошел выход за границы массива). Подобные ситуации обычно каким-либо образом обрабатываются в программе, и при этом чаще всего проверяется условие, а не равен ли результат `false`. Здесь важно корректно сформулировать проверяемое условие.

4.6.6. Логические операторы

Логические данные, обычно получаемые с помощью элементарных выражений, содержащих операторы сравнения, можно объединять в более сложные выражения. Для этого используются логические (булевы) операторы — логические И, ИЛИ, исключающее ИЛИ, а также оператор отрицания НЕ (табл. 4.5). Выражения с логическими операторами возвращают значение `true` или `false`.

Таблица 4.5. *Логические операторы*

Оператор	Название	Пример
`!`	Отрицание (НЕ)	`!$X`
`&&` `and`	И	`$X&&$Y`
`||` `or`	ИЛИ	`$X\|\|$Y` `$X or $Y`
`xor`	Исключающее ИЛИ	`$X xor $Y`

Оператор отрицания `!` применяется к одному операнду, изменяя его значение на противоположное: если `$X` имеет значение `true`, то `!$X` возвращает значение `false`, и наоборот, если `X` имеет значение `false`, то `!X` возвращает значение `true`. Логическим И и ИЛИ соответствуют по два оператора. Они действуют одинаково, но имеют различные приоритеты, определяющие порядок их вычисления в сложных выражениях. Приоритеты логических операторов определяются их положением в последовательности: `!`, `&&`, `||`, `and`, `xor`, `or`.

В табл. 4.6 указано, какие значения возвращают операторы `and`, `or` и `xor` при различных логических значениях двух операндов.

Таблица 4.6. *Примеры использования операторов* `and`, `or` *и* `xor`

$X	**$Y**	**$X && $Y**	**$X || $Y**	**$X xor $Y**
true	true	true	True	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

Операторы `&&` и `||` еще называют *логическим умножением* и *логическим сложением* соответственно.

Примеры:

```
$x=false || 2*2==4;         // значение переменной x равно true
$x=5<2 || "abcd" <="xy";    // значение переменной x равно true
$y=!$x;                     // значение переменной y равно false
$z=$x && $y;                // значение переменной z равно false
$z=$x || $y;                // значение переменной z равно true
$x=true; $y=false;
$z=$x xor !$y;              // значение переменной z равно false
```

Сложные логические выражения, состоящие из нескольких более простых, соединенных операторами И, ИЛИ и исключающего ИЛИ, выполняются по так называемому *принципу короткой обработки*. Дело в том, что значение всего выражения можно определить, вычислив лишь одно или несколько более простых выражений, не вычисляя остальные. Данное обстоятельство требует особого внимания при тестировании сложных логических выражений. Так, если какая-нибудь составная часть выражения содержит ошибку, то она может остаться невыявленной, поскольку эта часть выражения просто не выполнялась при тестировании.

4.6.7. Побитовые операторы

Побитовые (поразрядные) операторы применяются к целочисленным значениям и возвращают целочисленные значения. При их выполнении операнды предварительно приводятся к двоичной форме представления, в которой число является последовательностью из нулей и единиц длиной 32. Эти нули и единицы называются *двоичными разрядами* или *битами*. Далее производится некоторое действие над битами, в результате которого получается новая последовательность битов. В конце концов, эта последовательность битов преобразуется к обычному целому числу — результату побитового оператора. В табл. 4.7 приведен список побитовых операторов.

Таблица 4.7. *Побитовые операторы*

Оператор	Название	Левый операнд	Правый операнд
`&`	Побитовое И	Целое число	Целое число
`|`	Побитовое ИЛИ	Целое число	Целое число
`^`	Побитовое исключающее ИЛИ	Целое число	Целое число
`~`	Побитовое НЕ	—	Целое число
`<<`	Смещение влево	Целое число	Количество битов, на которое производится смещение
`>>`	Смещение вправо	Целое число	Количество битов, на которое производится смещение

Операторы `&`, `|`, `^` и `~` напоминают логические операторы, но их область действия — биты, а не логические значения. Оператор `~` изменяет значение бита на противоположное: 0 на 1, а 1 — на 0. Действие побитовых операторов такое же, как и в языке JavaScript *(см. разд. 2.15.1)*.

4.6.8. Операторы управления

Управление вычислительным процессом производится с помощью операторов условного перехода `if`, `switch`, и `?:` и операторов цикла `for`, `while`, `do-while` и `foreach`.

Операторы условного перехода обеспечивают выполнение того или иного фрагмента кода в зависимости от выполнения некоторого условия. Операторы цикла позволяют многократно выполнить некоторый фрагмент кода, пока не выполнится некоторое условие остановки этого процесса.

Основные из названных операторов, известные в большинстве языков программирования, такие же, как и в JavaScript. Поэтому за подробным их описанием я отсылаю вас к *разд. 2.5.6* и *2.5.7*. Однако в PHP оператор `if` имеет особенности, а оператор `foreach` вообще не имеет аналогов в JavaScript. Далее мы рассмотрим наиболее подробно лишь операторы `if`, `switch` и `?:`. Специфический для PHP оператор `foreach`, предназначенный для работы с массивами и объектами, будет рассмотрен в *разд. 4.10.4*.

Оператор *if*

Оператор условного перехода `if` позволяет реализовать структуру условного выражения:

если ..., то ..., иначе ...

Синтаксис оператора `if` перехода следующий:

```
if (условие)
  {код, который выполняется, если условие истинно}
else
  {код, который выполняется, если условие ложно}
```

В фигурных скобках располагается блок кода — несколько выражений, разделенных точкой с запятой, как это обязательно в PHP. Если в блоке используется не более одного выражения, то фигурные скобки можно не писать. Часть этой конструкции, определяемая ключевым словом `else` (иначе), необязательна. В этом случае остается только часть, определенная ключевым словом `if` (если):

```
if (условие){
  код, который выполняется, если условие истинно
}
```

Вместо ключевого слова `else` можно использовать `elseif (условие)`, чтобы установить проверку еще одного условия:

```
if (условие1){
  код, который выполняется, если условие1 истинно
```

```
} elseif (условие2){
  код, который выполняется, если условие1 ложно, а условие2 истинно
} else {
  код, который выполняется, если условие1 и условие2 ложны
};
```

Конструкций `elseif` в операторе `if` может быть несколько или ни одной.

Примечание

Оператор `if` в PHP почти такой же, что и в JavaScript *(см. разд. 2.5.6)*. Особенность состоит в том, что в JavaScript не предусмотрена конструкция `elseif`.

Оператор *switch*

Оператор `switch` (переключатель) удобен, когда требуется значение переменной или выражения сравнить с определенными величинами и выполнить тот или иной фрагмент кода в зависимости от результата сравнения.

Синтаксис оператора `switch` выглядит следующим образом:

```
switch (выражение) {
  case вариант1:
    код
    [break]
  case вариант2:
    код
    [break]
  ...
  [default:
    код]
}
```

Пример:

```
$x=2;
switch ($x) {
  case 1:
    echo "x равно 1";
  case 2:
    echo "x равно 2";
  case 3:
    echo "x равно 3";
}
```

В приведенном выше примере сработают 2-й и 3-й варианты. Если мы хотим, чтобы сработал только один какой-нибудь вариант (только тот, который соответствует значению выражения), то нужно использовать оператор `break`.

Пример:

```
$x=2;
switch ($x) {
  case 1:
    echo "x равно 1";
    break;
  case 2:
    echo "x равно 2";
    break;
  case 3:
    echo "x равно 3";
    break;
}
```

В этом примере сработает только 2-й вариант.

Примечание

Оператор `if` в PHP такой же, что и в JavaScript *(см. разд. 2.5.6)*.

Оператор условия *?:*

Оператор условия является сокращенной формой оператора условного перехода `if...else...`. Обычно он применяется вместе с оператором присваивания одного из двух возможных значений, в зависимости от значения условного выражения. Синтаксис оператора условия следующий:

условие ? *выражение1* : *выражение2*

С оператором присваивания оператор условия имеет такой вид:

переменная=условие ? *выражение1* : *выражение2*

Оператор условия возвращает значение выражения *выражение1*, если *условие* истинно, в противном случае — значение выражения *выражение2*. Поэтому он может использоваться в составных выражениях.

Пример:

```
$x=5;
$x<10 ? "x меньше 10": "x больше или равно 10";
echo $x;
```

Примечание

Оператор условия в PHP такой же, как и в JavaScript *(см. разд. 2.5.6)*.

4.6.9. Операторы цикла

Оператор цикла обеспечивает многократное выполнение блока программного кода до тех пор, пока не выполнится некоторое условие. В PHP предусмотрены четыре оператора цикла: `for`, `while`, `do-while` и `foreach`. Операторы `for`, `while` и `do-while` в PHP такие же, как и в JavaScript *(см. разд. 2.5.7)*. Поэтому в данном разделе мы не будем на них останавливаться. Оператор `foreach` (для каждого) специально предназначен для работы с массивами в PHP. Он очень удобен и будет подробно рассмотрен в *разд. 4.10.4 ("Перемещение по массиву")*.

4.7. Строки

Работа со строками в PHP при решении задач, связанных с Web-сайтами, является весьма приоритетной. Напомню, что строковые значения (последовательности символов) должны быть заключены в двойные или одинарные кавычки. Вид кавычек в PHP, в отличие от JavaScript, играет важную роль.

4.7.1. Двойные и одинарные кавычки

В двойных кавычках помимо обычных видимых и служебных символов могут находиться имена переменных (идентификаторы, начинающиеся с символа `$`), а также теги HTML. При выводе таких строк на экран с помощью оператора `echo` интерпретатор PHP заменяет идентификаторы переменных их значениями, а теги HTML интерпретируются браузером. В браузере служебные символы, такие как `\n` (перевод строки) и `\t` (табуляция), не выполняют своей функции, но отображаются в виде пробела. Если строковое значение заключено в одинарные кавычки, то служебные символы отображаются в виде пар `\n`, `\t` и т. п. При этом теги HTML интерпретируются браузером.

Пример:

```
$x="Привет";
$y="$x всем";     // Привет всем
$z='$x всем';     // $x всем
```

Длинные последовательности символов, которые в текстовом редакторе занимают несколько строк, можно задавать с помощью так называемого heredoc-синтаксиса, не используя кавычек:

```
Переменная=<<<МЕТКА
  текст
МЕТКА;
```

Пример:

```
<?php
$x=<<<MYLABEL
        Во первых
        строках
        своего письма
MYLABEL;
echo $x;
?>
```

При работе с модулем PHP CLI (с интерфейсом командной строки) текст на экране будет иметь такое же расположение, что и между метками в PHP-коде. Однако в окне браузера он будет отображен в виде одной или нескольких строк в зависимости от ширины клиентской области браузера. При этом части текста, расположенные в PHP-коде в различных строках, будут отображаться в одной строке через пробел.

Рассмотренный выше пример в окне браузера отображается точно так же, как и следующий код:

```
<?php
  $x="
  Во первых
  строках
  своего письма
  ";
  echo $x;
?>
```

Внимание

При использовании heredoc-синтаксиса необходимо соблюдать следующие правила:

- за первым идентификатором метки не должно быть никаких символов, в том числе и пробелов;
- за заключительным идентификатором метки должен находиться единственный символ точки с запятой.

Чтобы в строке, обрамленной двойными кавычками, отключить интерпретацию таких служебных символов, как двойные кавычки и `$`, перед ними следует указать обратный слэш (`\`). На рис. 4.8 показаны примеры.

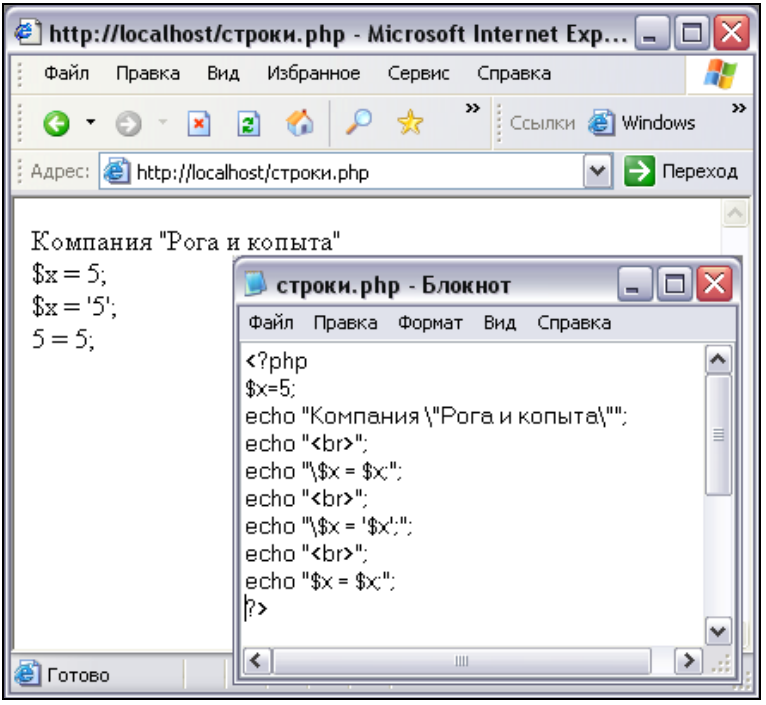


Рис. 4.8. Действие обратного слэша в строках, заключенных в двойные кавычки

Строки в одинарных кавычках хранятся как есть, за исключением пары `\'`, которая хранится как символ апострофа (одинарная кавычка). Например, следующий код

```
<?php
  $x=5;
  echo 'Компания \'Рога и копыта\' купит $x хвостов';
?>
```

выведет в окно браузера строку

```
Компания 'Рога и копыта' купит $x хвостов
```

Как уже отмечалось ранее, в строках могут встречаться теги HTML, которые позволяют отформатировать выводимый текст, а также выполнить другие действия. Интерпретатор PHP не интерпретирует их, а сохраняет их как есть.

С помощью оператора `echo` они направляются браузеру, который выполняет соответствующий HTML-код. Поскольку внутри HTML-кода могут использоваться кавычки различных видов, при формировании строк в PHP следует быть внимательным, чтобы правильно их расставлять и сочетать друг с другом.

4.7.2. Склейка строк

Соединение двух строк производится посредством операции склейки (конкатенация), которая обозначается точкой (`.`). В результате строка справа от этого символа приписывается без пробелов к концу строки, расположенной слева.

Пример:

```
$x="Привет";
$y=$x." всем";                  // Привет всем
$y="$x всем"." моим друзьям";   // Привет всем моим друзьям
$y.=" и знакомым";              // Привет всем моим друзьям и знакомым
```

В операции склейки могут участвовать не только строки, но и данные других типов. При этом все операнды приводятся к строковому типу автоматически.

Пример:

```
$x=5;
$y=$x." рублей";      // 5 рублей
$y=($x/2)." рубля";   // 2.5 рубля
$z=3>2;               // true
$y=$x.$z;             // 51
```

Обратите внимание на применение круглых скобок в третьем выражении, задающих порядок вычислений. Без них было бы выведено сообщение об ошибке.

4.7.3. Преобразование строк

Для преобразования символьных строк в PHP имеется большое количество встроенных функций. Здесь мы рассмотрим лишь несколько наиболее важных из них.

- `strlen(`*`строка`*`)` — возвращает длину строки (количество символов), указанной в качестве параметра. При этом служебные символы, такие как перевод строки, табуляция и др., также будут подсчитаны. Например, строка "Привет, друзья!" может быть набрана в PHP-коде по-разному, что может

сказаться на ее длине, хотя все варианты при этом могут отображаться одинаково:

```
$x="Привет, друзья!";
strlen($x);  // возвращает 15
$x="Привет,
друзья";     // возвращает 16
```

Здесь во втором варианте присваивания значения переменной `$x` после запятой присутствует невидимый служебный символ перевода строки, который в действительности состоит из двух символов, а перед словом "друзья" нет ни одного пробела (в первом варианте есть один пробел). Не трудно подсчитать, что 15 – 1 + 2 = 16. Данное обстоятельство следует учесть, если вам потребуется написать какой-нибудь код анализа текстовых строк.

- `trim(`*строка*`)`, `ltrim(`*строка*`)`, `rtrim(`*строка*`)` — удаляют ведущие и заключительные пробелы, только ведущие и только заключительные пробелы соответственно.
- `strchr(`*строка*`, `*символ*`)` — возвращает часть строки, начиная с указанного символа и до конца строки. Если строка не найдена, то возвращается `false`.

  Пример:

  ```
  strchr("BcADaf ag", "a");  // af ag
  ```

- `stristr(`*строка*`, `*символ*`)` — возвращает часть строки, начиная с указанного символа и до конца строки. Если строка не найдена, то возвращается `false`. В отличие от `strchr()` регистр строки и символа не учитывается.

  Пример:

  ```
  strchar("BcADaf ag", "a");  // ADaf ag
  ```

- `strichr(`*строка*`, `*символ*`)` — возвращает часть строки, начиная с указанного символа и до конца строки. Если строка не найдена, то возвращается `false`. В отличие от `strchr()` поиск символа ведется от конца строки.

  Пример:

  ```
  strrchr("BcADaf ag", "a");  // ag
  ```

- `strpos(`*строка1*`, `*строка2*`)` — возвращает позицию первого вхождения второй строки в первую. Если строка не найдена, то возвращается `false`. Нумерация позиций начинается с нуля и производится слева направо.

  Пример:

  ```
  strpos("Привет, мои друзья и мои знакомые", "мои");   // 8
  ```

- `strrpos(`*`строка1, строка2`*`)` — возвращает позицию последнего вхождения второй строки в первую. Если строка не найдена, то возвращается `false`. Нумерация позиций начинается с нуля и производится слева направо.

  Пример:

  ```
  strrpos("Привет, мои друзья и мои знакомые", "мои");   // 21
  ```

- `substr(`*`строка, число1, число2`*`)` — возвращает подстроку, начинающуюся с позиции *число1* и длиной *число2*. Нумерация позиций начинается с нуля и производится слева направо.

  Пример:

  ```
  substr("Привет, мои друзья и мои знакомые", 8, 10);   // мои друзья
  ```

- `str_replace(`*`строка1, строка2, строка`*`)` — заменяет в *строка* все вхождения *строка1* на *строка2*. В качестве заменяемой и замещающих строк могут быть не только видимые символы, но и символы табуляции, конца строки и файла.

  Пример:

  ```
  str_replace("мои", "наши", "Привет, мои друзья и мои знакомые");
   /* Привет, наши друзья и наши знакомые */
  ```

- `strtr(`*`строка, строка1, строка2`*`)` — заменяет в *строка* каждое вхождение любого символа из *строка1* на соответствующий ему символ в *строка2*. Если *строка1* и *строка2* имеют различные длины, то "лишние" символы в более длинной строке не используются.

  Пример:

  ```
  strtr("Привет, мои друзья", "иуяПрвтдзь","iuaPrvtdzj");
  // Privet, moi druzja
  ```

  Здесь второй и третий параметры функции содержат информацию о соответствии символов для "перекодировки" строки, указанной в качестве первого параметра. Так, в примере каждое вхождение буквы "и" заменяется буквой "i", "у" — "u" и т. д.

- `str_repeat(`*`строка, число`*`)` — повторяет указанную строку заданное количество раз и возвращает ее.

  Пример:

  ```
  str_repeat("#",10);  // ##########
  ```

- `strrev(`*`строка`*`)` — возвращает указанную строку, заменив порядок следования символов на противоположный.

  Пример:

  ```
  strrev("абвгд");  // дгвба
  ```

- `strtolower(`*`строка`*`)` — возвращает строку, преобразовав все символы указанной строки в нижний регистр.
- `strtoupper(`*`строка`*`)` — возвращает строку, преобразовав все символы указанной строки в верхний регистр.
- `ucfirst(`*`строка`*`)` — преобразует первый символ указанной строки в верхний регистр и возвращает ее.
- `ucwords(`*`строка`*`)` — преобразует первый символ каждого слова указанной строки в верхний регистр и возвращает ее.
- `str_word_count(`*`строка`*`[, `*`формат`*`])` — возвращает массив слов, входящих в строку или количество слов в строке, если второй параметр не указан. Параметр *`формат`* может принимать два целочисленных значения, которые определяют вид возвращаемого массива:
  - 1 — числовые индексы элементов массива соответствуют порядковому номеру слова в строке; нумерация начинается с 0;
  - 2 — числовые индексы элементов массива соответствуют позиции слова в строке; нумерация начинается с 0.

  Пример:

```
$x="How do you do";
str_word_count($x);     // 4
print_r(str_word_count($x, 1));
/* Выводит информацию о массиве слов: */
Array (
  [0] => How
  [1] => do
  [2] => you
  [3] => do
 )
print_r(str_word_count($x, 2));
/* Выводит информацию о массиве слов: */
Array (
  [0] => How
  [4] => do
  [7] => you
  [11] => do
 )
```

Примечание

Массивы подробно рассматриваются в *разд. 4.10*.

- `strtok(`*`строка1, строка2`*`)` — возвращает первую подстроку строки *`строка1`*, отделенную от остальной части любым из символов-разделителей, указанных в строке *`строка1`*. Если исходная строка не содержит указанных разделителей, то возвращается эта строка целиком.

  Пример:

```
<?php
  $string = "Это пример\nстроки";
  $word=strtok($string, " \n");   // значение $word равно "Это"
?>
```

  Здесь во втором параметре функции `strtok()` указаны два разделителя — пробел и перевод строки (`\n`). Разумеется, вы можете указать там и другие символы, например, символ табуляции (`\t`), вертикальную черту (`|`), запятую (`,`) и другие специальные или обычные символы.

  Данная функция имеет особенности, которые следует учесть при ее применении. А именно `strtok()` сохраняет позицию выделенной подстроки и при втором и последующих вызовах ей достаточно передать в качестве единственного параметра только строку разделителей, чтобы разбить всю исходную строку на подстроки. В следующем примере выводятся все слова (подстроки), которые принципиально можно выделить посредством таких разделителей, как пробел, перевод строки (`\n`) и табуляция (`\t`). Обратите внимание, что в исходной строке символ табуляции отсутствует, а потому этот символ во втором параметре функции не производит никакого действия.

  Пример:

```
<?php
  $string = "Это пример\nстроки";
  $word = strtok($string, " \n\t");
  while ($word) {
    echo "$word<br>";
    $word = strtok(" \n\t");
  }
?>
```

  В результате выполнения этого кода, на экран будет выведен список из трех слов, которые выделил из исходной строки оператор `echo`. Заметьте, что тег `<br>`, добавляемый к выводимой строке, обеспечивает лишь вывод обнаруженных слов в отдельных строках. Иначе говоря, он здесь нужен лишь для форматирования вывода.

Начиная с версии PHP 4.1.0, пустые подстроки пропускаются функцией `strtok()`.

Итак, функция `strtok()` берет в качестве параметра некоторую строку и возвращает подстроку в соответствии с заданными разделителями. Повторными вызовами, обычно в цикле (операторы `for`, `while`, `do-while`), с помощью этой функции можно "выудить" все подобные подстроки.

На практике обычно требуется преобразовать строку в массив или наоборот. В PHP для этого используются функции `explode()` и `implode()`, которые мы рассмотрим в *разд. 4.10*.

4.7.4. Форматирование строк

Для форматирования текстовых строк, выводимых в окно браузера оператором `echo`, часто используется HTML и каскадные таблицы стилей. Служебные символы `\n` (перевод строки) и `\t` (табуляция) также можно использовать в строках. Если они находятся в строке, заключенной в одинарные кавычки, то при выводе этой строки оператором `echo` эти символы не будут работать по своему назначению и будут видны на экране. В случае двойных кавычек они, вообще говоря, также не сработают, но отобразятся в виде пробелов. Однако если вы хотите все же заставить их функционировать, то строку с двойными кавычками следует начать тегом `<pre>` *(см. разд. 1.2.6)*. Например, следующий оператор выведет сообщение в виде двух строк, причем название фирмы отобразится жирным шрифтом:

```
echo "<pre><b>АО 'Рога и копыта'<b>\nприглашает на работу главного бух-
галтера"
```

Вместе с тем, с целью форматирования можно применять специальные функции PHP `printf()` и `sprintf()`. Если вы используете PHP CLI, работающий в режиме командной строки, то специальные функции являются единственным средством форматирования строк.

На практике часто приходится формировать строки, содержащие не только обычные алфавитные символы и знаки препинания, но и числа и значения переменных. При этом числа можно представлять в различных системах счисления, а в десятичной системе — в различных видах. В функциях `printf()` и `sprintf()` можно указать шаблон, в соответствии с которым будет отформатирована строка. Эти функции имеют одинаковый синтаксис, но отличаются действием и возвращаемым значением:

- `printf(`*формат_строка*`, `*список_переменных*`)` — выводит отформатированную строку и возвращает длину этой строки;
- `sprintf(`*формат_строка*`, `*список_переменных*`)` — возвращает отформатированную строку (но не выводит ее).

список_переменных — имена переменных, разделенные запятой, значения которых следует подставить в форматируемую строку. Этот параметр не обязателен.

Параметр *формат_строка* представляет собой текстовую строку, которую требуется отформатировать в соответствии с описателями преобразований. Описатели преобразований находятся в форматируемой строке, и каждый из них описывает представление значения соответствующей переменной, указанной в списке: первый описатель для первой переменной, второй — для второй переменной и т. д. Разумеется, строка может и не содержать описателей. В этом случае она просто не будет отформатирована.

Каждый описатель преобразований состоит из знака процента (%), за которым следует один или более дополнительных элементов в таком порядке: *%ЗаполнительВыравниваниеДлинаТочностьТип*. Рассмотрим эти необязательные элементы.

- *Заполнитель* — символ, который будет использоваться для дополнения результата преобразования до заданной длины *(см. далее)*. Это может быть пробел (по умолчанию), 0 или любой другой символ, перед которым необходимо указать одинарную кавычку (').
- *Выравнивание* — символ минус (-); если он указан, то выравнивание происходит по левому краю поля, выделенного для отображения значения, а в противном случае (т. е. по умолчанию) — по правому краю.
- *Длина* — число, определяющее ширину поля, отводимого для вывода результата преобразования. Если результат содержит меньше символов, то оставшееся пространство будет заполнено пробелом или символом заполнения, если он был указан.
- *Точность* — число, перед которым указывается точка. Определяет, сколько десятичных разрядов в дробной части числа следует отображать. Если число имеет больше таких разрядов, то результат получается округлением, а не отбрасыванием лишних знаков. Применяется для чисел с плавающей точкой (float).
- *Тип* — символ, определяющий, как трактовать тип значения переменной из списка, которое подставляется в форматируемую строку. Допустимые следующие указатели типа:
  - % — значение переменной не используется;
  - b — значение трактуется как целое и выводится в виде двоичного числа;
  - c — значение трактуется как целое и выводится в виде символа с соответствующим кодом ASCII;

- `d` — значение трактуется как целое и выводится в виде десятичного числа со знаком;
- `e` — значение трактуется как число с плавающей точкой и представляется в экспоненциальной форме;
- `u` — значение трактуется как целое и выводится в виде десятичного числа без знака;
- `f` — значение трактуется как число с плавающей точкой и представляется в неэкспоненциальной форме (т. е. только с разделительной точкой);
- `o` — значение трактуется как целое и выводится в виде восьмеричного числа;
- `s` — значение трактуется как строка;
- `x` — значение трактуется как целое и выводится в виде шестнадцатеричного числа в нижнем регистре;
- `X` — значение трактуется как целое и выводится в виде шестнадцатеричного числа в верхнем регистре.

В листинге 4.3 приведен код, в котором используется функция `printf()` для форматирования строк так, чтобы получился список. Результат выполнения этого кода показан на рис. 4.9.

Листинг 4.3. Пример форматирования строк *(вариант 1)*

```
<?php
  $x1=15.378;
  $x2=145.2;
  $n1="Балаганов";
  $n2="Паниковский";
  printf("<pre><b>Платежная ведомость</b>\n");
  printf("%'.-12s%'.10.2f руб.\n", $n1, $x1);
  printf("%'.-12s%'.10.2f руб.\n", $n2, $x2);
?>
```

В данном примере был использован тег `<pre>`, который обеспечивает отображение браузером строк в соответствии с их предварительным форматированием. Здесь это понадобилось, чтобы сработали служебные символы `\n` для перехода на другую строку. В качестве заполнителя была применена точка (`'.`). Фамилии выравнивались по левому краю (`-`) поля шириной 12 символов с заполнителем в виде точки (`'.`). Таким образом, описатель преобразования для строковых значений имеет вид: `%'.-12s`.

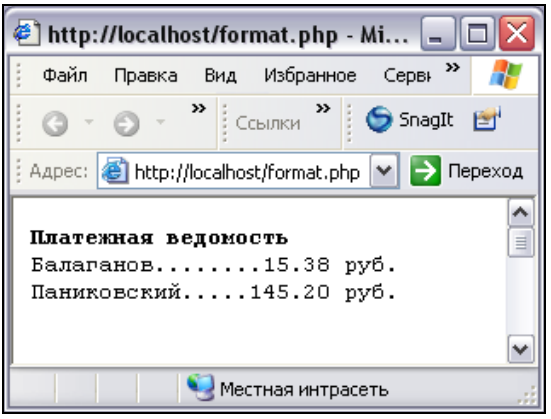


Рис. 4.9. Пример форматирования строк

Числа представлялись с десятичной точкой и двумя разрядами после нее. Они выравнивались по правому краю поля из 10 символов, а в качестве заполнителя использовалась точка. Таким образом, описатель преобразования для чисел имеет вид: `%'.10.2f`.

Такой же результат можно получить, используя функцию `sprintf()` для создания отформатированной строки и оператор `echo` — для ее вывода (листинг 4.4).

Листинг 4.4. Пример форматирования строк *(вариант 2)*

```
<?php
  $x1=15.387;
  $x2=145.2;
  $n1="Балаганов";
  $n2="Паниковский";
  $str="<pre><b>Платежная ведомость</b>\n";
  $str.="%'.-12s%'.10.2f руб.\n";
  $str=sprintf($str,$n1, $x1);
  $str.="%'.-12s%'.10.2f руб.\n";
  echo sprintf($str,$n2, $x2);
?>
```

Напомним, что оператор `.=` обеспечивает склейку текущего строкового значения переменной, расположенной слева от него, с тем, которое указано справа от этого оператора *(см. разд. 4.6.4)*.

Заметим, что порядок описателей в форматируемой строке соответствует порядку других параметров функций `sprintf()` и `printf()`. Для PHP версии 4.0.6 и старше можно изменить этот порядок и даже использовать один и тот же описатель несколько раз. Для этого в каждом описателе сразу же за символом `%` следует указать номер, обратный слэш и символ `$` (далее могут следовать другие элементы). Этот номер соответствует порядковому номеру последующих аргументов функций `sprintf()` и `printf()`. В листинге 4.5 приведен пример кода, выводящего строку "Человек, обезьяна и корова есть млекопитающие".

Листинг 4.5. Пример форматирования с использованием нумерации параметров

```
<?php
  $x="человек";
  $y="обезьяна";
  $z="корова";
  $type="млекопитающие";
  echo ucfirst(sprintf("%2\$s, %3\$s и %4\$s есть %1\$s", $type, $x, $y, $z));
?>
```

В результате выполнения этого кода будет выведена строка "Человек, обезьяна и корова есть млекопитающие". Здесь функция `ucfirst()` использовалась для того, чтобы отформатированная строка начиналась с прописной (заглавной) буквы.

4.8. Числа

Основные сведения о числах были рассмотрены в *разд. 4.3* и *4.6.2*. Там мы узнали, что числа в PHP бывают целыми (`integer`) и с плавающей точкой (`float`), а также то, что к ним могут быть применены арифметические операторы, обозначаемые как `+`, `-`, `*`, `/` и `%`. Отметим еще одно важное обстоятельство: арифметические операторы можно выполнять над строками, содержащими числа. Другие, более сложные, вычисления можно выполнять с помощью встроенных функций, которых в PHP имеется большое количество. Кроме того, есть функции, позволяющие отображать числа в нужном виде.

4.8.1. Математические функции

Из большого набора имеющихся в PHP математических функций приведем лишь наиболее употребительные:

- `abs(`*число*`)` — модуль (абсолютное значение) числа;

- `acos(`*число*`)` — арккосинус числа;
- `asin(`*число*`)` — арксинус числа;
- `atan(`*число*`)` — арктангенс числа;
- `atan2(`*x*`, `*y*`)` — угол в полярных координатах точки;
- `ceil(`*число*`)` — округление числа вверх до ближайшего целого;
- `cos(`*число*`)` — косинус числа;
- `exp(`*число*`)` — число *e* в степени *число*;
- `floor(`*число*`)` — округление числа вниз до ближайшего целого;
- `fmod(`*число1*`, `*число2*`)` — возвращает дробный остаток от деления первого числа на второе. Этот остаток определяется как число, удовлетворяющее уравнению *число1* = `i` × *число2* + `r`, где `i` — некоторое целое. Остаток `r` имеет такой же знак, что и *число1*;
- `log(`*число*`)` — натуральный логарифм числа;
- `log10(`*число*`)` — десятичный логарифм числа;
- `max(`*число1*`, `*число2*`, ..., `*числоN*`)` — большее из чисел в списке;
- `max(массив_чисел)` — большее из чисел в массиве;
- `min(`*число1*`, `*число2*`, ..., `*числоN*`)` — меньшее из чисел в списке;
- `max(массив_чисел)` — большее из чисел в массиве;
- `pow(`*число1*`, `*число2*`)` — *число1* в степени *число2*;
- `rand(`*число1*`, `*число2*`)` — случайное число между заданными числами;
- `round(`*число*`)` — округление числа до ближайшего целого;
- `sin(`*число*`)` — синус числа;
- `sqrt(`*число*`)` — квадратный корень из числа;
- `srand(`*число*`)` — устанавливает начальное значение генератора случайных чисел в указанное число; если параметр не указан, то начальное число выбирается случайно;
- `tan(`*число*`)` — тангенс числа.

4.8.2. Математические константы

В PHP имеется множество предопределенных числовых констант, которые используются в инженерных и научных расчетах (табл. 4.8).

Таблица 4.8. *Математические константы*

Имя константы	Значение	Описание
M_PI	3.14159265358979323846	Число π
M_E	2.7182818284590452354	Число *e* (основание натуральных логарифмов)
M_LOG2E	1.4426950408889634074	$\log_2 e$
M_LOG10E	0.43429448190325182765	lg *e*
M_LN2	0.69314718055994530942	ln *e*
M_LN10	2.30258509299404568402	ln 10
M_PI_2	1.57079632679489661923	$\pi/2$
M_PI_4	0.78539816339744830962	$\pi/4$
M_1_PI	0.31830988618379067154	$1/\pi$
M_2_PI	0.63661977236758134308	$2/\pi$
M_SQRTPI	1.77245385090551602729	$\sqrt{\pi}$
M_2_PI	0.63661977236758134308	$2/\sqrt{\pi}$
M_SQRTPI	1.77245385090551602729	$\sqrt{2}$
M_SQRT3	1.73205080756887729352	$\sqrt{3}$
M_SQRT1_2	0.70710678118654752440	$1/\sqrt{2}$
M_LNPI	1.14472988584940017414	ln π
M_EULER	0.57721566490153286061	Постоянная Эйлера

4.8.3. Представление чисел в различных системах счисления

Числа в PHP можно представлять в различных системах счисления, т. е. в системах с различными основаниями: 10 (десятичной), 16 (шестнадцатеричной), 8 (восьмеричной) и 2 (двоичной). Десятичная форма представления чисел, использующая цифры от 0 до 9, наиболее нам привычна в обиходе. Однако в компьютерных технологиях часто используются и другие системы: восьмеричная (цифры от 0 до 7), шестнадцатеричная (цифры 0, ..., 9, a, b, ..., f) и двоичная (цифры 0 и 1). Например, десятичное число 17 в восьмеричной

системе представляется как 15, в шестнадцатеричной — как b0, а в двоичной — как 10001.

Для преобразований чисел из одной системы счисления в другую служат специальные функции.

- `base_convert(`*`строка1, основание1, основание2`*`)` — преобразует строку *`строка1`*, содержащую число в системе счисления по основанию *`основание1`*, в строку с числом по основанию *`основание2`* и возвращает строку, содержащую результат этого преобразования.

  Примеры:

  ```
  base_convert("17", 10, 16);  // "11"
  base_convert("17", 10, 8);   //  "21"
  base_convert("17", 10, 2);   // "111"
  base_convert("ff", 16, 10);  // "255"
  base_convert("101", 2, 10);  // "5"
  ```

  Обратите внимание, что данная функция возвращает строку, содержащую число в том или ином представлении, а не число (т. е. данные числового типа). При рассмотрении следующих функций обратите внимание на то, какого типа данные они возвращают и принимают в качестве параметров.

- `dechex(`*`целое_число`*`)` — возвращает строку, содержащую указанное целое десятичное число, в шестнадцатеричном виде.

  Примеры:

  ```
  dechex(3);      // "3"
  dechex(250);    // "fa"
  dechex(255);    // "ff"
  ```

- `hexdec(`*`строка`*`)` — возвращает десятичное число, соответствующее указанному в строковом параметре шестнадцатеричному числу.

  Пример:

  ```
  hexdec("a");    // 10
  ```

- `decbin(`*`целое_число`*`)` — возвращает строку, содержащую указанное целое десятичное число, в двоичном виде.

  Примеры:

  ```
  decbin(3);      // "11"
  decbin(250);    // "11111010"
  decbin(255);    // "11111111"
  ```

- `bindec(`*`строка`*`)` — возвращает десятичное число, соответствующее указанному в строковом параметре двоичному числу.

Пример:

```
hexdec("101");  // 5
```

- `decoct(целое_число)` — возвращает строку, содержащую указанное целое десятичное число, в восьмеричном виде.

  Примеры:

```
decoct(3);      // "3"
decoct(250);    // "372"
decoct(255);    // "377"
```

- `octdec((строка)` — возвращает десятичное число, соответствующее указанному в строковом параметре восьмеричному числу.

  Пример:

```
octdec("10");   // 7
```

Внимание

Во всех перечисленных выше функциях строковые параметры можно передавать и без кавычек (т. е. в виде литералов).

Представления чисел в различных системах счисления, отличных от десятичной, являются строковыми данными. Арифметические операторы для этих данных действуют так, как будто это десятичные числа, независимо от того, принадлежат они к числовому или строковому типу. Иначе говоря, арифметика остается десятичной, независимо от системы счисления операндов.

Примеры:

```
decbin(2)+decbin(3);        // 21 ("10"+"11")
dechex(3)+dechex(16);       // 13 ("3"+"10")
bindec("10")+bindec("11");  // 5 (2+3)
```

4.8.4. Форматирование чисел

Числа в PHP хранятся и выводятся в наиболее эффективном формате. Например, если переменная имеет значение 15.00, оно будет отображено на экране как 15. В приложениях может потребоваться отобразить число в некотором определенном формате, например, разделить группы разрядов пробелом, отобразить два знака после разделительной точки для представления денежных сумм и т. п. Некоторые возможности форматирования чисел мы уже рассматривали в *разд. 4.7.4* — функции `printf()` и `sprintf()`. Здесь же мы остановимся на удобной функции форматирования чисел:

`number_format(число, точность, разделитель1, разделитель2)`

Эта функция возвращает строку, содержащую отформатированное число, и принимает следующие параметры:

- *число* — число (`float`), которое требуется отформатировать; если следующие параметры не используются, то число будет представлено без дробной части (с округлением до ближайшего большего целого) и запятыми в качестве разделителя групп по три разряда;
- *точность* — целое число, указывающее, сколько знаков в дробной части числа следует показать (по умолчанию — 0); этот параметр обязателен, если вы собираетесь использовать параметры *разделитель1* и *разделитель2*; если вы указываете меньшее количество знаков, чем есть в числе, то происходит округление в большую сторону;
- *разделитель1* — строка с символом, который используется для разделения целой и дробной частей числа (по умолчанию — точка); этот параметр обязателен, если вы собираетесь использовать параметр *разделитель2*;
- *разделитель2* — строка с символом, который используется для разделения групп по три разряда в целой части числа (по умолчанию — запятая); этот параметр обязателен, коль скоро вы использовали *разделитель1*.

Таким образом, функция `number_format()` может принимать один, два или четыре параметра, но не три.

Примеры:

```
number_format(1234567890);                    // 1,234,567,890
number_format(123456.789, 2);                 // 123,456.79
number_format(123456.789, 2, ",", " ");       // 123 456,79
number_format(123456.789, 2, "рублей", " ");  // 123 456р79
```

Третий и четвертый параметры могут быть строками из нескольких символов, но в качестве разделителей будут использованы лишь первые символы.

Примечание

В западной традиции в качестве разделителя целой и дробной частей числа принято использовать точку, а для разделения групп разрядов — запятую. В отечественных документах применяют запятую и пробел соответственно.

Внимание

Математические вычисления следует выполнять до форматирования чисел. Не забывайте, что функция форматирования возвращает текстовую строку и применяется для вывода чисел в надлежащем виде.

4.9. Дата и время

Дата и время хранятся в компьютере в специальном формате timestamp как количество секунд, прошедшее от 1.01.1970 00:00:00 GMT (или, что то же самое, — UCT, *см. разд. 2.11*). В приложениях обычно требуется другое представление даты и времени, для чего в PHP предусмотрены специальные функции.

Текущее системное время в формате timestamp (т. е. как целое количество секунд) получается с помощью функции `time()`. Для получения в этом же формате любого другого времени служит функция

`mktime(`*час*`,` *мин*`,` *сек*`,` *мес*`,` *день*`,` *год*`)`

с целочисленными параметрами, смысл которых ясен. Например, вызов функции `mktime(12, 49, 0, 2, 9, 2006)` вернет число 1139478540 — количество секунд, прошедшее от 1.01.1970 00:00:00 до 9.02. 2006 12:49:00 GMT.

Для представления даты и времени в требуемом формате как строки можно использовать функцию

`date(`*формат*`,` *timestamp*`)`

Здесь параметр *timestamp* — целое число, соответствующее количеству секунд, прошедших с момента 1.01.1970 00:00:00 GMT, а *формат* — строка символов форматирования даты и времени. Допустимые символы представлены в табл. 4.9.

Таблица 4.9. *Символы форматирования даты и времени*

Символ	Значение
d	День месяца (две цифры, используется ведущий 0)
j	День месяца (одна или две цифры)
m	Месяц (две цифры, используется ведущий 0)
n	Месяц (одна или две цифры)
Y	Год (четыре цифры)
y	Год (две цифры)
w	Номер дня недели (0 — воскресенье, 6 — суббота)
g	Часы (12-часовой формат без ведущего 0, от 1 до 12)
G	Часы (24-часовой формат без ведущего 0, от 0 до 23)
h	Часы (12-часовой формат с ведущим 0, от 01 до 12)
H	Часы (24-часовой формат с ведущим 0, от 00 до 23)

Таблица 4.9 *(окончание)*

Символ	Значение
i	Минуты (от 00 до 59)
s	Секунды (от 00 до 59)
a	До или после полудня: am или pm (в нижнем регистре)
A	До или после полудня: AM или PM (в верхнем регистре)
U	Целое количество секунд, прошедшее от 1.01.1970 00:00:00 GMT
M	Трехбуквенное сокращение английского названия месяца
F	Английское название месяца
D	Трехбуквенное сокращение английского названия дня недели
l	Английское название дня недели

Между символами форматирования можно использовать символы-разделители, например, дефис (-), точку (.), косую черту (/), двоеточие (:), пробел и др.

Если второй параметр функции `date()` не указан, то предполагается текущее время.

Примеры:

```
date("d.m.Y");                    // текущая дата, например, 10.02.2006
date("d.m.Y H:i:s", 10000000000);      // 07.09.2014 08:50:08
date("j F.Yг. g:i:s A", 10000000000);  // 7 September.2014г. 8:50:08 AM
```

Чтобы выполнить какие-либо вычисления над датами/временем, следует сначала перевести их в формат timestamp. В следующем примере выводится отформатированные дата и время, соответствующие 12 часам 49 минутам 0 секунд пяти суткам спустя после 9 февраля 2006 года:

```
<?php
  $t=mktime(12, 49, 0, 2, 9, 2006) + 5*24*60*60;
  echo date("d.m.Y H:i:s", $t);   // 14.02.2006 12:49:00
?>
```

Для перевода текстовых строк, содержащих дату и время, в формат timestamp служит функция

`strtotime(`*дата-время*`, `*timestamp*`)`

Эта функция принимает строку, которая содержит ключевые слова, связанные с датой и временем, и необязательный целочисленный параметр. Если он указан, то функция возвращает количество секунд относительно его значения,

а в противном случае — относительно текущего времени. При возникновении ошибки преобразования функция возвращает –1.

В строковом параметре *дата-время* функция `strtotime()` распознает следующие ключевые слова на английском языке:

- названия месяцев и их трехбуквенные сокращения;
- названия дней недели и их трехбуквенные сокращения;
- названия элементов даты и времени: `year` (год), `month` (месяц), `week` (неделя), `day` (день), `hour` (час), `minute` (минута), `second` (секунда), `am` (до полудня), `pm` (после полудня). Если требуются слова во множественном числе (что не обязательно), то следует добавить окончание `s`, например, `days`, `weeks` и т. п.;
- слова `ago` (тому назад), `now` (сейчас), `last` (последний), `next` (следующий), `this` (этот), `tomorrow` (завтра), `yesterday` (вчера);
- числа и знаки `+` и `-`;
- временные зоны: например, `gmt` (Greenwich Mean Time, среднее время по Гринвичу), `pdt` (Pacific Daylight Time, дневное тихоокеанское время).

Далее приведены примеры использования ключевых слов для формирования значения параметра *дата-время* функции `strtotime()`:

- `"now"` — текущее время;
- `"now + 48 hours"` — через 48 часов от текущего времени; здесь слово `now` можно опустить;
- `" + 48 hours"` — через 48 часов от текущего времени;
- `"10am + 5 days"` — 10 часов до полудня через 5 дней;
- `"3 months ago"` — 3 месяца назад;
- `"tomorrow 5pm"` — 5 часов пополудни завтра;
- `"last Friday"` — предыдущая пятница;
- `"next year gmt"` — один год спустя;
- `"2006/2/10"` — 10 февраля 2006г. 00:00:00;
- `"2006-2-10"` — то же самое;
- `"10 Feb 2006"` — то же самое.

Предположим, необходимо вывести дату и время, соответствующие 6 часам пополудни 3 месяца и 2 дня спустя. Это можно сделать с помощью следующего кода:

```
<?php
  $timestamp="d.m.Y H:i:s";
```

```
  $t=strtotime("+ 3 months + 2 days 6pm");
  echo date($timestamp, $t);
?>
```

Допустим, требуется узнать, сколько часов между двумя моментами времени, заданными как строковые значения, например, "9 Feb 2006 5am" и "2 Mar 2006 3pm". Для этого следует перевести строковые значения *даты-времени* в формат timestamp с помощью функции `strtotime`, найти разницу между ними и разделить ее на 3600:

```
<?php
  $t1=strtotime("9 Feb 2006 5am");
  $t2=strtotime("8 Mar 2006 3pm");
  echo ($t2-$t1)/3600;  // 658
?>
```

4.10. Массивы

Массив представляет собой упорядоченный набор данных (элементов), объединяемых под общим именем. Обращение к элементам массива происходит по индексу, который может быть как числовым, так и символьным.

4.10.1. Создание массива

Массив можно создать несколькими способами. Самый простой из них состоит в том, чтобы рядом с именем переменной поставить квадратные скобки и с помощью оператора присваивания задать значение элемента. Например:

```
$myarray[]="Иван";
```

В результате будет создан массив с именем `$myarray` и единственным элементом "Иван". К этому массиву можно добавить другие элементы различных типов, используя тот же синтаксис:

```
$myarray[]="Федоров";
$myarray[]=125.7;
```

В рассматриваемом случае массив `$myarray` будет содержать уже три элемента — два строковых и один числовой.

При создании массива таким способом PHP автоматически индексирует элементы, используя для этого целые числа, начиная с 0. Индексы (ключи) позволяют получить доступ к значениям элементов массива:

```
$myarray[0];  // "Иван"
$myarray[1];  // "Федор"
$myarray[2];  // 125.7
```

При создании массива можно сразу индексировать элементы по своему усмотрению, причем не обязательно с нуля и по порядку:

```
$myarray[1]="Иван";
$myarray[2]="Федоров";
$myarray[3]=125.7;
```

Кроме числовых индексов массивы могут иметь символьные (строковые) индексы, например,

```
$myarray['Имя']="Иван";
$myarray['Фамилия']="Федоров";
$myarray['Зарплата']=125.7;
```

Примечание

Числовые индексы могут быть представлены и соответствующими строками. Например, индексы 3 и "3" соответствуют одному и тому же элементу массива, а "03" — другому.

Массив также может быть создан с помощью следующей конструкции:

```
$имя_массива=array([индекс1=>] значение1[, [индекс2=>] значение2, ... ])
```

Здесь квадратные скобки указывают лишь на то, что заключенный в них элемент не является обязательным.

Примеры:

```
$myarray=array("Иван", "Федоров", 125.7);
$myarray=array('Имя'=>"Иван", 'Фамилия'=> "Федоров", 'Зарплата'=>125.7);
$myarray=array('Имя'=>"Иван", 2=> "Федоров", 3=>125.7);
```

В первом случае элементы массива автоматически получат числовые индексы, во втором — указанные символьные индексы, а в третьем — символьные и числовые. При создании массива можно совместно использовать индексы различных типов. Однако каждый элемент имеет только один индекс — числовой или символьный. Если какой-то элемент имеет символьный индекс, то обращаться к этому элементу с помощью его порядкового номера в массиве нельзя.

Для создания *регулярных массивов* можно использовать функцию диапазона

```
range(начало, конец, шаг)
```

которая принимает в качестве параметров:

- *начало* — число или символ начала диапазона;
- *конец* — число или символ конца диапазона;

- *шаг* — число, соответствующее приращению значений элементов массива (не обязательный параметр, по умолчанию равный 1).

Примеры:

```
range(20, 25);            // массив из элементов: 20, 21, 22, 23, 24, 25
range(20, 25, 2);           // массив из элементов: 20, , 22, 24
range(25, 22, 2);           // массив из элементов: 25, 23, 21
range(20, 25, 1.5);         // массив из элементов: 20, 21.5, 23, 24.5
range("a", "d");            // массив из элементов: "a", "b", "c" , "d"
range("a", "d", 2);         // массив из элементов: "a",  "c"
range("d", "a", 2);         // массив из элементов: "d",  "b"
range("Борис", "Дмитрий");  // массив из элементов: "Б",  "В", "Г", "Д"
```

Количество элементов (длину) массива можно определить с помощью функции `count(`*массив*`)` или `sizeof(`*массив*`)`.

Пример:

```
$myarray=array("Иван", "Федоров", 125.7);
$n=count($myarray);  // 3
```

Каким бы способом ни был создан массив, значения его элементов всегда можно изменить с помощью оператора присваивания.

Чтобы добавить новый элемент к существующему массиву, достаточно использовать следующие выражения:

- `$`*имя_массива*`[]=`*значение* — добавляется новый элемент с числовым индексом;
- `$`*имя_массива*`[`*индекс*`]=`*значение* — если в массиве нет элемента с указанным индексом, то элемент добавляется, а в противном случае — изменяется (перезаписывается).

Пример:

```
$myarray=array("Имя"=>"Иван",125.7); /* Структура массива:
                                          ["Имя"]=>"Иван"
                                          [0]=>125.7  */
$myarray[]="Профессор";              /*  Структура массива:
                                          ["Имя"]=>"Иван"
                                          [0]=>125.7
                                          [1]=>"Профессор" */
$myarray["тел"]="123-4567";          /*  Структура массива:
                                          ["Имя"]=>"Иван"
                                          [0]=>125.7
```

```
                                        [1]=>"Профессор"
                                        ["тел"]=>"123-4567" */

$myarray[0]="Федоров";                /*  Структура массива:
                                        ["Имя"]=>"Иван"
                                        [0]=>"Федоров"
                                        [1]=>"Профессор"
                                        ["тел"]=>"123-4567" */
```

Чтобы удалить элемент из массива или массив целиком, используют функцию `unset()` с параметром, указывающим на элемент массива или сам массив соответственно.

Пример:

```
$myarray=array("Один", "Два", "Три");  // массив из трех элементов
unset($myarray[1]);                     /* удаление 2-го элемента;
                              $myarray теперь содержит два элемента:
                              "Один" и "Три"  с индексами 0 и 2
                              соответственно */
unset($myarray);    // переменная $myarray теперь имеет значение null
```

Внимание

При удалении элементов из массива оставшиеся в нем элементы сохраняют свои прежние индексы. Иначе говоря, переиндексация массива (даже только с числовыми индексами) не происходит.

Как уже отмечалось ранее, обращение к элементам массива происходит с помощью индексов. Однако в случае обращения к несуществующему элементу выводится сообщение об ошибке. В ряде случаев это не желательно. Чтобы подавить возможное сообщение об ошибке, используют символ `@`, например,

```
@$anyelement=$myarray[5];
```

4.10.2. Многомерные массивы

Массивы, рассмотренные выше, являются одномерными. Однако элементы массива могут содержать данные различных типов, в том числе и массивы. Если в качестве элементов некоторого одномерного массива создать массивы, то получится двумерный массив. Обращение к элементам такого массива происходит в соответствии со следующим синтаксисом:

имя_массива[*индекс_уровня1*] [*индекс_уровня2*]

Если массив имеет размерность большую двух, то обращение к массивам имеет аналогичный синтаксис: следует добавить необходимое количество квадратных скобок, заключающих нужные индексы.

Пример:

```
/* Двумерный массив */
$сотрудники=array(
  array("Иван", "Иванов", 100),
  array("Василий", "Васильев", 200),
  array("Федор", "Федоров", 300)
);
echo $сотрудники[2][1];    // Федоров
```

Массив, приведенный в данном примере, можно представить в виде таблицы из трех столбцов и трех строк. Каждой строке соответствует массив значений, который сам является элементом массива `$сотрудники`.

4.10.3. Отображение массивов

Для просмотра структуры и значений элементов массива (это может понадобиться при отладке программ) используются функции `print_r()` и `var_dump()`. Эти функции достаточно подробно описывались в *разд. 4.4.4*.

Разумеется, для вывода значений элементов массива вы можете воспользоваться оператором цикла, например,

```
<?php
  $myarray=array("Иван","Федоров", 125.7);
  $i=0;
  while ($i<count($myarray)) {
    echo $myarray[$i],"<br>";
    $i++;
  }
?>
```

Вывод элемента массива с помощью оператора `echo` производится так же, как и обычной переменной, например,

```
echo $myarray[2];
```

Однако если требуется вывести элемент массива в составе текстовой строки, то его следует заключить в фигурные скобки.

Пример:

```
$myarray=array("Иван", "Федоров", 125.7);
echo "Сотрудник {$myarray[1]} получает {$myarray[2]} руб.";
     /* вывод строки:
        "Сотрудник Федоров получает 125.7 руб."  */
```

4.10.4. Операции над массивами

Копирование массивов

Чтобы скопировать массив, достаточно переменную, содержащую этот массив, присвоить другой переменной. Изменение значения элемента в одном из них никак не отразится на соответствующем элементе в другом массиве.

Пример:

```
$a=array(5, 2, 4, 3);
$x=$a;
$a[2]=25;
echo $x[2];  // 4
```

Примечание

В JavaScript присвоение переменной типа "массив" другой переменной не приводит к копированию массива, а создает лишь новую ссылку (псевдоним) на тот же самый массив.

Сортировка массивов

Первоначально значения элементов массива хранятся в том порядке, как они были созданы. Однако этот порядок можно изменить. Упорядочивать (сортировать) элементы массива можно по индексу и значению. В PHP существует много функций, позволяющих осуществить сортировку массивов различными способами. Здесь мы рассмотрим только некоторые наиболее употребительные из них.

Для сортировки элементов массива с числовыми индексами по возрастанию значений служит функция

`sort(`*массив* `[,` *режим*`])`

которая возвращает `true` или `false` соответственно при удачном и ошибочном завершении операции. Данная функция изменяет указанный массив, а именно изменяет индексы элементов в соответствии с новым порядком их расположения. Если массив имеет символьные индексы, то они заменяются

числовыми. Второй необязательный параметр задает режим сортировки (способ сравнения значений элементов) и может принимать следующие значения:

- `SORT_REGULAR` — сравнивать элементы, не изменяя их типы (по умолчанию);
- `SORT_NUMERIC` — сравнивать элементы как числа;
- `SORT_STRING` — сравнивать элементы как текстовые строки;
- `SORT_LOCALE_STRING` — сравнивать элементы как строки с учетом установленной локали (начиная с версии PHP 5.0.2).

Перечисленные выше константы имеют целочисленные значения 0, 1, 2 и 3 соответственно.

Пример:

```
$myarray=array("Вася", "Анна", 5);  /* Структура массива:
                                          [0]=>"Вася"
                                          [1]=>"Анна"
                                          [2]=> 5 */
sort($myarray);                     /* Структура массива:
                                          [0]=> "Анна"
                                          [1]=> "Вася"
                                          [2]=> 5 */
$myarray=array("Вася", "Анна", 5);  /* Исходный массив */
sort($myarray, SORT_STRING );       /* Структура массива:
                                          [0]=> 5
                                          [1]=>"Анна"
                                          [2]=> "Вася" */
```

При использовании функции `sort()` исходная связь "индекс — значение" теряется: элементам в отсортированном массиве назначаются новые индексы в соответствии с их новым положением. Это может оказаться нежелательным, особенно в случае массивов с символьными индексами. Сортировка, при которой эта связь сохраняется, может быть выполнена с помощью функции

`asort(`*массив* `[,` *режим*`])`

которая по возвращаемой величине и параметрам аналогична функции `sort()`, но отличается своим действием: элементы массива упорядочиваются по возрастанию значений, но индексы сохраняются прежними.

Пример:

```
$myarray=array("Вася", "Анна", 5);  /* Структура массива:
                                          [0]=>"Вася"
```

```
                                        [1]=>"Анна"
                                        [2]=> 5 */
asort($myarray);                    /* Структура массива:
                                        [1]=> "Анна"
                                        [0]=> "Вася"
                                        [2]=> 5 */
$myarray=array("Вася", "Анна", 5);  /* Исходный массив */
sort($myarray, SORT_STRING);        /* Структура массива:
                                        [2]=> 5
                                        [1]=> "Анна"
                                        [0]=> "Вася" */
```

Сортировка массива по значениям элементов с сохранением прежних индексов может понадобиться при отображении элементов, но при сохранении прежнего способа обращения к ним.

Рассмотренные выше функции `sort()` и `assort()` упорядочивают элементы массива по возрастанию их значений. Для сортировки по убыванию служат соответственно функции `rsort()` и `arsort()`.

Сортировать элементы массива можно не только по их значениям, но и по индексам. Как вы уже заметили, порядок расположения элементов в массиве, задаваемый при его создании или последующей сортировке, не обязательно соответствует порядку индексов элементов. Для сортировки массива по индексам служат функции:

- `ksort()` — сортирует элементы массива по возрастанию значений индекса;
- `krsort()` — сортирует элементы массива по убыванию значений индекса.

Пример:

```
$myarray=array(
   "name"=>"Иван",
   "family"=>"Федоров",
   "bonus"=>125.7
);
ksort($myarray);   /* Структура массива:
                        ["bonus"]=> 125.7
                        ["family"]=> "Федоров"
                        ["name"]=> "Иван" */
```

Функции `ksort()` и `krsort()` имеют такие же параметры и возвращают такие же значения, что и рассмотренные ранее функции сортировки `sort()` и `assort()`. Обратите внимание, что применение функций `ksort()` и `krsort()`

изменяет порядок расположения элементов в массиве, но сохраняет исходную связь "индекс — значение".

Внимание

Функции сортировки изменяют исходный массив, а не возвращают его отсортированный вариант. Если вам необходимо сохранить исходный массив, то для этого следует предварительно сделать его копию.

Перемещение по массиву

Нередко требуется "пробежаться" по всем элементам массива, чтобы выполнить какие-нибудь операции над ними. Это можно сделать с помощью обычных операторов цикла `while`, `do-while` и `for` *(см. разд. 4.6.8)*. Вместе с тем для этой цели в PHP имеется специальная очень удобная конструкция `foreach` (для каждого), которая имеет две синтаксические формы:

```
foreach (массив as $значение) {выражения};
foreach (массив as $индекс=>$значение) {выражения};
```

Здесь `$значение` — переменная, выбираемая по своему усмотрению, для представления значения текущего элемента массива; `$индекс` — переменная для представления значения текущего индекса элемента массива. Слово `as` в выражении, указанном в круглых скобках, является ключевым.

Конструкция `foreach` последовательно перебирает все элементы указанного ей массива, присваивая их значения переменной `$значение`, а индексы (во втором варианте синтаксиса) — переменной `$индекс`. Эти переменные доступны в коде, указанном в фигурных скобках. Код, работая с данными переменными, имеет дело с текущим на данной итерации элементом массива. На рис. 4.10 показано применение `foreach` в двух своих формах для вывода элементов массива.

Перебор элементов массива можно выполнить и с помощью специальных функций, которые реализуют перемещение указателя на текущий элемент массива:

- `current(массив)` — возвращает текущее значение элемента массива, не перемещая указатель;
- `next(массив)` — перемещает указатель на следующий элемент массива;
- `prev(массив)` — перемещает указатель на предыдущий элемент массива;
- `end(массив)` — перемещает указатель на последний элемент массива;
- `reset(массив)` — перемещает указатель на начальный элемент массива.

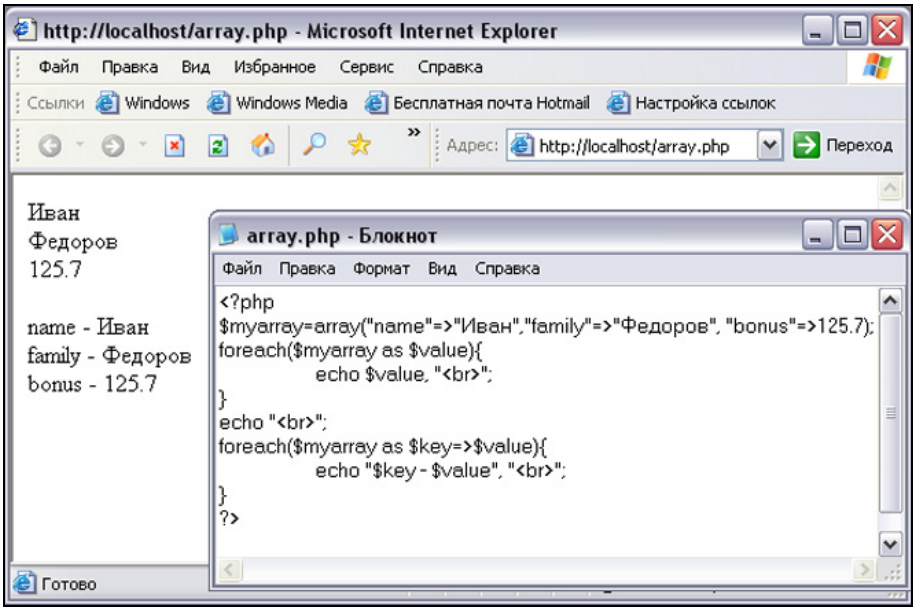


Рис. 4.10. Пример использования конструкции `foreach`

Если к массиву не применялась функция перемещения указателя, то считается, что он установлен на первый (начальный) элемент.

Пример:

```
$myarray=array("name"=>"Иван","family"=>"Федоров", "bonus"=>125.7);
$x=current($myarray);     // Иван
$x=next($myarray);        // Федоров
$x=reset($myarray);       // Иван
```

Все перечисленные выше функции перемещения указателя возвращают значение элемента массива. Однако применение функций `prev()` и `next()` может вывести указатель за пределы массива, и тогда они вернут логическое значение `false`, а не значение элемента массива. Поэтому эти функции обычно используются совместно с проверкой выхода за границы массива. Например,

```
$x=next($myarray);    // переход к следующему элементу массива
if ($x===false) {     // если выход за конец массива
  echo "Выход за конец массива";  // вывод предупреждения
} else {
  echo $x;              // вывод значения элемента массива
}
```

Обратите внимание, что здесь и в других подобных случаях необходимо использовать оператор жесткого равенства, поскольку элемент массива, возвращаемый функцией `next()`, может, вообще говоря, иметь значения `null`, `""`, `0`, `"0"` и `"false"`, но это еще не повод остановить обработку. Оператор жесткого равенства учитывает типы сравниваемых значений *(см. разд. 4.6.5)*.

Запись значений элементов массива в переменные

Чтобы значения элементов массива с числовыми индексами, начинающимися с 0, присвоить переменным из заданного списка, можно использовать следующую конструкцию:

```
list(список_переменных) = массив;
```

При этом создаются переменные, указанные в списке через запятую, и им присваиваются элементы массива с числовыми индексами: первой переменной присваивается значение элемента с индексом 0, второй переменной — значение элемента с индексом 1 и т. д.

Пример:

```
$myarray=array("a", "b", 5, 10);
list($x,$y)=$myarray;
var_dump($x, $y);  // вывод значений и типов переменных $x и $y
```

Примечание

Конструкция `list()` по своему синтаксису похожа на вызов функции, но отличается от него тем, что может быть записана слева от оператора присваивания. Однако выражению с вызовом настоящей функции нельзя присвоить никакое значение. Иначе говоря, вызов функции может встречаться только справа от оператора присваивания.

В списке переменных можно использовать элементы массива, например,

```
list($b[0], $b[1], $b[2]) = $a;
```

В этом случае происходит копирование значений элементов массива `$a` в элементы массива `$b`. Однако данная операция осуществляется, начиная с крайнего правого элемента массива `$a`, в результате чего порядок расположения элементов в массиве `$b` будет противоположен порядку элементов в массиве `$a`, хотя индексы элементов сохраняются.

Пример:

```
$a=array("a", "b", 5, 10);
list($b[0], $b[1], $b[2])=$a;
```

```
var_dump($b);  /* Структура массива $b:
                    [2]=> 5
                    [1]=> "b"
                    [0]=> "a"  */
```

Преобразование массива в текстовую строку

Чтобы значения элементов массива объединить в их строковом представлении в одну текстовую строку, можно использовать функцию

implode(*разделитель*, *массив*)

Параметр *разделитель* является строкой символов, которая вставляется между элементами массива при их объединении в текстовую строку. Если этот параметр не указан, то предполагается пустая строка "". Данная функция возвращает строку, полученную в результате преобразования.

Пример:

```
$myarray=array("Один", "Два", "Три");
$str=implode($myarray);       // строка "ОдинДваТри"
$str=implode("-", $myarray);  // строка "Один-Два-Три"
```

Преобразование текстовой строки в массив

Для записи фрагментов текстовой строки в элементы массива служит функция

explode(*разделитель*, *строка*[, *количество*])

Параметр *разделитель* является строкой символов, которая рассматривается как разделитель фрагментов строки. Необязательный параметр (о чем говорят квадратные скобки) *количество* указывает максимальное количество элементов в возвращаемом массиве (при этом последний элемент будет содержать весь остаток строки).

Если *разделитель* не содержится в строке, то функция возвращает массив, содержащий единственный элемент, значением которого является исходная строка. Если *разделитель* является пустой строкой, то функция возвращает логическое значение false.

Пример:

```
$str="Один Два Три";
$myarray=explode(" ",$str);    /* Структура массива:
                                     [0]=> "Один"
                                     [1]=> "Два"
                                     [2]=> "Три"  */
```

```
$myarray=explode("Два",$str);  /* Структура массива:
                                    [0]=> "Один "
                                    [1]=> " Три"  */
```

Другие операции над массивами

Здесь я перечислю еще несколько полезных функций для работы с массивами и приведу несколько примеров. Подробное их описание можно найти в справочном руководстве по PHP.

- `array_unique(`*массив*`)` — возвращает массив, полученный из указанного в качестве параметра путем удаления из него повторяющихся элементов.

  Пример:

  ```
  $myarray=array("a", "b", "a", "c", 5, 5);
  $x=array_unique($myarray);   /* Структура массива:
                                    [0]=> "a"
                                    [1]=> "b"
                                    [3]=> "c"
                                    [4]=>5 */
  ```

- `array_sum(`*массив*`)` — возвращает сумму значений элементов массива, указанного в качестве параметра.

  Пример:

  ```
  $myarray=array(1, 2, "3");
  $x=array_sum($myarray);  // 6
  ```

- `array_slice(`*массив*`, n1, n2)` — возвращает массив, являющийся частью исходного массива, указанного в качестве параметра. В результатный массив попадет `n2` элементов исходного массива, начиная с элемента, имеющего номер `n1`.

  Пример:

  ```
  $myarray=array(1, 2, 3, 4, 5, 6);
  $x=array_slice($myarray, 2, 3);   /* Структура массива:
                                         [0]=> 3
                                         [1]=> 4
                                         [2]=> 5  */
  $x=array_slice($myarray, -2, 3); /* Структура массива:
                                         [0]=> 5
                                         [1]=> 6 */
  ```

- `array_merge(`*массив1*`, `*массив2*`, ..., `*массивN*`)` — возвращает массив, получающийся в результате объединения двух или более массивов, указанных в качестве параметров.

Пример:

```
$a=array("a", "b", "c");
$b=array("a", "x", "y");
$x=array_merge($a, $b);  /* Структура массива:
                            [0]=> "a"
                            [1]=> "b"
                            [2]=> "c"
                            [3]=> "a"
                            [4]=> "x"
                            [5]=> "y"  */
```

- `array_diff(`*`массив1, массив2, ..., массивN`*`)` — возвращает массив, состоящий из тех элементов массива, указанного в качестве первого параметра, которых нет в остальных массивах.

  Пример:

```
$a=array("a", "b", "c");
$b=array("a", "x", "y");
$x=array_diff($a, $b);    /* Структура массива:
                             [1]=> "b"
                             [2]=> "c"  */
```

- `array_flip(`*`массив`*`)` — возвращает массив, получающийся из указанного в качестве параметра путем перестановки местами индексов и значений.

  Пример:

```
$a=array("a", "b", "c");
$x=array_flip($a);    /* Структура массива:
                         ["a"]=> 0
                         ["b"]=> 1
                         ["c"]=> 2  */
$a=array("Планета"=>"Земля", "Радиус"=>6378);
$x=array_flip($a);    /* Структура массива:
                         ["Земля"]=> "Планета"
                         [6378]=> "Радиус"  */
```

- `extract(`*`массив`*`)` — создает переменные из символьных индексов массива и присваивает им значения соответствующих элементов массива; возвращает количество созданных переменных.

  Пример:

```
$a=array("Планета"=>"Земля", "Радиус"=>6378);
$x=extract($a);
echo "Наша планета называется $Планета. Ее радиус равен $Радиус км";
```

- `compact(`*`список_имен_переменных`*`)` — возвращает массив, созданный из указанных переменных; действие этой функции противоположно действию `extract()`.

  Пример:

```
$name="Иван";
$lastname="Федоров";
$bonus=1234;
$x=compact("name", "lastname","bonus"); /* Структура массива:
                                            ["name"]=> "Иван"
                                            ["lastname"]=> "Федоров"
                                            ["bonus"]=> 1234  */
```

4.11. Глобальные предопределенные переменные

В PHP имеются предопределенные переменные типа `array` (массив) с глобальной областью видимости. Их еще называют *автоглобальными* или *суперглобальными массивами*, поскольку они доступны из любого сценария на языке PHP. Ниже приведен их список.

- `$GLOBALS` — содержит все глобальные переменные, как предопределенные, так и созданные программистом. Имена глобальных переменных в массиве `$GLOBALS` являются его символьными индексами (ключами). Поэтому, чтобы получить доступ к глобальной переменной, например, `$myvar`, достаточно использовать выражение `$GLOBALS["myvar"]`. Обратите внимание, что имя переменной в качестве индекса пишется без символа `$`.
- `$_GET` — содержит данные, переданные в сценарий на PHP как часть URL-адреса. Данные HTML-форм, переданные методом `GET`, также сохраняются в массиве `$_GET`.
- `$_POST` — содержит данные, переданные в сценарий на PHP из HTML-форм методом `POST`.
- `$_COOKIE` — содержит данные, переданные в текущий сценарий через механизм cookie.
- `$_ENV` — содержит переменные окружения, такие как название операционной системы, системный диск и др.; содержимое этого массива зависит от операционной системы.
- `$_FILES` — содержит имена файлов, загруженных методом `POST` с помощью браузера.

- $_SERVER — содержит переменные, установленные Web-сервером либо непосредственно связанные с окружением выполнения текущего сценария. Эта информация зависит от того, какой Web-сервер используется. Более того, если PHP-сценарий запущен из командной строки, то некоторые переменные заведомо будут недоступны. Таким образом, состав переменных в данном массиве может быть различным. Например, переменная DOCUMENT_ROOT содержит полное имя каталога, в котором Web-сервер ищет запрашиваемые Web-страницы; PHP_SELF — имя файла текущего сценария; REQUEST_METHOD — название метода, который использовался для доступа к странице: "GET", "POST", "HEAD", "PUT".
- $_SESSION — содержит все переменные сеанса, доступные в текущем сценарии.
- $_REQUEST — содержит все переменные, находящиеся в массивах $_GET, $_POST и $_COOKIE.

Для просмотра содержимого глобальных массивов можно воспользоваться функцией var_dump(), например, var_dump($_SERVER), или конструкцией foreach. Например, для просмотра массива $_SERVER сценарий может иметь такой вид:

```
foreach($_SERVER as $var=>$value){
  echo "$var = $value <br>";
}
```

Если, скажем, серверному сценарию из HTML-формы было передано методом POST содержимое элемента с атрибутом name="userdata", то оно может быть получено в сценарии как $_POST['userdata'].

Подробное описание содержимого глобальных массивов можно найти в справочном руководстве по PHP. Применение некоторых из них будет рассмотрено в следующей главе.

Внимание

По умолчанию глобальные массивы доступны, поскольку в конфигурационном файле php.ini имеется установка register_globals = off. Если заменить ее на register_globals = on, то глобальные массивы будут недоступны.

Примечание

В более ранних версиях PHP использовались глобальные массивы с длинными именами: $HTTP_GET_VARS, $HTTP_POST_VARS, $HTTP_COOKIE_VARS, $HTTP_ENV_VARS, $HTTP_FILES_VARS, $HTTP_SERVER_VARS, $HTTP_SESSION_VARS. В PHP 5 они также доступны, хотя и не рекомендуются к использованию. Вместе с тем установка register_long_arrays = off в конфигурационном файле php.ini позволяет отключить их. Если же вы используете старые сценарии, то оставьте установку register_long_arrays = off.

4.12. Функции

Функция — это конструкция, которая позволяет оформить блок программного кода для многократного использования. Такой блок кода (тело функции) можно озаглавить путем назначения ему имени и указать при необходимости параметры (значения переменных и выражений), с которыми он будет работать. Тело функции можно построить так, чтобы она возвращала во внешнюю программу, из которой была вызвана, некоторое значение. Впрочем, функция в PHP (как и в JavaScript) может и не требовать параметров, а также ничего не возвращать. Выражение вызова функции (имя функции с возможным указанием передаваемых ей параметров) можно использовать в простых и сложных выражениях языка PHP, если эта функция что-нибудь возвращает.

Культура программирования конкретных приложений настоятельно рекомендует создавать функции, которые могут использоваться в различных приложениях (принцип: делайте как можно более абстрактные функции). С другой стороны, для достижения наибольшей эффективности кода приложения старайтесь создавать функции, выполняющие специфические конкретные действия (принцип: действия функции должны быть как можно конкретнее). Искусство программирования выражается в компромиссных решениях, сочетающих, так или иначе, эти два противоположных принципа.

4.12.1. Пользовательские функции

Пользовательские функции — это функции, которые определяются программистом. Создать функцию означает написать в программном коде ее определение: имя, список параметров (не обязательно) и программный код (тело функции), который будет выполнен при вызове этой функции. Итак, с программной точки зрения с функцией связано две вещи: определение и вызов. Рассматривая встроенные функции, мы всегда имели дело с выражениями вызова этих функций. Их определения были скрыты от нас, поскольку созданы не нами.

Определение (описание) функции начинается ключевым словом `function`. Точнее, описание функции имеет следующий синтаксис:

```
function имя_функции(параметры){
  код
}
```

Имя функции выбирается так же, как и имя переменной, за исключением того, что недопустимо первым символом в имени функции использовать символ `$`. Недопустимо также использовать в качестве имени ключевые слова

языка PHP. За именем функции обязательно стоит пара круглых скобок. Программный код (тело) функции заключается в фигурные скобки. Они определяют группу выражений, которые относятся к коду именно этой функции. Если функция принимает параметры, то список их имен (идентификаторов) указывается в круглых скобках около имени функции. Имена параметров выбираются согласно тем же требованиям, что и имена обычных переменных. Если параметров несколько, то в списке они разделяются запятыми. Если параметры для данной функции не предусмотрены, то в круглых скобках около имени функции ничего не пишут.

Если требуется, чтобы функция возвращала некоторое значение, то в ее теле (в программном коде, заключенном в фигурные скобки) используется оператор возврата `return` с указанием справа от него того, что следует возвратить. В качестве возвращаемой величины может выступать любое выражение: простое значение, имя переменной или вычисляемое выражение. Оператор `return` может встречаться в коде функции несколько раз. Если оператор `return` не использовать или не указывать справа от него возвращаемую величину, то функция ничего не будет возвращать.

Пример:

```
function S_rectangle($width, $height) {
  $S=$width* $height;
  return $S;
}
```

Здесь функция с именем `S_rectangle` принимает два параметра — `$width` и `$height`, вычисляет и возвращает произведение их значений.

Определение функции, будучи помещенным в программу, усваивается интерпретатором, но сама функция (ее код или тело) не выполняется. Чтобы выполнить функцию, определение которой задано, необходимо написать в программе выражение вызова этой функции. Оно имеет следующий синтаксис:

имя_функции(*параметры*);

Пример:

```
S_rectangle(3, 5);   // возвращаемое значение равно 15
```

Имя функции в ее вызове должно совпадать с именем ранее определенной функции. Однако совпадение с точностью до регистра не обязательно. Например, выражения `S_rectange(3, 5)` и `s_Rectangle(3, 5)` эквивалентны.

Примечание

В JavaScript имена функций и в определении, и в вызове должны совпадать с точностью до регистра.

Параметры, если они заданы в определении функции, в вызове этой функции представляются конкретными значениями, переменными или выражениями.

Пример:

```
$x=5;
$S=S_rectangle(3, $x+2);
```

Параметры функции могут быть переданы по значению и по ссылке. По умолчанию они передаются по значению. Это означает, что изменение в теле функции значения переменной, переданной в качестве параметра, никак не отразится на значении этой же переменной во внешнем коде. Кроме того, любая переменная, созданная в теле функции, является локальной, т. е. видимой только в пределах ее кода.

Пример:

```
<?php
  $x=5;
  function myfunc($arg){
    $arg=$arg+2;
    $y=10;
    return $arg;
  }
  myfunc($x);  // возвращает 7
  echo $x;     // выводит 5 (прежнее значение)
  echo $y;     /* ничего не выводится, т. к.
                  значение $y вне функции myfunc() равно null
?>
```

При передаче параметров по значению в функции создаются локальные переменные с такими же именами, что и имена формальных параметров, указанных в определении функции. Это — локальные переменные, отличные от внешних, даже если их имена совпадают. Следующий код эквивалентен приведенному выше.

```
<?php
  $x=5;
  function myfunc($x){
    $x=$x+2;
    $y=10;
    return $x;
  }
  myfunc($x);  // возвращает 7
```

```
  echo $x;      // выводит 5 (прежнее значение)
  echo $y;      /* ничего не выводится, т. к.
                   значение $y вне функции myfunc() равно null
?>
```

Чтобы изменения параметров, произведенные в теле функции, были видны и за пределами ее тела, необходимо передавать их по ссылке. Для указания того, что параметр передается по ссылке, перед его именем в списке параметров в определении функции следует написать символ `&` (амперсанд):

```
function имя_функции(&$arg1, &$arg2, ...){
  код
}
```

Пример:

```
<?php
  $x=5;
  function myfunc(&$arg){
    $arg=$arg+2;
    return $arg;
  }
  myfunc($x);
  echo $x;    // 7 (новое значение)
?>
```

При передаче параметров по ссылке формальному параметру, указанному в определении функции, передается не значение, а адрес внешней переменной.

Примечание

В JavaScript параметры передаются функции только по значению.

При передаче функции большего количества параметров, чем их предусмотрено в определении, лишние параметры будут просто проигнорированы. Если передается меньшее количество, то пропущенным параметрам присваивается значение `null` и при этом может быть выведено соответствующее предупреждающее сообщение, что, однако, не приведет к остановке выполнения кода. Вместе с тем, в определении функции можно указать значения параметров, принимаемые по умолчанию. Это полезно на тот случай, если вы укажете в вызове функции меньшее количество параметров, чем их предусмотрено в определении. Для задания значений по умолчанию используется оператор присваивания непосредственно в списке формальных параметров определения функции.

Пример:

```
<?php
  function S_rectangle($width=10, $height=5) {
    $S=$width* $height;
    return $S;
  }
  echo S_rectangle();  // 50
  echo "<br>";
  echo Srectangle(3);  // 15
?>
```

Внимание

При задании для параметров в определении функций значений по умолчанию, последние должны быть простыми значениями, а не переменными, вычисляемыми выражениями или вызовами функций и методов объектов.

Если в списке используются параметры без значений по умолчанию, а также параметры с заданными значениями по умолчанию, то последние должны располагаться в правой части списка, т. е. за теми параметрами, для которых значения по умолчанию не предусмотрены.

При создании функций, способных работать с переменным по длине списком параметров, можно применять следующие встроенные функции.

- `func_num_args()` — возвращает количество параметров, переданных функции, из которой она была вызвана.

  Пример:

```
<?php
function S_rectangle($width, $height) {
  if (func_num_args()==0){  // если нет параметров
    $width=10;
    $height=10;
  }
  if (func_num_args()==1){  // если один параметр
    $height=10;
  }
  return $width* $height;
}
echo S_rectangle();   // 100
echo "<br>";
echo S_rectangle(2);  // 20
?>
```

- `func_get_args()` — возвращает массив значений параметров, переданных функции, из которой она была вызвана.

  Пример:

```
<?php
function S_rectangle($width, $height) {
  /* значения параметров по умолчанию: */
  $width=10;
  $height=10;
  $args=func_get_args();          // массив значений параметров
  $narg=count(func_get_args());  // длина массива значений параметров
  if ($narg==1) $width=$args[0]; // если один параметр
  if ($narg==2) {$width=$args[0]; $height=$args[1];}
                                  // если два параметра
  return $width* $height;
}
echo S_rectangle();      // 100
echo "<br>";
echo S_rectangle(2);     // 20
echo "<br>";
echo S_rectangle(2, 3);  // 6
?>
```

- `func_get_arg(`*`номер_параметра`*`)` — возвращает значение параметра, указанного с помощью его номера. Если *`номер_параметра`* превышает длину списка переданных параметров, то данная функция возвращает `false`. Обычно эта функция используется совместно с `func_get_args()` и `func_num_args()`.

  Пример:

```
<?php
  function S_rectangle($width, $height) {
    echo "width=".func_get_arg(0)." height=".func_get_arg(1);
  }
  S_rectangle();      // width= height=
  echo "<br>";
  S_rectangle(2);     // width=2 height=
  echo "<br>";
  S_rectangle(2, 3);  // width=2 height=3
?>
```

Внимание

При вызове функции с некорректным количеством параметров (даже если вы обрабатываете эти ситуации в теле функции) может выводиться предупреждающее сообщение "Warning: Missing argument...". Однако программа при этом продолжает выполняться. Если вы желаете предотвратить вывод подобных сообщений, то перед вызовом функции следует указать символ `@`, например, `@S_rectangle()`.

4.12.2. Переменные функции

Если рядом с именем переменной написать круглые скобки, то интерпретатор PHP возьмет значение этой переменной и попытается выполнить одноименную функцию. Иначе говоря, если строковое значение переменной совпадает с именем определенной функции и рядом с переменной указаны круглые скобки, то будет совершена попытка выполнить эту функцию. В круглых скобках при необходимости можно указать список параметров.

Пример:

```
<?php
  function S_rectangle($width, $height) {  // определение функции
    return $width*$height;
  }
  $x="S_rectangle";  // значение переменной $x совпадает с именем функции
  echo $x(2, 3);     // 6
?>
```

Имена функций можно хранить и в массивах. Например,

```
<?php
  $afuncname=array("sqrt","sin","cos");  // массив имен функций
  $x=0.5;
  $i=0;
  while ($i<count($afuncname)){
    echo $afuncname[$i]($x);
    echo "<br>";
    $i++;
  }
?>
```

Данный код выводит на экран значения математических функций `sqrt(0.5)`, `sin(0.5)` и `cos(0.5)`, имена которых заданы в массиве `$afuncname`.

4.12.3. Встроенные функции

В PHP есть большое множество *встроенных функций*. Некоторые из них уже были рассмотрены в предыдущих разделах. Подробное описание других функций можно найти в справочном руководстве по PHP. Следует заметить, что некоторые из встроенных функций доступны при обычной установке PHP, другие требуют особых, хотя и несложных, настроек модуля PHP. Все рассмотренные ранее встроенные функции PHP доступны всегда без специфических настроек модуля PHP. Если обратиться к справочной информации по функциям PHP, то там можно найти сведения об их доступности. Для доступа к некоторым функциям, например работы с базами данных, графической информацией и др., требуется подключение специальных библиотек. Подробнее об этом рассказано в *приложении 2*.

4.13. Объекты

Язык PHP 5 является объектно-ориентированным языком программирования. Объектно-ориентированное программирование (ООП) предполагает особый синтаксис и подходы к разработке и анализу программ. Основными элементами программ в ООП являются *объекты*. Обычно они, так или иначе, соответствуют предметам моделируемой посредством программ действительности. Например, автомобиль, банковский счет, форму для ввода данных и персонажи компьютерной игры можно рассматривать как объекты. С формальной же точки зрения объект является неким контейнером, в котором сгруппированы свойства и методы объекта. Свойства и методы можно понимать как переменные и функции, связанные с объектом или, если хотите, присущие ему.

Наряду с понятием объекта в ООП используется понятие класса. *Класс* — это описание, своего рода шаблон, по которому создаются объекты. Иногда вместо термина "класс" применяют термин "объект", а вместо термина "объект" — "экземпляр объекта". В этом разделе мы будем говорить о классах и объектах.

С точки зрения систем программирования класс является обобщением таких понятий, как функция и структура данных. И то и другое объединяются понятием и соответствующей синтаксической конструкцией, которые называются классом или объектом.

Между классами может быть установлено отношение наследования. Мы можем создать класс А как контейнер, содержащий описание свойств некоторого моделируемого предмета действительности. Эти свойства могут быть различного уровня. Так, они могут восприниматься нашими органами чувств более или менее непосредственно, обнаруживаться с помощью датчиков и измерительных приборов, а также "вычисляться" умом посредством усвоенных

концепций и теорий. Класс А может быть более или менее абстрактным (общим). Возможно, мы захотим уточнить его или, другими словами, сделать его менее абстрактным, но, помимо своих специфических свойств, содержащим и некоторые важные исходные свойства. Речь идет о создании нового класса В, связанного каким-то образом с исходным. Если класс В является наследником (потомком) класса А, то он имеет (наследует) все его свойства и методы, а также, возможно, обладает и собственными свойствами и методами. При этом класс А называют *родительским* или *базовым* для класса В. Например, можно создать два класса автомобилей: Грузовик и Легковой. Ни один из них не является наследником другого. Поскольку между ними много общего, можно сначала создать общий класс Автомобиль, а затем два класса-наследника — Грузовик и Легковой.

Примечание

PHP 5 не в полной мере реализует принципы ООП в отличие от, например, классического объектно-ориентированного языка C++. Так, PHP не поддерживает полиморфизм (в одном классе не может быть двух и более одноименных методов с различными наборами параметров), а также множественное наследование (каждый класс не может иметь более одного родительского класса). Тем не менее ООП-возможности PHP 5 существенно шире, чем в JavaScript.

4.13.1. Определение класса

Определение класса, который не является наследником другого класса, имеет следующий синтаксис:

```
class имя_класса {
  определение_свойств
  определение_методов
}
```

Чтобы определить *класс-наследник*, используется такой синтаксис:

```
class имя_класса-наследника extends имя_класса-родителя {
  определение_свойств
  определение_методов
}
```

В объекте, созданном на основе класса `имя_класса-наследника`, доступны все свойства и методы класса `имя_класса-родителя`. Однако обратное утверждение не верно: в объекте родительского класса свойства и методы его наследников не доступны.

Свойства в определении класса задаются как обычные переменные. При этом вы можете использовать оператор присваивания, чтобы установить для свойств значения по умолчанию. Значения должны быть простыми (различных типов), а не задаваться посредством вычисляемых выражений. Вместе с тем, можно и рекомендуется использовать ключевое слово `var` для объявления переменных. Вот пример определения класса с несколькими свойствами-переменными:

```
class myobject{
  var $x;
  var $y=5;
  var $color="красный"
  var $address("Санкт-Петербург", "Литейный пр. ", 22, 3);
}
```

Внимание

Спецификатор (ключевое слово) `var` перед именем свойства означает доступность этого свойства из-за пределов кода класса. Вместо `var` можно использовать спецификатор `public` (публичный, общедоступный). Допустимы также и другие спецификаторы: `private` и `protected`, но они задают ограничение доступа к свойствам *(см. разд. 4.13.3)*. При объявлении свойств указание спецификатора обязательно.

Методы объекта также являются его свойствами, но задаются не как переменные, а как функции. Методы обычно используются, чтобы изменить значения свойств объекта и/или произвести какие-нибудь другие действия. Если в методе (функции) требуется обратиться к какому-нибудь свойству (переменной) этого же объекта, то перед именем этого свойства следует указать префикс `$this->`. Например, выражение `$this->color` означает обращение к свойству `$color` этого же класса (`this` — этот). Метод задается в определении класса как обычная функция с использованием ключевого слова `function`. В следующем примере описывается простой класс для создания счета.

```
class myaccount {
  var $summa=0;       // остаток на счете
  function addsum($sum) {  // добавление к сумме на счете
    $this->summa = $this->summa + $sum;
    echo "Добавлено $sum рублей";
  }
}
```

Данный класс имеет одно свойство `$summa` и один метод `addsum()`. Класс предназначен для хранения некоторого числа, понимаемого как остаток

денежной суммы на некотором счете, т. е. как состояние счета. С помощью метода `addsum()` можно изменить текущее состояние счета, указав в качестве параметра число, которое следует прибавить к остатку на счете (если это число положительное) или вычесть из остатка (если число отрицательное).

В определении класса при необходимости можно задать один особый метод, называемый *конструктором*. Конструктор выполняется при создании объекта на основе данного класса, хотя объект можно создать и не определяя конструктор явно. Обычно конструктор определяют явным образом, если при создании объекта требуется сразу же выполнить некоторый код, установить начальные значения его свойств и/или передать значения его свойств как параметры. Формально конструктор является функцией, имя которой не произвольно, а является ключевым словом `__constuct` (с двумя ведущими символами подчеркивания) или же должно совпадать с именем класса.

Пример:

```
class account {
  var $summa=0;             // свойство
  function __construct(){   // конструктор
    $x=1000;
    $this->summa=$x;
    echo "На вашем счете $x руб.";
  }
  function addsum($sum) {   // метод: добавление к сумме на счете
    $this->summa = $this->summa + $sum;
    echo "Добавлено $sum рублей";
  }
}
```

Здесь конструктор устанавливает текущее значение счета `$summa` (хотя значение по умолчанию для этого свойства уже установлено) и выводит соответствующее сообщение. Заметим, что в данном примере в качестве имени конструктора можно было бы выбрать `account`, т. е. имя, совпадающее с именем класса.

Как уже отмечалось, конструктор в явном виде используется еще и для того, чтобы передать создаваемому объекту значения его свойств подобно тому, как это делается в случае обычных функций. Вот пример:

```
class account {
  var $summa;                    // свойство
  function __construct($summa){  // конструктор
    $this->summa=$summa;
```

```
    echo "На вашем счете $this->summa руб.";
  }
  function addsum($sum) {  // метод: добавление к сумме на счете
    $this->summa = $this->summa + $sum;
    echo "Добавлено $sum рублей";
  }
}
```

Здесь в определении класса объявлено свойство `$summa`. Конструктор выполняет две вещи:

- обеспечивает передачу создаваемому объекту значения свойства `$summa` как параметра;
- осуществляет вывод сообщения о состоянии счета на момент создания объекта.

Теперь рассмотрим создание двух классов — родительского с именем `account` и его наследника с именем `myaccount` (листинг 4.6).

Листинг 4.6. Пример родительского класса и его наследника

```
<?php
class account {
  var $summa;              // объявление свойства
  function addsum($sum) {  // метод: добавление к сумме на счете
    $this->summa = $this->summa + $sum;
  }
}

class myaccount extends account{
  var $name;                          // объявление свойства
  function __construct($nam, $sum){   // конструктор
    $this->name=$nam;
    $this->summa=$sum;
  }
  function set_name($name) {  /* метод: установка имени владельца
                                 счета */
    $this->name = $name;
  }
}
?>
```

Класс `account` содержит только свойство `$summa` для хранения суммы счета и метод `addsum()` для его пополнения. Класс `myaccount` уточняет (дополняет) родительский класс `account` свойством `$name` для хранения имени владельца счета и методом `set_name()` для его установки. Конструктор класса `myaccount` обеспечивает установку имени и суммы на счете при создании объекта `myaccount`. Создание и применение объектов будет рассмотрено в следующем разделе.

Разумеется, рассмотренная задача могла быть решена и на основе одного, а не двух классов. Скорее всего, так и следовало бы поступить, но рассмотренный выше код призван лишь проиллюстрировать наследование свойств классов.

4.13.2. Применение объектов

Если в сценарии находится определение класса, то его можно использовать для создания объектов. Определение класса может располагаться в сценарии непосредственно либо находиться в другом файле, содержимое которого включается в текущий сценарий с помощью функции `include()`.

Объект создается на основе определенного класса с помощью оператора `new`:

```
$имя_переменной = new имя_класса(параметры);
```

Оператор присваивания здесь используется, чтобы присвоить переменной ссылку на созданный объект, которую затем удобно применять для доступа к его свойствам и методам. Эта переменная имеет тип `object`. Для доступа к свойствам и методам объекта применяется так называемый стрелочный (а не точечный, как в JavaScript) синтаксис:

```
объект->свойство;
объект->метод();
```

Пример:

```
<?php
  class account {
  ... // код из листинга 4.5
  }
  class myaccount extends account{
  ... // код из листинга 4.5
  }
  $w=new myaccount("Вася",1000);   // Положили Васе на счет 1000
  $w->addsum(500);                 // Добавили 500
  $w->set_name("Петя");            // Заменили имя на Петя
```

```
  echo "Имя: $w->name".". Сумма: $w->summa рублей";  /* Вывод строки:
                                        "Имя: Петя. Сумма: 1500 рублей"  */
?>
```

В описании класса `myaccount` (см. листинг 4.6) конструктор организует прием значений двух параметров — имя и начальное значение счета. В результате выполнения выражения

```
$w=new myaccount("Вася",1000);
```

свойствам `$name` и `$summa` будет присвоены значения "Вася" и 1000 сразу же при создании объекта. Кроме того, переменной `$w` присвоена ссылка на созданный объект, которая была использована в выражениях вызова методов:

```
$w->addSum(500);
$w->set_name("Петя");
```

Обратите внимание, что первый метод определен в родительском классе, а второй — в его классе-наследнике. Тем не менее мы можем вызвать их как методы объекта класса-наследника `myaccount`.

4.13.3. Ограничение доступа к свойствам и методам

Рассмотренные выше свойства и методы класса доступны из сценария как свойства и методы созданного на основе этого класса объекта. Иначе говоря, при синтаксически правильном обращении свойства и методы объекта доступны из программного кода, являющегося внешним по отношению к коду описания класса. Однако в ряде случаев требуется, чтобы некоторые свойства и/или методы объекта были недоступны из внешнего кода. Так, например, считается, что прямой доступ к свойствам-переменным класса — плохой стиль программирования. Лучше обеспечивать доступ к свойствам извне с помощью специальных методов класса.

В PHP можно установить ограничение доступа к свойствам (переменным и функциям) класса двояко посредством следующих спецификаторов (ключевых слов):

- `private` (закрытый, частный) — запрещает доступ к свойствам и методам класса из-за его пределов (сценариев и других классов);
- `protected` (защищенный) — запрещает доступ к свойствам и методам класса из-за его пределов, за исключением его классов-наследников.

Примеры:

```
private $x;
protected $myvar;
protected function myfunc($arg){...}
```

Перепишем определение класса `account` (см. листинг 4.6) так, чтобы нельзя было воспользоваться его методом непосредственно:

```
class account {
  var $summa;                        // объявление свойства
  protected function addsum($sum){ /* защищенный метод:
                                       добавление к сумме на счете */
    $this->summa = $this->summa + $sum;
  }
}
```

Теперь попытка выполнить, например, код

```
$x=new account();
$x->addsum(500);
```

приведет к выводу сообщения об ошибке. Однако изменить значение свойства `$summa` можно из кода класса-наследника `myaccount`, если добавить к нему соответствующий метод:

```
class myaccount extends account{
  var $name;  // объявление свойства
  function __construct($nam, $sum){    //  конструктор
    $this->name=$nam;
    $this->summa=$sum;
  }
  function set_name($name) {  // метод: добавление к сумме на счете
    $this->name = $name;
  }
  function add_sum($sum) {   // метод: добавление к сумме на счете
    $this->addsum($sum);     // вызов защищенного метода
  }
}
```

Здесь метод `add_sum($sum)` просто вызывает защищенный метод `addsum($sum)` родительского класса `account`.

Внимание

Спецификатор `var` означает, что следующее за ним свойство не имеет ограничений доступа. Вместо него можно использовать эквивалентный спецификатор `public`.

4.13.4. Клонирование и удаление объектов

Для клонирования (копирования) объекта используется оператор

```
$объект_копия = clone $исходный_объект;
```

Этот оператор вызывает специальный метод `__clone()` (с двумя лидирующими символами подчеркивания), который по умолчанию просто создает новый объект со свойствами исходного.

Пример:

```
$w->new mycount("Саша", 500); // см. листинг 4.6.
$q-> clone $w;                // $q — копия объекта $w
$q->add_sum(500);
$q->set_name("Маша");
echo "Имя: $q->name".". Сумма: $q->summa";
```

Однако метод `__clone()` можно переопределить подобно тому, как это делается с конструктором (методом `__construct()`).

Удалить ранее созданный объект можно с помощью функции `unset($объект)`.

Пример:

```
$w->new mycount("Саша", 500);   // создание объекта (см. листинг 4.6).
unset($w);                      // удаление объекта $w
```

Нередко при удалении объекта требуется выполнить какие-то действия, например, записать данные в файл или базу данных, закрыть файлы и т. п. В подобных случаях можно в классе определить код специального метода `__destruct()` (два символа подчеркивания), который выполнится перед тем, как объект будет удален. Этот метод называют *деструктором*.

Пример:

```
class account {
  var $summa;       // объявление свойства
  function _destructor(){
  echo "Объект уничтожен";
}
  protected function addsum($sum){ /* защищенный метод:
                                      добавление к сумме на счете */
    $this->summa = $this->summa + $sum;
  }
}
```

4.13.5. Использование методов несозданных объектов

Чтобы обратиться к методу или свойству класса, объект для которого еще не создан, можно использовать следующий синтаксис:

```
имя_класса::свойство();
имя_класса::метод();
```

В следующем примере определен класс с единственным методом, выводящим сообщение, и далее вызов этого метода без предварительного создания объекта данного класса.

```
<?php
  class myclass {
    function mymethod(){
      echo "Привет из класса myclass";
    }
  }
  myclass::mymethod();
?>
```

4.13.6. Обработка исключений

Обработка ошибок в программах на языке PHP происходит на основе так называемого механизма исключительных ситуаций, который использует встроенный класс `Exception` (исключение). Вы можете определить некоторые нежелательные (исключительные) ситуации и обработать их по своему усмотрению. Если этого не делать, то при возникновении ошибок интерпретатор PHP обработает их некоторым стандартным образом с выводом соответствующих сообщений.

Обратимся в качестве примера к определению класса `account`, приведенному в листинге 4.6. Метод `addsum()` в этом классе позволяет изменять состояние счета (свойство `$summa`). Предположим, что состояние счета не должно быть отрицательным. Иначе говоря, отрицательное значение свойства `$summa` определяется нами как исключительная ситуация, которую следует обработать особым образом. В листинге 4.7 обработка данной ситуации состоит в выводе сообщения и прекращении выполнения сценария.

Листинг 4.7. Пример родительского класса и его наследника

```
<?php
class account {
  var $summa;                    // объявление свойства
```

```
  function addsum($sum) {     // метод: добавление к сумме на счете
    $this->summa = $this->summa + $sum;
    try {
      if ($this->summa < 0) { // если исключение
        throw new Exception ("Вы должник!");
                    }
    } catch (Exception $e) {
      exit($e->getMessage());
    }
  }
}
$w=new account();   // создание объекта класса account
$w->addsum(-100);   // снять со счета 100
?>
```

Здесь в коде метода `addsum()` содержатся блоки:

- `try` (попытка) — содержит оператор условия `if`; если результат проверки условия равен `true`, то генерируется исключение и выполняется выражение, следующее за ключевым словом `throw`, в данном случае — создание нового объекта `Exception` с передачей ему текста сообщения;
- `catch` (перехват) — указывается объект класса `Exception`, созданный для данного исключения, а также блок кода, который следует выполнить, если исключительная ситуация возникла. В примере это вызов метода `getMessage()`, который возвращает сообщение о ситуации, хранящееся в классе `Exception`, вывод этого сообщения и прекращение выполнения сценария с помощью функции `exit()`. Если исключение не сгенерировано, то блок кода `catch` не выполняется.

4.13.7. Пример класса формы

Рассмотрим в качестве примера программное создание HTML-формы на основе применения класса, содержащего следующие свойства и методы:

- `$afld` — массив полей ввода данных;
- `$action` — значение атрибута `action` тега `<FORM>`, которое используется для указания серверной программы, обрабатывающей данные формы;
- `$submitlabel` — надпись на кнопке типа `submit` для отправки данных на сервер;
- `$nfld` — количество полей в форме;

- `__construct()` — конструктор для присвоения значений свойствам `$action` и `$submitlabel` при создании объекта формы;
- `addfld()` — метод, добавляющий поле в форму;
- `showform()` — метод, отображающий форму.

Таким образом, создаваемая форма будет содержать кнопку типа `submit` и поля ввода данных типа `input`, которые при необходимости можно добавлять к форме. В листинге 4.8 приведены определение класса `Form` и сценарий, добавляющий в форму два поля ввода данных, а затем отображающий ее в окне браузера.

Листинг 4.8. Пример создания HTML-формы на основе класса

```
<?php
class Form {
/* Класс формы */
  var $afld=array();
  var $action;
  var $submitlabel="Отправить";
  var $nfld=0;
/* Конструктор */
  function __construct($action, $submitlabel){
    $this->action=$action;
    $this->submitlabel=$submitlabel;
  }
/* Добавление полей */
  function addfld($name, $label){
    $this->afld[$this->nfld]["name"]=$name;
    $this->afld[$this->nfld]["label"]=$label;
    $this->nfld=$nfld + 1;
  }
/* Отображение формы */
  function showform(){
    echo "<FORM action={$this->action} method=POST>";
    for($i=1; $i<=count($this->afld); $i++){
      echo "{$this->afld[$i-1]['label']}: ";
      echo "<input type=text
         name={$this->afld[$i-1]['name']}><br>";
    }
    echo "<input type=submit value='$this->submitlabel'>";
```

```
    echo "</FORM>";
  }
}
/* Создание объекта класса Form */
$f=new Form("http://localhost/myform.php", 'Отправить данные');
/* Добавление полей */
$f->addfld("username","Имя");
$f->addfld("address","Адрес");
/* Отображение формы */
$f->showform();
?>
```

4.14. Выполнение PHP-кода в текстовых строках

В текстовой строке можно написать код на языке PHP, а затем, когда это потребуется, выполнить его. Например, такие коды могут храниться в базе данных. В этом случае их можно считать в переменную для последующего выполнения.

Для выполнения PHP-кода в текстовой строке служит функция `eval(`*`строка`*`)`.

Пример:

```
<?php
  $x=2; $y=3;
  $str = '$z = $x + $y;';
  eval($str);  // вычисляет выражение $z = $x + $y;
  echo $z;     // выводит число 5
?>
```

Обратите внимание, что значение переменной `$str` обрамлено одинарными кавычками, чтобы заключенные в них переменные не интерпретировались. Функция `eval($str)` вычисляет выражение, являющееся значением переменной `$str`.

Функция `eval(`*`строка`*`)` может возвращать значение, указанное в строке справа от оператора `return`, если, разумеется, он используется. В противном случае возвращается `null` или `false` (при возникновении ошибки преобразований). Если возникает фатальная ошибка, происходит прекращение работы всего сценария.

Пример:

```
<?php
  $x=2; $y=3;
  $str = '$z = $x + $y; return $z;';
  echo eval($str);  // выводится возвращаемое функцией eval() значение $z
?>
```

Из приведенных выше примеров видно, что с помощью функции `eval()` можно создавать новые переменные и присваивать им значения. Вот еще один пример:

```
<?php
  $x="newvar";
  eval('$'.$x.'=123;');
  echo $newvar;  // выводится число 123
?>
```

Примечание

Аналогичная функция `eval()` имеется и в JavaScript *(см. разд. 2.6.1)*. Особенность ее заключается в возвращаемом значении.

Глава 5

Серверные сценарии на PHP

В этой главе мы рассмотрим основные принципы и примеры создания серверных сценариев на языке PHP для решения типичных задач разработки Web-сайтов, таких как прием и обработка данных форм, отправленных браузером пользователя, создание счетчика посещений Web-страницы, баннера и гостевой книги. Решая эти прикладные задачи, мы попутно рассмотрим и более общие вопросы взаимодействия сервера с клиентом, работу с файловой системой, базами данных и графикой. Впрочем, именно решение более общих задач является главной целью данной главы. Конкретные прикладные проекты — лишь примеры их воплощения.

5.1. Получение данных от клиента

Данная тема является самой важной для понимания взаимодействия клиента и сервера — двух сторон одного процесса. Несмотря на кажущуюся простоту, это первый шаг в область двустороннего программирования.

В этом разделе мы остановимся на получении сервером данных из клиентских HTML-форм. Но сначала рассмотрим передачу содержимого обычных полей формы, а затем — загрузку на сервер файлов, расположенных на компьютере клиента.

5.1.1. Получение данных из HTML-форм

Браузер пользователя (клиент) может отправить на Web-сервер данные, содержащиеся в элементах HTML-формы. В атрибуте `action` HTML-тега `<FORM>` указывается URL-адрес файла со сценарием, который должен принять эти данные и, быть может, как-то их обработать. Это может быть сценарий на языке PHP. Щелчок на кнопке специального типа `submit` (Отправить), вставленной в форму, приводит к отправке на сервер данных, содержащихся

в элементах формы. Подробно формы обсуждались в *разд. 1.2.7* и *3.16*, а здесь мы остановимся на задаче приема данных сервером (точнее, серверным сценарием).

Получив данные от клиента, серверный сценарий может распорядиться ими так или иначе, например, просто сохранить в файле или базе данных, обработать каким-либо образом и в зависимости от результата послать клиенту некоторое сообщение, сгенерировать HTML-документ или сделать что-нибудь иное. Сейчас для нас важнее узнать, как данные клиентской формы воспринимаются серверным сценарием.

Данные HTML-форм могут быть переданы на сервер методами `GET` и `POST`. Метод `GET` добавляет передаваемые данные к URL-адресу сценария-обработчика, и, следовательно, их можно увидеть в адресной строке браузера. Иногда это не желательно, что и вынуждает использовать метод `POST`. Кроме того, методом `POST` можно передавать сколь угодно большие по объему данные. Поэтому он и наиболее часто используется.

Напомним, что метод передачи указывается в атрибуте `method` тега `<FORM>`. Данные, полученные сервером при их передаче тем или иным методом, сохраняются в суперглобальных встроенных массивах `$_GET` и `$_POST` соответственно *(см. разд. 4.11)*. Если в тегах элементов формы (`<INPUT>`, `<TEXTAREA>` и т. п.) вы указали значение атрибута `name` (имя элемента), то сервер получит соответствующие пары вида *`имя_элемента=значение_элемента`*. При этом *`имя_элемента`* (значение атрибута `name`) станет символьным индексом (ключом) массива `$_GET` или `$_POST` в зависимости от того, каким методом передавались данные. По индексу массива, как известно, можно получить значение его элемента, т. е. значение соответствующего элемента формы. Например, чтобы получить содержимое элемента формы с именем (значением атрибута `name` HTML-тега этого элемента) `myelement`, достаточно обратиться к элементу массива `$_POST['myelement']` или `$_GET['myelement']` в зависимости от использованного метода передачи.

Внимание

Чтобы получить данные формы независимо от метода передачи (`GET`, `POST` или cookie), можно воспользоваться суперглобальным массивом `$_REQUEST`. Указав в качестве символьного индекса имя (значение атрибута `name`) элемента формы, вы получите значение этого элемента.

В листинге 5.1 в качестве примера приводится содержимое файла Форма.php, который предоставляет браузеру клиента некоторую форму и отображает введенные пользователем данные. Точнее, вызов пользователем данного файла приводит к загрузке в браузер формы с двумя полями для ввода дан-

ных и кнопкой типа `submit`. Щелчок на этой кнопке отправляет на сервер данные, введенные пользователем, и отображает полученные данные как список имен элементов формы и их значений (рис. 5.1).

Листинг 5.1. Получение и отображение данных HTM-формы (Форма.php)

```
<FORM action="http://localhost/forma.php" method=POST>
  <INPUT type="text" name="name" value="">
  <INPUT type="text" name="message" value="">
  <INPUT type="submit" name="submit" value="Отправить">
  </FORM>
<?php
  /* Вывод содержимого массива $_POST */
  foreach($_POST as $key=>$value){
    echo "$key=$value <br>";
  }
?>
```

Если в атрибуте `method` тега `<FORM>` указать значение `GET`, то для получения данных необходимо воспользоваться суперглобальным массивом `$_GET` или `$_REQUEST`. Разумеется, в PHP-сценарии вместо вывода элементов массива `$_POST` можно сделать что-либо более интересное или полезное. Здесь мы лишь демонстрируем возможность доступа к полученным от клиента данным.

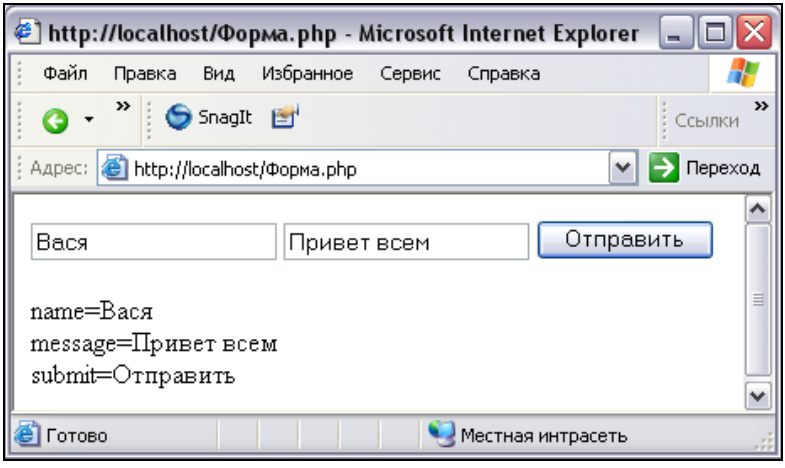


Рис. 5.1. Форма для ввода данных и их отображение с помощью сценария

Теперь рассмотрим подробнее, что именно происходит при реализации данного примера:

1. При загрузке в браузер файла Форма.php (например, **http://localhost/Форма.php**) в окне браузера отображаются элементы формы (два поля ввода и кнопка типа `submit`): HTML-код этого файла отображается, а PHP-код ничего не выводит, поскольку в данный момент массив `$_POST` пуст.
2. После ввода в элементы формы каких-либо данных и щелчка на кнопке типа `submit` (Отправить) эти данные передаются серверу — исходному файлу Форма.php. Теперь массив `$_POST`, всегда доступный любому серверному PHP-сценарию, уже не пуст. Сервер возвращает браузеру клиента форму с пустыми полями, а затем то, что посылает оператор `echo` в PHP-сценарии — индексы и значения массива `$_POST`.

Обратите внимание, что в рассмотренном выше примере HTML-форма была создана с помощью HTML-кода непосредственно. Однако то же самое можно сделать исключительно с помощью PHP-сценария, используя оператор `echo` (листинг 5.2). В этом сценарии с помощью оператора `echo` в окно браузера выводится строка, содержащая теги формы, которые интерпретируются браузером. А именно, в окне браузера отображается форма в соответствии с принятыми тегами.

Листинг 5.2. Форма, созданная PHP-сценарием

```
<?php
  echo '<FORM action="http://localhost/Форма.php" method=POST>
  <INPUT type="text" name="name" value="">
  <INPUT type="text" name="message" value="">
  <INPUT type="submit" name="submit" value="Отправить">
  </FORM>';
  foreach($_POST as $key=>$value){
    echo "$key=$value <br>";
  }
?>
```

В форме может содержаться набор элементов, допускающих множественный выбор (т. е. выбор сразу нескольких значений). Типичным примером является набор флажков, задаваемых тегами `<INPUT type=checkbox>`. Каждый флажок в наборе задается таким тегом, а пользователь щелчком может установить один или более из них. Принадлежность флажков одному и тому же

набору задается с помощью одинаковых имен — значений атрибута `name`. В этом случае удобно задавать значение атрибута `name` как массив, поскольку в PHP-сценарии легко организовать разбор того, что выбрал пользователь в наборе одноименных элементов. В *разд. 3.16* такой вариант уже рассматривался (см. листинг 3.72). В примере, приведенном в листинге 5.3, показано, как организовать получение массива значений, содержащихся в элементах типа "флажок". Вид формы в окне браузера и отклик сценария показаны на рис. 5.2.

Листинг 5.3. Передача значений флажков как массива

```
<FORM action="http://localhost/check.php" method=POST>
  <h2>Выберите товар:</h2>
  <INPUT type="checkbox" name="check[]" value="Автомобиль">Автомобиль<br>
  <INPUT type="checkbox" name="check[]" value="Телевизор">Телевизор<br>
  <INPUT type="checkbox" name="check[]" value="Пирожок">Пирожок<br>
  <INPUT type="submit" name="submit" value="Отправить">
</FORM>
<?php
  if (isset($_POST['check'])){      // если $_POST['check'] существует
    echo "<b>Вы выбрали: </b><br>";
    /* Вывод принятых значений флажков */
    foreach($_POST['check'] as $key=>$value){
      echo "$value <br>";
    }
  }
?>
```

Обратите внимание, что все флажки имеют одинаковые имена `"check[]"`, определенные как массив. Значение `$_POST['check']` является массивом, содержащим значения атрибутов `value` только тех флажков, которые были установлены, а доступ к ним можно получить с помощью числовых индексов. В данном примере `$_POST['check'][0]` содержит `"Автомобиль"`, а `$_POST['check'][1]` — `"Пирожок"`. Таким образом, длина массива `$_POST['check']` заранее не фиксирована и зависит от того, сколько флажков было установлено пользователем. Кроме того, нефиксированным оказывается и индекс в данном массиве одного и того же элемента формы. Это является следствием того, что в именах элементов формы (`check[]`) в квадратных скобках индексы не указывались.

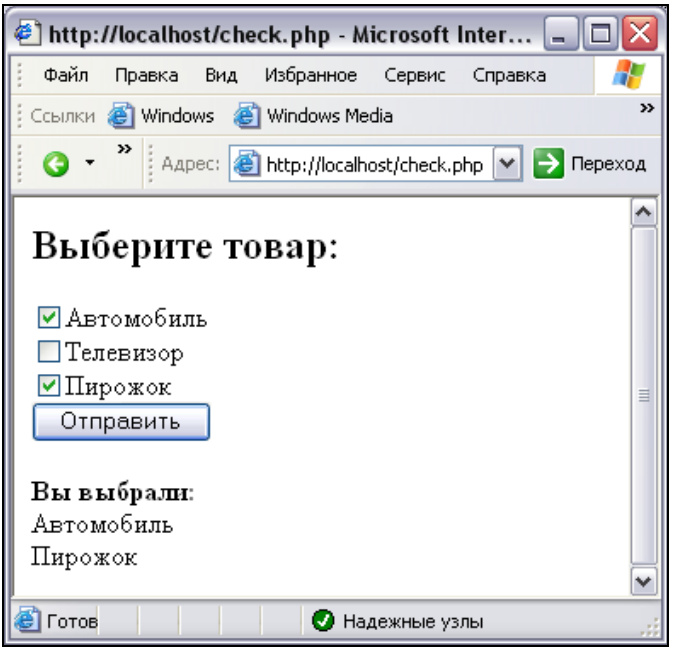


Рис. 5.2. Результат работы кода из листинга 5.3

При первом вызове данного файла массив `$_POST['check']` не существует, поэтому сценарий ничего не выводит. Если бы проверка на его существование с помощью функции `isset()` не производилась, то при первом вызове файла появилось бы сообщение об ошибке.

Внимание

Аналогичным образом можно организовать отправку на сервер групп значений радиокнопок и раскрывающихся списков.

Особенностью всех ранее рассмотренных в данном разделе примеров является то, что после отправки значений элементов формы на сервер последние принимают пустые значения. Чтобы сохранить в элементах те значения, которые были отправлены последний раз, сценарий придется несколько усложнить (листинг 5.4).

Листинг 5.4. Передача значений флажков с сохранением их в форме

```
<?php
  /* Какие флажки были установлены?
   * Массив $is_checked для каждого флажка содержит
```

```
   * либо " CHECKED" либо ""
   */
  if (isset($_POST['check'])){
    for ($i=0; $i<3;$i++){
      if (isset($_POST['check'][$i])){
        $ischecked[$i]=" CHECKED";
      }else{
        $ischecked[$i]="";
      }
    }
  }
  /* Вывод формы */
  echo '<FORM action="http://localhost/check.php" method=POST>
  <h2>Выберите товар:</h2>
  <INPUT type="checkbox" name="check[0]"
         value="Автомобиль"'.$ischecked[0].'>Автомобиль<br>
  <INPUT type="checkbox" name="check[1]"
         value="Телевизор"'. $ischecked[1].'>Телевизор<br>
  <INPUT type="checkbox" name="check[2]"
         value="Пирожок"'. $ischecked[2].'>Пирожок<br>
  <INPUT type="submit" name="submit" value="Отправить">
  </FORM>';
  if (isset($_POST['check'])){  // если $_POST['check'] существует
    echo "<b>Вы выбрали: </b><br>";
    /* Вывод принятых значений флажков */
    foreach($_POST['check'] as $key=>$value){
      echo "$value <br>";
    }
  }
?>
```

Как известно, установить флажок программным способом можно, если в теге этого элемента указать атрибут `CHECKED` (он не имеет значений). Таким образом, нам необходимо выводить форму, теги элементов которой имеют или не имеют атрибуты `CHECKED` в зависимости от того, были они установлены или нет при последней передаче на сервер. Чтобы было удобно работать с массивом `$_POST['check']`, имена флажков определены как массив с числовыми индексами (`check[0]`, `check[1]` и `check[2]`). В результате каждому переданному значению флажка в массиве `$_POST['check']` будет соответствовать элемент с фиксированным индексом. Например, первому флажку (**Автомобиль**) всегда будет соответствовать элемент `$_POST['check'][0]`. Разумеет-

ся, этот элемент будет находиться в массиве `$_POST['check']`, только если соответствующее ему значение формы было передано на сервер.

Собственно форма формируется как строка тегов и выводится с помощью оператора `$echo`. В эту строку в соответствующих местах вставляются элементы массива `$ischecked`, имеющие значения `"CHECKED"` либо `""` в зависимости от того, был ли установлен соответствующий флажок при последней передаче.

Теперь рассмотрим другую довольно типичную задачу передачи данных из формы. Пусть требуется сначала предоставить пользователю форму для выбора элементов (например, флажков) из заданного списка, а после отправки данных эту форму уже не предоставлять, а вывести только сообщение о полученных данных. Код для решения этой задачи представлен в листинге 5.5.

Листинг 5.5. Форма не выводится после передачи данных

```
<?php
  if (!isset($_POST['submit'])){   // если данные не передавались
    echo '<FORM action="http://localhost/check.php" method=POST>
          <h2>Выберите товар:</h2>
          <INPUT type="checkbox" name="check[]"
                 value="Автомобиль">Автомобиль<br>
          <INPUT type="checkbox" name="check[]"
                 value="Телевизор">Телевизор<br>
          <INPUT type="checkbox" name="check[]"
                 value="Пирожок">Пирожок<br>
          <INPUT type="submit" name="submit" value="Отправить">
          </FORM>';
  }else{    // если была передача
    if (isset($_POST['check'])){    // если что-то выбрано
      echo "<b>Благодарю за Ваш прекрасный выбор: </b><br>";
      foreach($_POST['check'] as $key=>$value){
        echo "$value <br>";
      }
    } else {
      echo "К сожалению, Вы ничего не выбрали";
    }
  }
?>
```

Немного изменим содержание только что рассмотренной задачи. А именно пусть в форме будут две кнопки — **Отправить** и **Отмена**. Форма должна отображаться в браузере только в следующих случаях:

- при первоначальном вызове файла check.php со сценарием;
- при отправке (был щелчок на кнопке **Отправить**) пустых данных (не установлен ни один из флажков).

Если были отправлены непустые данные (при отправке был установлен хотя бы один из флажков) или же был щелчок на кнопке **Отмена**, то форма не выводится. Обе кнопки, **Отправить** и **Отмена**, — элементы типа `submit`.

Для решения этой задачи нам потребуется дважды в различных местах использовать фрагмент кода, который выводит форму. Поэтому целесообразно сохранить его в отдельном файле, например, myform.inc (листинг 5.6). Затем в основной сценарий содержимое этого файла можно вставить с помощью функции `include()`.

Листинг 5.6. Файл myform.inc, включаемый в сценарий из файла check.php

```
<?php
  /* Форма, включаемая в сценарий check.php */
  echo '<FORM action="http://localhost/check.php" method=POST>
        <h2>Выберите товар:</h2>
        <INPUT type="checkbox" name="check[]"
               value="Автомобиль">Автомобиль<br>
        <INPUT type="checkbox" name="check[]"
               value="Телевизор">Телевизор<br>
        <INPUT type="checkbox" name="check[]"
               value="Пирожок">Пирожок<br>
        <INPUT type="submit" name="send" value="Отправить">
        <INPUT type="submit" name="cancel" value="Отмена">
        </FORM><br>';
?>
```

Основной сценарий представлен в листинге 5.7.

Листинг 5.7. Основной сценарий, содержащий включения кода из файла myform.inc

```
<?php
  if (isset($_POST['cancel'])) exit("<h1>До свидания!</h1>"); /* выход,
                                    если был щелчок на кнопке "Отмена" */
```

```
  if (!isset($_POST['send'])){  // если был щелчок на кнопке "Отправить"
    include("myform.inc");      // вывод формы
  }else{
    if (isset($_POST['check'])){    // если что-то выбрано
      echo "<b>Благодарю за Ваш прекрасный выбор: </b><br>";
      foreach($_POST['check'] as $key=>$value){
        echo "$value <br>";
      }
    } else {
      include("myform.inc");  // вывод формы
      echo "К сожалению, Вы ничего не выбрали";
    }
  }
?>
```

Заметим, что при наличии в форме нескольких кнопок типа `submit` на сервер передается значение только одной из них, а именно той, на которой был щелчок.

Совет

При рассмотрении сценариев, приведенных в этой книге, а также при создании и отладке собственных программ используйте функцию `var_dump(`*переменная*`)`. Например, с ее помощью полезно выводить на экран значения предопределенных переменных, таких как `$_POST`, `$_FILE` и др.

5.1.2. Загрузка файлов на сервер

Передачу на сервер файла, находящегося на стороне клиента, можно осуществить посредством формы. Для этого необходимо выполнить следующее:

- в теге `<FORM>` указать атрибуты:
  - `action=`*файл_со_сценарием* — php-файл, содержащий сценарий, обрабатывающий загруженный файл;
  - `enctype=multipart/form-data`;
  - `method=POST`;
- добавить в форму элементы:
  - `<INPUT type=file>` — поле ввода имени передаваемого файла с кнопкой **Обзор**, щелчок на которой открывает диалоговое окно для выбора файла из числа имеющихся;
  - `<INPUT type=hidden name=MAX_FILE_SIZE value=`*число*`>` — скрытое поле, содержащее *число*, которое указывает максимальный размер файла в байтах. Файл большего размера передать нельзя. Однако это

число не может превышать величины, указанные в конфигурационном файле php.ini:

◊ upload_max_filesize — максимальный размер загружаемого файла, по умолчанию — 2M. Это значение при необходимости можно изменить;

◊ post_max_size — максимальный объем данных, который можно передать методом POST, по умолчанию — 8M. Это значение при необходимости можно изменить.

Этот скрытый элемент не обязателен, но если он используется, то должен предшествовать элементу <INPUT type=file>;

❒ <INPUT type=submit> — кнопка.

Полученный сервером файл сначала сохраняется в папке для временных файлов. Эту папку с указанием пути к ней можно установить в конфигурационном файле php.ini как значение переменной upload_tmp_dir. Разумеется, она должна быть предварительно создана. Если же в переменной upload_tmp_dir ничего не указано, то используется временная папка, установленная по умолчанию, например, C:\Windows\Temp. Принятый файл далее необходимо скопировать или переместить в другую папку, поскольку содержимое временной папки не сохраняется после завершения работы сценария.

Кроме самого файла, сервер получает и всю информацию о нем, которая сохраняется в суперглобальном массиве $_FILES с символьными индексами. Если файл на компьютере клиента был выбран с помощью элемента <INPUT type=file name="userfile">, то для получения в серверном сценарии сведений об этом файле используются следующие элементы массива $_FILES['userfile']:

❒ $_FILES['userfile']['name'] — оригинальное имя файла (без пути) на компьютере клиента;

❒ $_FILES['userfile']['type'] — MIME-тип файла, если браузер предоставил такую информацию (например, "text/plain", "image/gif" и др.);

❒ $_FILES['userfile']['size'] — размер файла в байтах;

❒ $_FILES['userfile']['tmp_name'] — полное имя, под которым принятый файл был сохранен в папке для временных файлов;

❒ $_FILES['userfile']['error'] — код ошибки, которая может возникнуть при загрузке файла:

- 0 — ошибок нет (загрузка завершилась успешно);

- 1 — размер принятого файла превысил максимально допустимое значение, заданное директивой `upload_max_filesize` конфигурационного файла php.ini;
- 2 — размер загружаемого файла превышает значение `MAX_FILE_SIZE`, указанное в HTML-форме;
- 3 — загружаемый файл был принят только частично;
- 4 — файл не был загружен.

Еще раз отметим, что здесь `'userfile'` — имя элемента формы, т. е. значение его атрибута `name`, которое выбирается разработчиком по своему усмотрению.

В листинге 5.8 приведен код, предоставляющий пользователю форму для выбора передаваемого файла (рис. 5.3) и содержащий сценарий, который перемещает временный файл в папку DOWNLOADED, предварительно созданную в той же папке, в которой находится файл со сценарием.

Листинг 5.8. Загрузка файла на сервер

```
<FORM action="http://localhost/fileup.php" method=POST
      enctype=multipart/form-data>
  <INPUT type=hidden name="MAX_FILE_SIZE" value="100000">
  <INPUT type="file" name="userfile" value="">
  <INPUT type="submit" name="submit" value="Отправить">
  </FORM>
<?php
  $tempfile=$_FILES['userfile']['tmp_name']; // имя временного файла
  if (is_uploaded_file($tempfile)){  // если файл загружен
    if (!copy($tempfile, "./DOWNLOADED/".$_FILES['userfile']['name']) ){
      echo "Не удалось скопировать файл ";
    } else {
      echo "Файл {$_FILES['userfile']['name']}
          ({$_FILES['userfile']['size']}байт) загружен успешно";
    }
  } else {
  echo "Не удалось загрузить файл {$_FILES['userfile']['name']}";
  }
?>
```

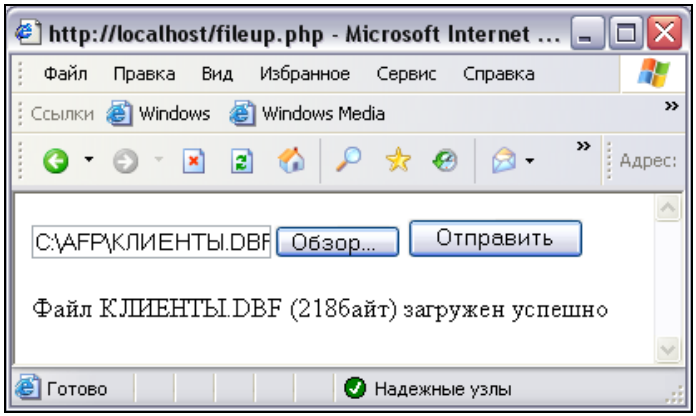


Рис. 5.3. Форма для передачи файла на сервер

Для проверки успешности загрузки файла во временную папку используется встроенная функция `is_uploaded_file(`*`имя_файла`*`)`. Для перемещения файла в требуемую папку применяется функция `move_uploaded_file(`*`исходный_файл`*`, `*`целевой_файл`*`)`. Чтобы просто скопировать файл, можно использовать функцию `copy(`*`исходный_файл`*`, `*`целевой_файл`*`)`. Перечисленные функции принимают строковые параметры и возвращают `true` или `false` в зависимости от успешности операции.

5.2. Переходы и передача данных между Web-страницами

Переходы между Web-страницами чаще всего реализуются посредством тех элементов, в которых можно указать URL-адрес, — ссылок и форм. Вместе с тем иногда требуется осуществить переход на другую страницу сайта без вмешательства пользователя, т. е. программным способом. Кроме того, при переходах нередко возникает задача передачи данных от одной страницы к другой. Причем передаваемая информация может быть не только введена пользователем в элементы формы, но и сгенерирована программно.

В данном разделе мы рассмотрим способы перехода между страницами, а также средства обеспечения доступности данных, сгенерированных где-то на одной или нескольких страницах сайта, другим страницам (точнее, PHP-сценариям).

Итак, для переходов между страницами в PHP можно использовать следующие средства:

- ссылки;

- формы;
- встроенную PHP-функцию `header()`.

Для передачи данных между страницами можно использовать такие приемы:

- добавление информации к URL-адресу;
- применение форм *(см. разд. 5.1)*;
- применение cookie;
- применение сеансов PHP.

Далее мы рассмотрим перечисленные выше средства более подробно.

5.2.1. Вывод ссылок

Чтобы отобразить на Web-странице ссылку, в PHP-файле достаточно записать где-нибудь вне дескрипторов `<?` и `?>` соответствующий HTML-код или же внутри этих дескрипторов воспользоваться оператором `echo`:

```
<a href = 'page1.php'>Страница 1</a>
<?php echo "<a href = 'page2.php'>Страница 2</a>" ;?>
```

Напомним, что значением атрибута `href` является, вообще говоря, URL-адрес и, следовательно, к нему можно дописать информацию, которая будет передана указанному сценарию *(см. разд. 5.2.3)*.

5.2.2. Использование форм

Возможность применения HTML-форм для перехода на другую страницу обусловлена наличием в теге `<FORM>` атрибутов `action` с URL-адресом, `method` с указанием метода передачи и кнопки типа `submit`. Передача данных из элементов формы рассматривалась в *разд. 5.1*. Однако форма может и не содержать никаких элементов для ввода и/или хранения данных. В этом случае щелчок на кнопке типа `submit` просто приведет к загрузке в браузер страницы с указанным адресом. Таким способом реализуются, например, кнопки для навигации **Далее**, **Назад** и т. п. Если к URL-адресу добавить информацию, то она будет передана на сервер *(см. разд. 5.2.4)*.

5.2.3. Применение функции *header()*

Чтобы перейти на заданную страницу без каких-либо действий пользователя браузера, можно применить встроенную функцию `header("Location: URL-адрес")`. Предположим, на странице page0.php выводится форма для ввода данных о пользователе (`login`, `password` и т. п.). Эти данные отправляются

серверному сценарию, например, redirect.php, который как-то обрабатывает полученные данные и в зависимости от результата автоматически перенаправляет пользователя к странице page2.php или page3.php.

Примечание

Все файлы страниц, за исключением файла с серверным сценарием (в нашем случае это redirect.php), не обязательно должны быть php-файлами: они могут содержать только обычные HTML-коды и иметь расширения htm или html.

Допустим, что пользователь со страницы page0.php передал на сервер значение поля с именем `login`. Тогда содержимое файла redirect.php может иметь, например, вид, как в листинге 5.9.

Листинг 5.9. Переадресация в зависимости от полученных данных

```
<?php
  if (strtolower($_REQUEST['login']) == "admin"){
    header("Location: http://www.myserver.ru/page1.php");
  }else{
    header("Location: http://www.myserver.ru/page2.php");
  }
?>
```

Здесь использовался суперглобальный массив `$_REQUEST`, чтобы не привязываться к конкретному методу передачи данных. Функция `strtolower()` применена для того, чтобы результат проверки не зависел от регистра, в который были введены данные.

Внимание

Функция `header()` имеет особенность: ее применение должно предшествовать всему, что отправляется браузеру клиента.

Функцию `header()` нельзя использовать после текста, HTML-кода и операторов `echo`. Другими словами, если что-то уже было выведено в браузер, то применение функции `header()` приведет к предупреждающему сообщению о невозможности изменить заголовок, поскольку данные уже отправлены. Вот пример неправильного использования функции `header()`:

```
<?php
  echo "Ваше регистрационное имя - {$_REQUEST['login']}";
  if (strtolower($_REQUEST['login']) == "admin"){
```

```
    header("Location: http://www.myserver.ru/page1.php");
  }else{
    header("Location: http://www.myserver.ru/page2.php");
  }
?>
```

Примечание

Вообще говоря, функция `header()` предназначена для дополнения или изменения HTTP-заголовков, передаваемых браузеру вместе с данными. Еще один пример использования этой функции можно найти в *разд. 5.4.3*.

В листинге 5.10 приведен в качестве примера сценарий, который перенаправляет пользователя на одну страницу, если браузер клиента — Microsoft Internet Explorer, и на другую страницу — в противном случае. Сведения о браузере клиента, который обратился к данному файлу со сценарием, можно взять из `$_SERVER['HTTP_USER_AGENT']` как строку, имеющую, например, такой вид:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; MRA 4.3 (build 01218))
```

Листинг 5.10. Переадресация в зависимости от браузера клиента

```
<?php
  if (strpos($_SERVER['HTTP_USER_AGENT'],"MSIE")>0){
    header("Location: http://www.myserver.ru/page1.php");
  }else{
    header("Location: http://www.myserver.ru/page2.php");
  }
?>
```

5.2.4. Добавление информации к URL-адресу

Информация, которую можно добавить к URL-адресу, состоит из элементов вида *имя_переменной=значение*. Перед именем переменной символ `$` не используется. Если таких элементов несколько, то между ними указывается символ `&` в качестве разделителя. Вся последовательность элементов дописывается к URL-адресу через знак вопроса (`?`), например,

```
<a href=page1.php?name=Вадим&email=vadim@xyz.ru>Страница 1</a>
```

Информация, добавляемая к URL-адресу, передается методом `GET` и в серверном сценарии может быть получена посредством суперглобальных массивов `$_GET` и `$_REQUEST`.

Пусть, например, текущая страница сгенерирована следующим кодом:

```
<?php
  echo "<a href = page1.php?ip={$_SERVER['REMOTE_ADDR']}&name=
  Вадим&email=vadim@xyz.ru>
Страница 1</a>";
?>
```

Если в файле page1.php записать код `<?php var_dump($_GET) ?>`, то в окне браузера отобразится примерно следующее:

```
array(3) {
  ["ip"]=> string(9) "127.0.0.1"
  ["name"]=> string(5) "Вадим"
  ["email"]=> string(12) vadim@xyz.ru
}
```

Здесь `$_SERVER['REMOTE_ADDR']` — IP-адрес компьютера, на котором просматривается страница, сгенерированная данным сценарием.

Не следует забывать, что добавляемая к URL-адресу информация отображается в адресной строке браузера и может быть в нем сохранена (в папке Избранное, например). Поэтому не стоит таким способом передавать пароли и другие конфиденциальные сведения. Кроме того, объем передаваемых методом `GET` данных ограничен.

5.2.5. Применение cookie

Что такое cookie, рассматривалось в *разд. 3.8*. Этот механизм первоначально задумывался как основной для обеспечения обмена данными между различными страницами. Однако следует помнить, что данные cookie сохраняются на компьютере клиента. Пользователь браузера может управлять настройками механизма cookie вплоть до полного его отключения (многие так и поступают). Это обстоятельство ограничивает использование cookie в разработках серьезных Web-приложений. Так, если приложение ориентировано на использование cookie, а пользователь отключил их, то приложение может работать некорректно. Например, в некоторых гостевых книгах добавление записей пользователей возможно, только если механизм cookie включен. В частности, так работает моя гостевая книга.

В PHP установка значений cookie-записей производится с помощью встроенной функции `setcookie()`, а их чтение — с помощью суперглобального массива `$_COOKIE`.

Функция `setcookie(name, value, expires, path, domain, secur)` принимает следующие параметры:

- *name* — строка, содержащая имя cookie-записи (обязательный параметр);
- *value* — строка, содержащая значение cookie;
- *expires* — срок хранения cookie-записи как количество секунд; если отсутствует, то после завершения работы браузера cookie-запись удаляется; время жизни следует задавать с помощью функции `time()` или `mktime()` *(см. разд. 4.9)*. Например, если необходимо установить время жизни cookie-записи 2 часа, начиная от момента ее создания, то значение данного параметра следует задать как `time()+2*60*60`;
- *path* — строка, содержащая путь cookie-записи;
- *domain* — строка, содержащая домен нужной cookie-записи;
- *secure* — логическое значение (`true` или `false`), определяющее необходимость использования безопасного HTTP-соединения.

Для применения функции `setcookie()` обязательным является только первый параметр — имя cookie-записи. Для создания cookie-записи достаточно указать непустое значение второго параметра.

Для удаления cookie-записи достаточно установить для нее пустое значение:

```
setcookie("mycookie");
setcookie("mycookie", "");
```

Внимание

Функция `setcookie()` имеет такую же особенность, что и `header()`: ее применение должно предшествовать всему, что отправляется браузеру. Так, ее нельзя использовать после текста, HTML-кода и операторов `echo`.

Данные cookie недоступны из PHP-сценария на той же странице, где они были созданы. Чтобы получить к ним доступ с помощью суперглобального массива `$_COOKIE`, следует перейти на другую страницу либо перезагрузить текущую страницу. Это не обременительно, поскольку cookie для того и существуют, чтобы сохранять данные, доступные для разных страниц. Тем не менее доступ к cookie все-таки возможен и на той же странице, на которой они были созданы, но из клиентского сценария. Функции на JavaScript для установки и чтения записей на стороне клиента приведены в *разд. 3.8.2*.

Для проверки возможности применения механизма cookie убедитесь, что в вашем браузере он не отключен, и создайте два файла — setcookie.php и readcookie.php, как в листингах 5.11 и 5.12 соответственно. Затем загрузите в браузер файл setcookie.php, который установит cookie-запись и выполнит переход к файлу readcookie.php. Последний прочитает cookie-запись и отобразит ее значение.

Листинг 5.11. Установка cookie-записи (файл setcookie.php)

```
<?php
  setcookie("test","Привет!"); // установка cookie-записи с именем test
  header("Location: readcookie.php");  // переход
?>
```

Листинг 5.12. Чтение cookie-записи (файл readcookie.php)

```
<?php
  echo "Значение cookie-записи с именем test: {$_COOKIE['test']}";
?>
```

5.2.6. Применение сеансов PHP

Сеанс (сессия, session) в PHP — механизм, который позволяет обеспечить доступность данных, созданных на различных серверных страницах в рамках одного сайта, для других страниц в этом же сайте. Технология сеансов перекрывает возможности и преодолевает ограничения cookie по обмену данными между Web-страницами. Сеанс создается и закрывается. В интервале времени между этими событиями все PHP-сценарии, расположенные в различных файлах и подключенные к данному сеансу, имеют доступ к переменным, созданным в других сценариях этого же сайта.

Создание сеанса

Сеанс создается с помощью встроенной функции `session_star()` на каждой серверной странице, которую требуется подключить к сеансу. Если необходимо, чтобы все страницы имели доступ ко всем данным, сгенерированным на любой из них, то функцию `session_star()` следует вызвать на всех страницах.

Функция `session_star()` не имеет параметров, а возвращает всегда `true`.

Внимание

Вызов функции `session_start()` должен предшествовать выводу какой-либо информации в браузер.

Рассмотрим механизм создания сеанса.

1. При вызове в сценарии функции `session_star()` интерпретатор PHP проверяет, был ли создан сеанс ранее. Если сеанс уже создан на какой-то странице сайта, то переменные сеанса (созданные на других страницах)

устанавливаются для данного сценария. Если сеанс еще не существует, то создается новый сеанс, для которого автоматически генерируется уникальный идентификатор (длинная строка символов). Идентификатор сеанса присваивается глобальной предопределенной переменной `$PHPSESSID`.

2. В пределах существующего сеанса (после вызова функции `session_star()`) можно установить сеансовые переменные. Для этого служит суперглобальный массив `$_SESSION`. Чтобы присвоить значение переменной, используется такой синтаксис:

   `$_SESSION['имя_переменной'] = значение;`

   Для получения значения уже созданной переменной используют обращение `$_SESSION['имя_переменной']`.

   Чтобы удалить сеансовую переменную, можно применить функцию

   `unset($_SESSION['имя_переменной']);`

3. Все переменные сеанса сохраняются на сервере в файле с именем, совпадающим с идентификатором сеанса. Можно самому определить папку для хранения файлов сеансов. Такая предварительно созданная папка задается директивой `session.save_path` в конфигурационном файле php.ini. Файл сеанса доступен для всех сценариев в пределах данного сеанса.

4. Интерпретатор PHP передает идентификатор сеанса каждой странице. Передача данных между сервером и клиентом происходит с использованием cookie или без него. Механизм cookie применяется, если пользователь браузера не отключил его. В противном случае работа определяется директивой `session.use_trans_sid` в конфигурационном файле php.ini, о чем будет рассказано позже.

Внимание

Можно заставить интерпретатор автоматически добавлять функцию `session_start()` в начале каждой серверной страницы. Для этого необходимо в конфигурационном файле php.ini записать директиву `session.auto_start = 1`.

Получить значение идентификатора текущего сеанса можно не только как значение переменной `$PHPSESSID`, но и с помощью функции `session_id()`. Эта функция возвращает строку, содержащую идентификатор сеанса или пустую строку, если сеанс не был создан. Кроме того, передав этой функции строку символов в качестве параметра, можно установить соответствующий идентификатор для текущего сеанса.

В ряде случаев требуется принудительно закрыть сеанс. Например, при организации посещения сайта через регистрационное имя (login) и пароль для

каждого посетителя создается отдельный сеанс. После проверки этих данных следует закрыть сеанс. Для этого служит встроенная функция `session_destroy()`, которая удаляет все переменные сеанса и прекращает передачу идентификатора сеанса. Однако в текущем сценарии эти переменные остаются доступными, и для их удаления следует воспользоваться функцией `unset()`.

Особенности механизма сеансов

Ранее уже отмечалось, что работа механизма сеансов определяется тем, включена ли поддержка cookie на стороне клиента, а также значением директивы `session.use_trans_sid` в файле php.ini. Рассмотрим эти особенности подробнее.

- **Поддержка cookie включена.** В этом случае глобальной переменной `$PHPSESSID` присваивается значение идентификатора сеанса, а cookie используется для передачи значения `$PHPSESSID` между сценариями (страницами).
- **Поддержка cookie отключена.** Интерпретатор PHP создает константу SID, значением которой является "`PHPSESSID=`*`идентификатор_сеанса`*". Использование идентификатора сеанса определяется значением директивы `session.use_trans_sid`. По умолчанию это значение равно 0 (директива отключена). Чтобы включить данную директиву, необходимо в файле php.ini изменить соответствующую строку: `session.use_trans_sid = 1`. При включенной директиве можно создавать сценарии, не зависящие от настройки браузера по применению cookie. Рассмотрим случаи различных значений директивы `session.use_trans_sid`:
  - `session.use_trans_sid = 1`. Идентификатор сеанса при отключенной поддержке cookie пересылается посредством элементов, в которых указан URL-адрес. При переходе на другую страницу с помощью ссылки, функции `header()` или формы с методом `GET` идентификатор сеанса добавляется к URL-адресу и виден в адресной строке браузера, что иногда нежелательно. Следует иметь в виду, что идентификатор сеанса добавляется к URL-адресам, соответствующим одному и тому же сайту. Если в URL-адресе содержится имя сервера, то к такому URL-адресу идентификатор сеанса не будет добавлен. Например, при использовании ссылки `<a href="nextpage.php">` идентификатор будет добавлен к адресу, а при использовании ссылки `<a href="http://myserver.ru/nextpage.php">` — нет.

    Однако можно использовать передачу идентификатора сеанса в скрытых полях формы с методом `POST`, при котором идентификатор сеанса не добавляется к URL-адресу:

    ```
    <input type=hidden name='PHPSESSID' value='$PHPSESSID'>
    ```

- `session.use_trans_sid = 0`. Идентификатор сеанса при отключенной поддержке cookie не пересылается. При необходимости разработчик приложения делает это самостоятельно, используя константу SID. Поскольку эта константа имеет значение вида "`PHPSESSID=`*`идентификатор_сеанса`*", то можно просто добавить ее к URL-адресу, например,

  ```
  <a  href="mainpage.php?<?php echo SID?>">Главная страница</a>
  ```

  Однако в этом случае идентификатор сеанса, видимый в адресной строке браузера, может быть использован злоумышленниками в неблаговидных целях.

Тестовый пример использования сеанса

Рассмотрим тестовый пример организации сеанса. Для его воспроизведения убедитесь, что в вашем браузере не отключена поддержка cookie, или установите директиву `session.use_trans_sid = 1` в файле php.ini. Чтобы изменения в php.ini вступили в силу, необходимо перезапустить Web-сервер.

В рассматриваемом далее примере имеются две серверные страницы, каждая из которых вносит собственный вклад в формирование множества сеансовых переменных. Первая страница сначала создает две переменные, затем вторая страница добавляет еще одну переменную, и, наконец, все три переменные становятся доступными для двух страниц.

В листинге 5.13 приведен код, формирующий первую страницу (файл s1.php). Здесь создается сеанс и две сеансовые переменные — `myname` и `myaddr`. Однако сценарий пытается вывести значение переменной `mylastname`, которая еще не была создана. Поэтому на странице в браузере (рис. 5.4, слева вверху) это значение не будет отображаться.

Листинг 5.13. Применение сеанса (первая страница)

```
<?php
  session_start(); // открытие сеанса
  /* Создание переменных сеанса */
  $_SESSION['myname']="Вадим";
  $_SESSION['myaddr']="Санкт-Петербург";
?>
<html>
  <h2>Тестирование сеансов</h2>
  <b>Страница 1</b><br>
```

```
  <a href=s2.php>Перейти на страницу 2</a>
</html>
<?php
  /* Вывод переменной сеанса */
  echo "<br>";
  echo "Фамилия: {$_SESSION['mylastname']}";
?>
```

В результате щелчка на ссылке будет загружена вторая страница (файл s2.php), сформированная кодом листинга 5.14. Здесь создается только одна переменная `mylastname`, которой еще не было при загрузке первой страницы, и выводятся значения всех трех переменных текущего сеанса (рис. 5.4, справа вверху). Таким образом, одна переменная была создана на второй странице, а две другие — переданы ей из первой. Если на данной странице щелкнуть по ссылке, произойдет возвращение к первой странице, на которой теперь будут доступны все переменные сеанса. Поэтому значение переменной `mylastname` отобразится (рис. 5.4, внизу).

Листинг 5.14. Применение сеанса (вторая страница)

```
<?php
  session_start();// подключение к сеансу
  /* Создание переменной сеанса */
  $_SESSION['mylastname']="Дунаев";
?>
<html>
  <h2>Тестирование сеансов</h2>
  <b>Страница 2</b><br>
  <a href=s1.php>Перейти на страницу 1</a>
</html>
<?php
  /* Вывод переменных сеанса */
  echo "<br>";
  echo "Имя: {$_SESSION['myname']}<br>";
  echo "Фамилия: {$_SESSION['mylastname']}<br>";
  echo "Адрес: {$_SESSION['myaddr']}";
?>
```

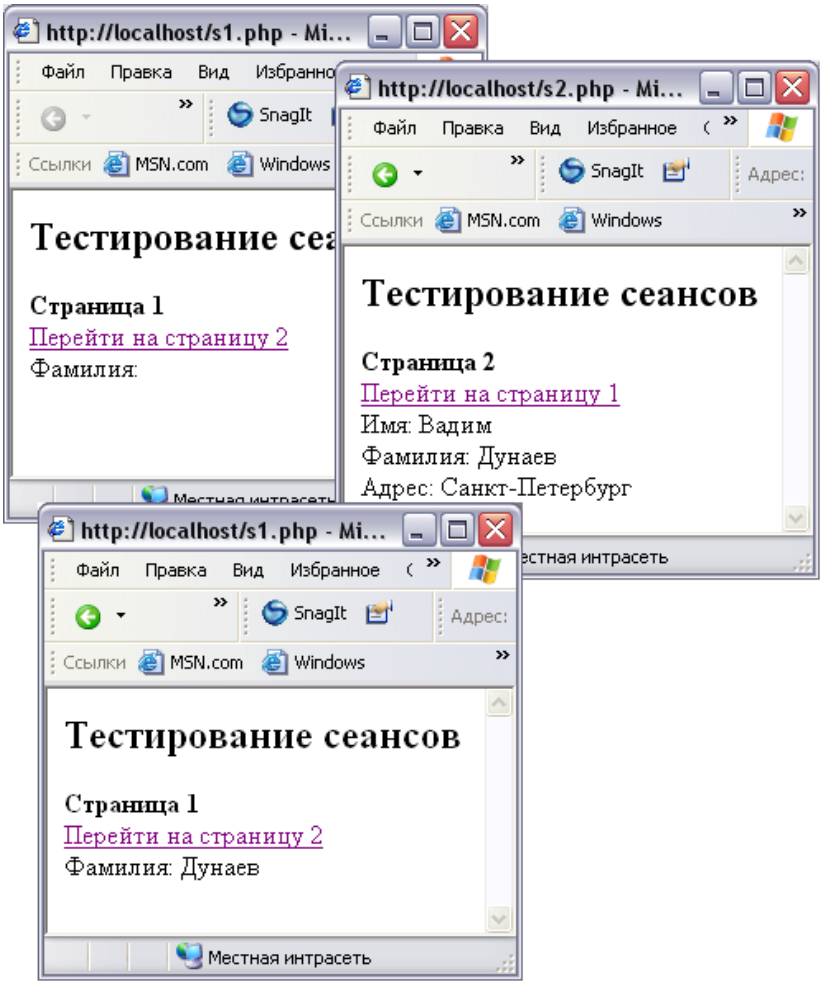


Рис. 5.4. Пример использования сеанса

Защита страниц паролем

Теперь рассмотрим в качестве примера задачу защиты страниц сайта паролем и/или требованием регистрации. С помощью сеанса можно потребовать от пользователя зарегистрироваться или ввести пароль только один раз, а не делать это при посещении каждой страницы сайта. В общих чертах алгоритм действий может быть таким:

1. Отобразить в браузере страницу, содержащую форму для ввода регистрационного имени и/или пароля.

2. Если регистрация по вашему критерию прошла успешно, создать сеанс с соответствующими переменными.
3. При переходе пользователя на другую страницу проверить успешность регистрации на основе соответствующих сеансовых переменных. Если пользователь успешно зарегистрировался, то загрузить ему требуемую страницу, а в противном случае — предложить ему зарегистрироваться, т. е. загрузить ему страницу с элементами ввода регистрационных данных (см. п. 1).

Возможны несколько вариантов реализации данного алгоритма. Мы рассмотрим два из них.

Вариант 1

Серверная страница pwinput.php, на которой пользователь вводит пароль, может иметь примерно такой код, который представлен в листинге 5.15.

Листинг 5.15. Код для ввода пароля (pwinput.php) *(вариант 1)*

```
<?php
  /* Страница для ввода пароля */
  session_start();// открытие сеанса
  echo "<form action='nextpage.php' method=POST>Введите пароль:
          <input type=password name='userpw'>
          <input type=submit value='OK'>
        </form>";
?>
```

Здесь пароль, введенный пользователем в поле, маскируемое символами `*`, передается на страницу nextpage.php методом `POST`. Код этой страницы может иметь вид, представленный в листинге 5.16.

Листинг 5.16. Код с проверкой правильности пароля (nextpage.php) *(вариант 1)*

```
<?php
  /* Проверка правильности пароля */
  session_start();                        // подключение к текущему сеансу
  if (!isset($_SESSION['userpw'])) {
    $_SESSION['userpw']=$_POST['userpw'];
  }
  $truepw="abracadabra";                  // верный пароль
```

```
  if ($_SESSION['userpw'] !=$truepw) {  // если пароль не верен
    session_destroy();                  // закрываем сеанс
    header("Location: pwinput.php");    // вернуться к вводу пароля
  } else {                              // если пароль верен
    $_SESSION['userpw']=$truepw;
  }
?>
<! Здесь может быть какой-нибудь HTML-документ или PHP-код >
```

Сценарий в листинге 5.16 проверяет правильность пароля, и если он не верен, производит переход к странице для ввода пароля. Рассмотрим этот сценарий подробнее.

Поскольку пароль вводится пользователем в поле формы и передается методом `POST`, при первой его проверке сеансовая переменная `$_SESSION['userpw']` еще не установлена. В этом случае мы создаем ее, присваивая полученное от пользователя значение `$_POST['userpw']`.

Далее происходит сравнение полученного значения пароля с правильным. Если они не совпадают, то мы закрываем сеанс и возвращаемся на страницу ввода пароля. Заметьте, что если не закрыть сеанс, то при следующем обращении к странице проверки пароля новое, полученное от пользователя, значение пароля не присвоится сеансовой переменной `$_SESSION['userpw']`, поскольку она уже установлена. В результате даже при правильно введенном пароле последующий код сценария всегда переадресовывал бы пользователя на страницу ввода пароля.

Если полученный от пользователя пароль верен, то его значение присваивается переменной `$_SESSION['userpw']` и, далее, отображается содержимое данной страницы (в листинге 5.16 оно не указано).

Сценарий, приведенный в листинге 5.16, следует вставить в каждую серверную страницу, которую вы хотите защитить паролем. Можно сохранить сценарий в отдельном файле, а вставлять его содержимое в защищаемые страницы с помощью функции `include()`. В любом случае для перемещения по страницам сайта пользователю придется ввести верный пароль только один раз.

Вариант 2

На серверной странице pwinput.php (листинг 5.17) форма для ввода пароля отображается, только если верный пароль еще не был введен пользователем. Кроме того, на этой же странице выводится ссылка на страницу nexpage.php. Щелчок на кнопке **OK** типа `submit` отправляет данные формы не на nexpage.php, как в первом варианте, а на ту же самую страницу pwinput.php.

Если отправленный пароль верен, то форма на этой странице не отображается. Вместо нее выводится сообщение об успешной регистрации. Обратите внимание, что на странице pwinput.php сценарий сначала проверяет значение сеансовой переменной `$_SESSION['userpw']`. Поскольку она может еще не существовать или содержать неверный пароль, то в этих случаях проверяется значение поля `userpw`, полученное при передаче из формы. Если значение `$_POST['userpw']` совпадает с верным паролем, то оно присваивается сеансовой переменной, которая далее используется в сценарии проверки правильности пароля на всех защищаемых страницах и, в частности, на странице nexpage.php (листинг 5.18).

Листинг 5.17. Код для ввода пароля (pwinput.php) ***(вариант 2)***

```
<?php
  /* Страница для ввода пароля */
  session_start();           // открытие сеанса
  $truepw="abracadabra";     // верный пароль
  if ($_SESSION['userpw']!=$truepw) {
    if ($_POST['userpw']!=$truepw){
      echo "<form action= pwinput.php' method=POST>
              Введите пароль:
              <input type=password name='userpw'>
              <input type=submit value='OK'>
            </form>";
    } else {
      $_SESSION['userpw']=$truepw;
      echo "Регистрация прошла успешно";
    }
  }
  echo "<br><a href=s8.php>Следующая страница</a>";
?>
```

Листинг 5.18. Код с проверкой правильности пароля (nextpage.php) ***(вариант 2)***

```
<?php
  /* Проверка правильности пароля */
  session_start();                      // подключение к текущему сеансу
  $truepw="abracadabra";                // верный пароль
  if ($_SESSION['userpw'] !=$truepw) {  // если пароль не верен
```

```
    header("Location: pwinput.php");    // вернуться к вводу пароля
  }
?>
<! Здесь может быть какой-нибудь HTML-документ или PHP-код >
```

Сценарий, приведенный в листинге 5.18, можно сохранить в отдельном файле и с помощью функции `include()` вставить в каждую серверную страницу, которую вы хотите защитить паролем.

5.3. Работа с текстовыми файлами

Для длительного хранения данных в файловой системе сервера часто используются обычные текстовые файлы. Вот типичная последовательность действий при работе с файлом:

1. Открыть файл.
2. Прочитать данные из файла или записать данные в файл.
3. Закрыть файл.

С открытым файлом связано понятие *указателя*. При выполнении различных файловых операций указатель может перемещаться, отмечая место, с которого можно получить доступ к содержимому файла. Например, если указатель находится в начале файла, то операция записи новых данных в этот файл будет уничтожать содержащиеся в нем старые данные. Если указатель находится в конце файла, то записываемые в него данные будут добавляться к уже имеющимся.

5.3.1. Открытие файла

Для открытия файла служит встроенная функция `fopen(`*имя_файла, режим*`)`, которая возвращает дескриптор (идентификатор) открытого файла или `false` при неудаче. Дескриптор файла используется в сценарии для обращения к открытому файлу как ресурсу. Параметры имеют строковый тип.

Первый параметр — имя или URL-адрес файла. Если путь к файлу не указан, то предполагается текущая папка, т. е. та, в которой расположен файл со сценарием. Все папки, указанные в пути, должны существовать, а вы должны иметь к ним и открываемому файлу доступ.

Внимание

В системе Windows при указании пути следует использовать либо одинарные прямые, либо двойные обратные слэши, например, C:/myfolder/myfile.txt, C:\\myfolder\\myfile.txt.

Второй параметр определяет режим доступа к открываемому файлу и может принимать следующие строковые значения:

- `r` — только чтение; указатель в начале файла; если файл не существует, то `fopen()` вернет `false` и выведет предупреждение. Таким образом, это режим чтения файла сначала;
- `r+` — чтение и запись; указатель в начале файла; если файл не существует, то `fopen()` вернет `false` и выведет предупреждение. Таким образом, это режим чтения файла сначала и перезаписи данных;
- `w` — только для записи; указатель в начале файла; если файл существует, то он обрезается до нулевой длины, в противном случае — делается попытка его создать. Таким образом, это режим только перезаписи данных;
- `w+` — чтение и запись; указатель в начале файла; если файл существует, то он обрезается до нулевой длины, в противном случае — делается попытка его создать. Таким образом, это режим чтения и перезаписи данных;
- `a` — только запись; указатель в конце файла; если файл не существует, то делается попытка его создать. Таким образом, это режим только добавления данных;
- `a+` — чтение и запись; указатель в конце файла; если файл не существует, то делается попытка его создать. Таким образом, это режим чтения и добавления данных;
- `x` — файл создается и открывается только для записи с указателем в начале файла. Если файл уже существует, то `fopen()` вернет `false` и выведет предупреждение;
- `x+` — файл создается и открывается для чтения и записи с указателем в начале файла. Если файл уже существует, то `fopen()` вернет `false` и выведет предупреждение.

Например, для открытия уже существующего или создания с последующим открытием файла mytest.txt в режиме чтения и перезаписи можно воспользоваться следующим выражением:

```
$fh=fopen("mytest.txt", "w+");
```

В данном случае предполагается, что файл расположен или будет создан в текущей папке. Для выполнения той же операции над файлом в папке C:\myfolder следует применить такое выражение:

```
$fh=fopen("C:/myfolder/mytest.txt", "w+");
```

Однако в этом случае предполагается, что указанная папка существует и вы имеете доступ к ней.

При открытии существующего файла в режиме `w+` его содержимое уничтожается.

Внимание

Вывод предупреждающих сообщений в случаях возвращения функцией `fopen()` значения `false` при неудачах открытия файла не прерывает выполнения последующих выражений сценария, что, однако, может привести к его некорректной работе.

Чтобы подавить вывод предупреждающего сообщения при неуспешном применении функции `fopen()`, можно использовать выражение `@fopen()`. Если же требуется предотвратить последующее выполнение сценария, то воспользуйтесь такой конструкцией:

```
$fh=fopen("mytest.txt", "w+r") or exit("Не удалось открыть файл");
```

При работе с файлами часто применяются функции проверки существования папки и файла:

- `file_exists(`*имя_файла*`)` — проверяет существование указанного файла и возвращает `true` или `false`;
- `is_dir(`*имя_файла*`)` — возвращает `true`, если указанное в параметре имя является именем существующей папки; в противном случае возвращает `false`.

5.3.2. Закрытие и удаление файлов

Если после использования открытый файл следует закрыть, для этого применяется функция `fclose(`*дескриптор*`)` с указанием в качестве параметра дескриптора ранее открытого файла. Данная функция возвращает `true` или `false` в зависимости от успешности операции закрытия файла. При неудаче выводится предупреждающее сообщение.

Пример:

```
$fh=fopen("mytest.txt", "w+r");
// здесь могут быть какие-то операции над открытым файлом
fclose($fh);
```

Для удаления файла с диска на сервере применяется функция `unlink(`*имя_файла*`)`, которая возвращает `true` или `false` в зависимости от успешности операции. При неудаче выводится предупреждающее сообщение. Удалять можно как открытый, так и закрытый файл.

5.3.3. Чтение файла

Содержимое файла можно записать в переменную или массив. Для этого используются специальные встроенные функции чтения файла:

- чтение файла в переменную: `fgets()`, `file_get_contents()`, `fread()`, `fgetss()`, `fgetcsv()`;
- чтение файла в массив: `file()`.

Чтение файла в переменную

Функция `fgets(`*дескриптор*`, `*длина*`)` позволяет читать содержимое файла побайтно до тех пор, пока не будет прочитано столько байтов, сколько указано в параметре *длина* минус 1, либо не будет достигнут символ конца строки или файла. Если параметр *длина* не указан, то чтение будет происходить до конца строки или файла. Данная функция возвращает прочитанную строку с добавлением символов конца строки и файла или `false`, если при чтении произошла ошибка. Напомним, что в браузере символы конца строки и файла отображаются в виде пробела.

Чтобы прочитать весь файл в переменную, достаточно выполнить код, представленный в листинге 5.19.

Листинг 5.19. Построчное чтение файла в переменную

```
<?php
  $filename="test.txt";
  $fh=fopen($filename,"r"); // открытие файла в режиме только чтения
  while (!feof($fh)) {      // пока не достигнут конец файла
    $buff .= fgets($fh);    // запись в переменную
  }
  fclose($fh);              // закрытие файла
  echo $buff;               // вывод значения переменной
?>
```

Здесь в операторе цикла используется встроенная функция `feof(`*дескриптор*`)`, которая возвращает `true` при достижении указателем конца файла, а в противном случае возвращается `false`.

Запись содержимого файла в переменную можно выполнить, просто вызвав встроенную функцию `file_get_contents(`*имя_файла*`)`. При этом не требуется предварительно открывать файл. Данная функция имеет еще несколько необязательных параметров (см. справочное руководство по PHP).

Записать в переменную заданное количество байтов из открытого файла можно с помощью функции `fread(дескриптор, длина)`.

Чтение файла в массив

Нередко прочитанные строки удобно разместить в массиве. С этой целью можно воспользоваться кодом, представленным в листинге 5.20.

Листинг 5.20. Построчное чтение файла в массив

```
<?php
  $filename="test.txt";
  $fh=fopen($filename,"r"); // открытие файла в режиме только чтения
  while (!feof($fh)) {      // пока не достигнут конец файла
    $abuff[]=fgets($fh);    // запись в массив
  }
  fclose($fh);              // закрытие файла
?>
```

Вместо данного кода для построчного чтения файла в массив можно воспользоваться встроенной функцией `file(имя_файла)`.

Чтение файла с удалением тегов HTML

Иногда возникает задача прочитать файл, удалив из полученной строки все или только некоторые теги HTML, а также `<?php` и `?>`. Например, вам может потребоваться выделить из HTML-документа только текстовое содержимое, удалив графику и Flash-ролики и другие объекты. Для этого можно использовать функцию `fgetss(дескриптор, длина, допустимые_теги)`. Данная функция аналогична рассмотренной ранее функции `fgets()`. Второй и третий параметры не обязательны. Строковый третий параметр может содержать список тегов, которые требуется оставить в возвращаемой строке. Если он не указан, то удаляются все теги. Пример использования функции `fgetss()` приведен в листинге 5.21.

Листинг 5.21. Чтение файла в переменную с удалением тегов HTML (`fgetss()`)

```
<?php
  $filename="test.txt";
  $fh=fopen($filename,"r"); // открытие файла в режиме только чтения
  while (!feof($fh)) {      // пока не достигнут конец файла
```

```
    $buff=$buff+fgetss($fh,1024, "<p>,<br>,<a>");
  }
  fclose($fh);                // закрытие файла
  echo $buff;                 // вывод значения переменной
?>
```

Здесь в переменной `$buff` сохраняется содержимое файла. При этом из значения этой переменной удаляются все теги за исключением тегов абзаца, перевода строки и ссылок. Разумеется, исходный файл остается без изменений.

Кроме функции `fgetss()` с этой же целью можно использовать функцию `strip_tags(`*строка*`, `*допустимые_теги*`)`, которая удаляет из указанной строки теги, не перечисленные во втором необязательном параметре. Пример использования функции `strip_tags()` приведен в листинге 5.22.

Листинг 5.22. Чтение файла в переменную с удалением тегов HTML (`strip_tags()`)

```
<?php
  $filename="test.txt";
  $buff=file_get_contents($filename); // чтение файла в переменную
  $buff=strip_tags($buff, "<p>,<br>,<a>");
  echo $buff;                         // вывод значения переменной
?>
```

Здесь функция `strip_tags()` применяется к строке, которая получена из файла с помощью функции `file_get_contents()`.

Чтение CSV-файла

Данные в текстовом файле могут иметь табличную структуру: в каждой строке они отделяются друг от друга некоторым символом-разделителем. Текстовые файлы с такой организацией данных называются CSV-файлами (comma separated values — значения, разделенные запятой). Многие прикладные системы, работающие с таблицами и базами данных (например, MS Excel, MS Access и др.), могут экспортировать свои данные в CSV-файлы и, наоборот, импортировать CSV-файлы в свои структуры. Это позволяет различным приложениям обмениваться данными.

Для чтения данных, разделенных некоторым символом (не только запятой), служит встроенная функция

`fgetcsv(`*дескриптор*`, `*длина*`, `*разделитель*`, `*ограничитель*`)`

Данная функция, подобно рассмотренной ранее функции `fgets()`, читает строку открытого файла, но дополнительно анализирует ее и возвращает

массив данных, отделенных друг от друга в этой строке указанным разделителем. При возникновении ошибки, а также при достижении конца файла функция возвращает `false`. Таким образом, функция выделяет поля в строке CSV-файла и записывает их содержимое в массив.

Первый параметр является дескриптором открытого файла. Второй и последующие параметры не обязательны. Если параметр *длина* не указан или равен 0, то предполагается, что строка может быть сколь угодно длинной, однако при этом скорость анализа уменьшается по сравнению со случаем, когда этот параметр больше 0. Если же вы указали его, то он должен превышать длину самой большой строки в файле. Значением по умолчанию параметра *разделитель* является запятая. Если в полях могут находиться запятые, то при создании CSV-файла не стоит использовать этот символ в качестве разделителя полей. Параметр *ограничитель* — символ, который влияет на объединение строк CSV-файла и разграничение полей в строке. Значением по умолчанию является двойная кавычка. Так например, если в CSV-файле имеется единственная двойная кавычка, расположенная в его начале, то функция `fgetcsv()` вернет массив с единственным элементом, содержащим весь файл.

В листинге 5.23 приведен сценарий, считывающий содержимое файла test.csv, данные в котором разделены вертикальной чертой, и выводящий их в окно браузера (рис. 5.5). Попутно выводятся номер строки и количество найденных в ней полей.

Листинг 5.23. Чтение CSV-файла и вывод его содержимого

```
<?php
  $row = 1;
  $fh = fopen("test.csv", "r");      // открытие файла
  while (($buff = fgetcsv($fh, 300, "|")) !== false) {
    $ncol = count($buff);  // количество полей
    echo "<i>Строка $row, полей $ncol:</i><br>";
    $row++;
    for ($i=0; $i < $ncol; $i++) {  // вывод полей строки
      echo $buff[$i] ;
    }
    echo "<br>";
  }
  fclose($fh);  // закрытие файла
?>
```

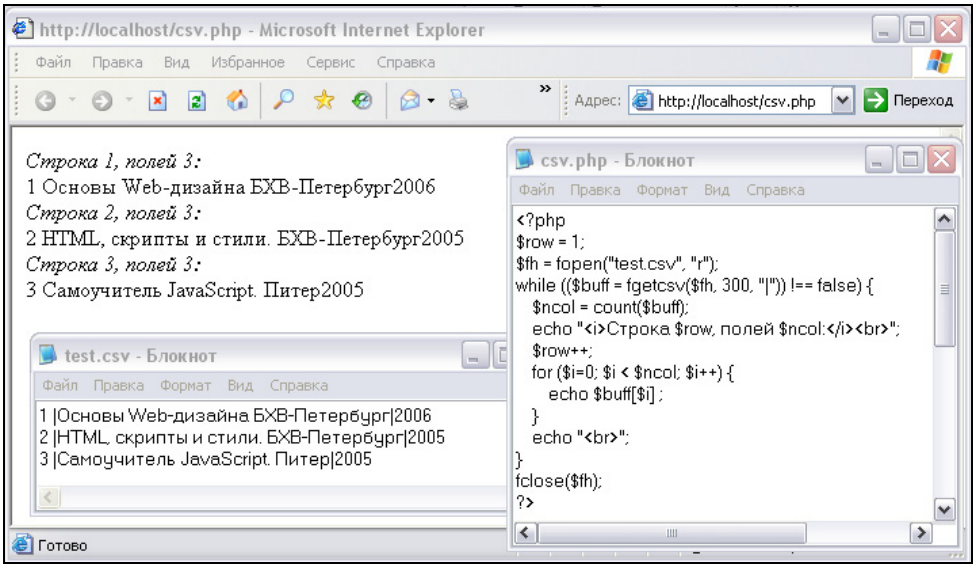


Рис. 5.5. Содержимое CSV-файла, сценарий и результат его работы в браузере

5.3.4. Запись в файл

В файл, открытый в режиме, допускающем изменение его содержимого *(см. разд. 5.3.1)*, можно записать данные. Для этого служит функция

`fwrite(`*дескриптор*`, `*строка*`, `*длина*`)`

которая возвращает количество записанных байтов или `false` в случае ошибки.

Записываемые данные представляются как строка символов. Если указан необязательный параметр *длина*, меньший чем длина строки, то будет записано данное количество байтов. В противном случае — все данные. В листинге 5.24 приведен пример сценария записи в файл текущей даты и пояснения. При этом данные записываются в две строки. Обратите внимание, что в Windows для перевода строки используются два служебных символа — `\r\n`, а в системах семейства UNIX достаточно одного — `\n`.

Листинг 5.24. Запись данных в файл

```
<?php
  $fh = fopen("myfile.txt", "w+");                    // открытие файла
  $data=date('d.m.Y')."\r\nПробная запись в файл";  // данные
  fwrite($fh,$data);       // запись данных в файл
  fclose($fh);             // закрытие файла
?>
```

5.3.5. Простой счетчик посещений страницы

В данном разделе мы рассмотрим совместное применение функций чтения и записи в файл на примере простого счетчика посещений страницы.

Текущее количество посещений Web-страницы можно хранить в текстовом файле, например, mycounter.txt. При вызове страницы следует прочитать содержимое этого файла, увеличить значение счетчика на 1 и перезаписать файл mycounter.txt с новым значением. Все это можно сделать с помощью сценария, представленного в листинге 5.25.

Листинг 5.25. Простой счетчик посещений страницы

```
<?php
  $fh = fopen("mycounter.txt", "r");
  $counter=fread($fh,20); // прочитали текущее значение счетчика
  fclose($fh);
  $counter++;             // увеличили значение счетчика на 1
  $fh = fopen("mycounter.txt", "w");
  fwrite($fh, "$counter");  // записали новое значение счетчика
  fclose($fh);
?>
```

Данный код можно сохранить в отдельном файле, например, counter.inc, а затем включить его в каждую серверную страницу сайта. Тогда вызов любой из них будет увеличивать значение счетчика. В листинге 5.26 приведен тестовый код страницы, отображающей значение счетчика (рис. 5.6). Значение счетчика выводится в таблице с помощью функции форматирования `sprintf()` как число, дополненное ведущими нулями *(см. разд. 4.7.4)*.

Листинг 5.26. Страница со счетчиком посещений

```
<?php
  include("counter.inc");  // см. листинг 5.25
  /* Значение счетчика находится в переменной $counter */
?>
<html>
  <h1>Моя Web-страница</h1>
  <table border=7>
    <tr><td align=center><b>Счетчик</b></td></tr>
```

```
    <tr><td>
      <?php echo sprintf("%'09s",$counter); ?>
    </td></tr>
  </table>
</html>
```

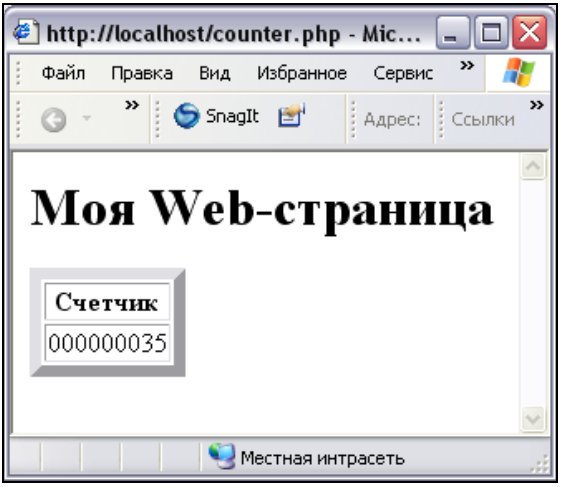


Рис. 5.6. Пример вывода счетчика на Web-странице

К теме создания счетчиков мы еще вернемся в *разд. 5.4.2*.

5.4. Работа с таблицами в текстовых файлах

Во многих случаях данные удобно представлять в виде таблиц. Долговременно табличные данные можно сохранять в текстовых CSV-файлах *(см. разд. 5.3.3)*, в которых поля отделены друг от друга каким-нибудь символом-разделителем. В сценариях табличные данные представляются двумерными массивами. В основу таких приложений, как счетчик посещений Web-страниц, баннер, гостевая книга и др., можно положить несколько функций, обеспечивающих работу с таблицами. В данном разделе мы сначала рассмотрим эти функции, а затем — их применение в перечисленных выше приложениях. Несмотря на различное назначение и некоторую специфику этих приложений, они имеют общий архетип, увидеть который я вас настоятельно призываю.

5.4.1. Функции работы с табличными данными

Данные с табличной организацией естественно сохранять в оперативной памяти с помощью двумерного массива. Такой массив содержит в качестве элементов другие массивы. Другими словами, таблица представляется массивом ее строк, а строки — массивами значений полей таблицы. Вот пример создания таблицы из четырех строк и трех столбцов:

```
$rows[]=array("a1","b1","c1");
$rows[]=array("a2","b2","c2");
$rows[]=array("a3","b3","c3");
$rows[]=array("a4","b4","c4");
```

Так например, в таблице `$rows` в 1-й строке 0-й столбец (поле) имеет значение `"a2"`, т. е. значение `$rows[1][0]` равно `"a2"`. Напомним, что по умолчанию нумерация элементов массива начинается с 0.

Вы можете использовать символьные индексы в качестве имен столбцов, как в следующем примере:

```
$row[]=array("f1"=>"a1","f2"=>"b1","f3"=>"c1");
$row[]=array("f1"=>"a2","f2"=>"b2","f3"=>"c2");
$row[]=array("f1"=>"a3","f2"=>"b3","f3"=>"c3");
$row[]=array("f1"=>"a4","f2"=>"b4","f3"=>"c4");
```

Здесь во 2-й строке поле `f2` имеет значение `"b3"`, т. е. значение `$rows[2]['f2']` равно `"b3"`.

Далее мы рассмотрим несколько функций для работы с табличными данными, которые окажутся полезными при создании конкретных приложений. Коды определений этих функций вы можете собрать вместе в одном файле, например, tablib.inc. Впоследствии этот файл можно включать в сценарии приложений с помощью функции `include()`.

Запись двумерного массива в файл

Двумерный массив будем записывать в текстовый файл построчно, разделяя поля специальным символом. Иначе говоря, требуется записать двумерный массив в CSV-файл с указанным разделителем. Сценарий, решающий эту задачу, оформим в виде функции (листинг 5.27).

Листинг 5.27. Функция записи массива в файл (`atofile()`)

```
function atofile($arr,$filename, $separator="|"){
  /* Параметры:
   * $arr — двумерный массив
```

```
   * $filename — имя файла
   * $separator — символ-разделитель (по умолчанию "|")
   * Возвращает: количество записанных байтов или false в случае ошибки
   */
  foreach($arr as $ind=>$r){
    $i=0;          // начало строки
    foreach($r as $fld=>$value){
      if ($i>0){  // если не начало строки
        $str.=$separator.$value;
      } else {
        $str.=$value;
        $i=1;
      }
    }
    $str.="\r\n";  // добавляем символ конца строки
    $i=0;
  }
  $fh=fopen($filename,"w");
  $ret=fwrite($fh,$str);
  fclose($fh);
  return $ret; // возвращаем то же, что и fwtite()
}
```

Чтение двумерного массива из файла

В листинге 5.23 приводился код сценария чтения CSV-файла и вывода значений его полей. Немного модифицировав этот сценарий, можно написать функцию чтения CSV-файла в двумерный массив, код которой представлен в листинге 5.28.

Листинг 5.28. Функция чтения CSV-файла в двумерный массив (`filetoa()`)

```
function filetoa($filename,$separator="|"){
  /* Параметры:
   * $filename — имя файла
   * $separator — символ-разделитель (по умолчанию "|")
   * Возвращает: двумерный массив или null при  неудаче
   */
  $fh = fopen($filename, "a+");
  while (($buff = fgetcsv($fh, 1024, $separator)) !== false) {
```

```
    $ret[]=$buff;  // добавляем массив в массив строк
  }
  return $ret;
}
```

Здесь файл открывается в режиме чтения и добавления, чтобы в случае его отсутствия была предпринята попытка создания нового файла.

Поиск строки в массиве

В таблице, представленной в виде двумерного массива, может потребоваться найти строку, которая удовлетворяет некоторому условию. В листинге 5.29 приведена функция, которая делает это для простых условий вида:

поле отношение значение

Здесь *поле* — индекс (числовой или символьный) элемента в массиве-строке; *отношение* — строка, содержащая оператор сравнения, например, "==", ">" и т. п.; *значение* — образец значения.

Функция, приведенная в листинге 5.29, возвращает номер первой найденной строки, удовлетворяющей условию, или –1 при неудачном поиске.

Листинг 5.29. Функция поиска строки в массиве (`find()`)

```
function find($arr,$fld,$val,$rel="=="){
  /* Параметры:
   * $arr — двумерный массив, в котором осуществляется поиск
   * $fld — поле, т. е. индекс элемента в строке
   * $val — значение
   * $rel — отношение (по умолчанию — "==")
   * Возвращает: номер (индекс) строки, удовлетворяющей условию
   * $fld $rel $val, или -1 при  неудаче
   */
  $criterion='return $r[$fld]'. "$rel \"$val\";"; /* строка с условием
                                                     поиска  */
  foreach($arr as $ind=>$r){
    if (eval($criterion)===true){
      return $ind;     // возвращаем индекс
    }
  }
  return -1;
}
```

Обратите внимание, что здесь сначала формируется строка `$criterion`, содержащая условие поиска, которая затем вычисляется с помощью функции `eval()`.

В листинге 5.30 приведен тестовый сценарий с применением функции `find()`.

Листинг 5.30. Тестовый сценарий с применением функции поиска

```
<?php
  $rows[]=array("a1","b1","c1");
  $rows[]=array("a2","b2","c2");
  $rows[]=array("a1","b3","c3");
  $rows[]=array("a4","b4","c4");
  function find($arr,$fld,$val,$rel="=="){
    //  см. листинг 5.29
  }
  echo find($rows,"1","b2","=="); //   1
  echo "<br>";
  echo find($rows,"0","a1","=="); //   0
  echo "<br>";
  echo find($rows,"2","a4","=="); //  -1
?>
```

Выборка строк в массиве

Немного модифицировав код листинга 5.29, можно получить функцию, возвращающую массив номеров (индексов) всех строк в двумерном массиве, которые удовлетворяют заданному условию. Если таких строк не оказалось, то возвращается `null`. Код функции `filter()`, которая производит выборку или фильтрацию строк в двумерном массиве, представлен в листинге 5.31.

Листинг 5.31. Функция выборки строк в массиве (`filter()`)

```
function filter($arr,$fld,$val,$rel="=="){
  /* Параметры:
   * $arr — двумерный массив, в котором осуществляется поиск
   * $fld — поле, т. е. индекс элемента в строке
   * $val — значение
   * $rel — отношение (по умолчанию — "==")
   * Возвращает: массив номеров (индексов) всех строк,
```

```
   * удовлетворяющих условию  $fld.$rel.$val,
   * или null при  неудаче
   */
  $criterion= $criterion='return $r[$fld]'. "$rel \"$val\";"; /* строка
                                                        с условием выборки */
  foreach($arr as $ind=>$r){
    if (eval($criterion)===true){
      $ret[]=$ind;  // добавляем индекс в массив
    }
  }
  return $ret;  // возвращаем массив индексов или null
}
```

В листинге 5.32 приведен тестовый сценарий с применением функции `filter()`.

Листинг 5.32. Тестовый сценарий с применением функции выборки

```
<?php
  $rows[]=array("a1","b1","c1");
  $rows[]=array("a2","b2","c2");
  $rows[]=array("a1","b3","c3");
  $rows[]=array("a4","b4","c4");
  function filter($arr,$fld,$val,$rel="==") {
    //  см. листинг 5.31
  }
  $x=find($rows,"0","a1","==");
  print_r($x);     // выводит массив {0, 2}
?>
```

5.4.2. Сложный счетчик посещений страницы

В данном разделе мы рассмотрим приложение, при создании которого используются приемы, пригодные и для других проектов, таких как баннер и гостевая книга.

Создание простого счетчика посещений Web-страницы рассматривалось в *разд. 5.3.5*. Его простота заключалась в том, что файл, хранивший состояние счетчика, содержал лишь одно число — количество посещений страницы,

а сценарий считывал это число, увеличивал его на 1 и сохранял новое значение в том же файле. Однако можно создать более сложный счетчик, который будет хранить не только количество посещений, но и сведения о пользователе (например, IP-адрес компьютера, название браузера, URL-адрес страницы и т. п.).

Очевидно, что сложный счетчик можно создать на основе текстового файла с табличными данными. В листинге 5.33 представлен код, реализующий счетчик, который сохраняет свое состояние в двухстолбцовой таблице. Первый столбец содержит IP-адрес компьютера, который запросил страницу со сценарием счетчика, а второй столбец — количество посещений страницы данным пользователем. Обратите внимание, что CSV-файл предварительно создавать не надо: если он отсутствует, то будет создан автоматически функцией `filetoa()`. Основная задача сценария состоит в том, чтобы добавить в таблицу новую запись (строку) или изменить существующую. Что именно делать, зависит от того, успешен ли поиск в таблице IP-адреса посетителя. Если пользователь впервые посетил страницу, то добавляется новая запись, а в противном случае соответствующая ему запись изменяется. Переменная `$counter` содержит общее количество посещений страницы всеми пользователями. Определения функций, необходимых для работы сценария, сосредоточены в файле tablib.inc. Их коды были приведены в листингах 5.27 и 5.28.

Листинг 5.33. Сложный счетчик посещений страницы

```
<?php
  include('tablib.inc');          /* определения функций
                                     filetoa(), find() и atofile() */
  $filename="mycounter.txt";      // файл с таблицей состояния счетчика
  $rows=filetoa($filename,"|");   // читаем файл с таблицей в массив
  $ip=$_SERVER['REMOTE_ADDR'];    // IP посетителя
  if ($rows!=NULL){               // если что-то есть
    $i=find($rows,"0",$ip,"==");  // поиск строки с данным IP
    if ($i>=0) {  /* если уже есть такой IP,
                     то увеличиваем количество посещений на 1 */
      $rows[$i][1]=$rows[$i][1]+1;
    } else {  /* иначе добавляем строку с данным IP и количеством
                 посещений 1 */
      $rows[]=array($ip,1);
    }
  } else{                     // в файле счетчика ничего нет
```

```
    $rows[]=array($ip, 1);   /* добавляем строку с данным IP и
                                количеством посещений 1 */
  }
  atofile($rows,$filename, "|");  // записываем массив в файл
  foreach ($rows as $r){  // вычисляем общее количество посещений
    $counter=$counter+$r[1];
  }
?>
```

На страницу обычно выводится общее количество ее посещений. Если сохранить приведенный выше код в отдельном файле, например, counter.inc, то для отображения суммарного значения счетчика можно воспользоваться сценарием листинга 5.26.

Вы можете также создать отдельную страницу, которая будет отображать детальное состояние счетчика, например, в виде таблицы. В листинге 5.34 приведен код сценария, который отображает таблицу состояния счетчика.

Листинг 5.34. Страница с детальным состоянием счетчика посещений

```
<?php
  include('tablib.inc');  // здесь понадобится только функция filetoa()
  $filename="mycounter.txt";
  $rows=filetoa($filename,"|");
  echo "<h2>Счетчик посещений</h2>
        <table border=1><tr><th>№п/п</th><th>IP-адрес</th>
          <th>Количество</th></tr>";
  foreach ($rows as $r){
    $i++;
    echo "<tr><td>$i</td><td>$r[0]</td><td>$r[1]</td></tr>";
    $counter=$counter+$r[1];  // общее количество посещений
  }
  echo "</table><br>Общее количество: <b>$counter</b>";
?>
```

Таблица, с которой работает счетчик, может содержать и большее количество столбцов для хранения дополнительных параметров, которые являются элементами суперглобального массива `$_SERVER` (см. справочное руководство по PHP).

Примечание

Отображать на странице табличные данные из текстовых файлов можно и с помощью элемента ActiveX STD, который вставляется в HTML-код с помощью тега `<object>`. Как это делается, рассмотрено в *разд. 3.12*.

Итак, мы рассмотрели создание счетчика посещений страниц собственного сайта. Действительно, страница со счетчиком должна содержать серверный сценарий, например, вставку PHP-кода вида:

```
<?php include("counter.inc"); ?>
```

Разработчик сайта, не имеющего доступ к PHP, не сможет воспользоваться описанной выше технологией.

Теперь рассмотрим создание счетчика, который можно предоставить всем желающим, в том числе и не использующим PHP. Иначе говоря, займемся разработкой свободно распространяемого счетчика.

Счетчик, о котором пойдет речь, вставляется в HTML-код страницы с помощью тега

```
<img src=URL_адрес_счетчика>
```

Здесь *`URL_адрес_счетчика`* — URL-адрес файла со сценарием, реализующим функции счетчика и создающим графическое изображение со значением счетчика. Это изображение отображается на странице пользователя с помощью тега `<img>` в том месте, где он позиционирован. В листинге 5.35 приведен пример HTML-кода страницы с таким счетчиком, а на рис. 5.7 показан ее вид в окне браузера.

В теге `<img>` в атрибуте `src` указан Web-сервер, установленный на вашем компьютере. Там же должен находиться и сценарий counter.php. Однако html-файл из листинга 5.35 может находиться в любом месте на вашем компьютере. Это замечание — для экспериментов на локальном компьютере. В общем же случае HTML-страница со счетчиком может находиться на одном Web-сервере, а сценарий счетчика (counter.php) — на другом.

Листинг 5.35. HTML-код с распространяемым счетчиком посещений страницы

```
<html>
  <head><title>Моя страница</title></head>
  <body bgcolor=#f0f0f0>
    <center><h1>Моя страница</h1></center>
    <! Вставка счетчика >
    <img src='http://localhost/counter.php'>
  </body>
</html>
```

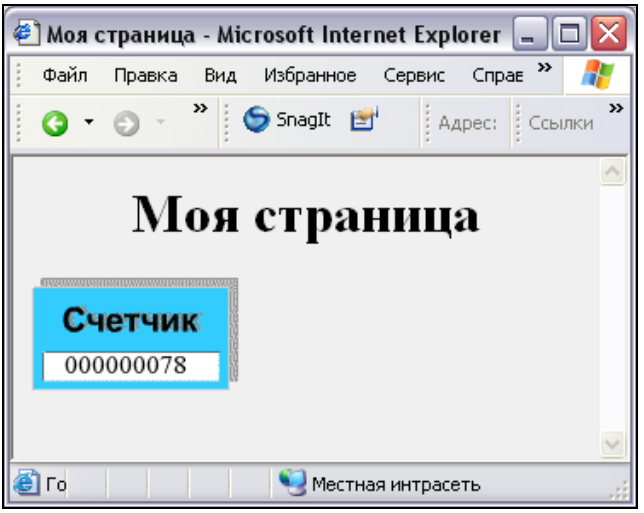


Рис. 5.7. Страница со счетчиком посещений в виде графического изображения

Сценарий, реализующий функции и создающий графическое изображение счетчика, представлен в листинге 5.36.

Листинг 5.36. Сценарий распространяемого графического счетчика

```
<?php
$filename="mycounter.txt";        // файл с таблицей состояния счетчика
$rows=filetoa($filename,"|");     // читаем из файла в массив
$page= $_SERVER['HTTP_REFERER'];/* адрес страницы, обратившейся
                                   к счетчику */
if ($rows!=NULL){                 // если что-то есть
  $i=find($rows,"0",$page,"==");
  if ($i>=0) {
    $rows[$i][1]=$rows[$i][1]+1;
    $counter=$rows[$i][1];
  } else {
    $rows[]=array($page,1);
    $counter=1;
  }
} else{                           // если массив пуст
       $rows[]=array($page, 1);
       $counter=1;
```

```
}
// Значение счетчика находится в переменной $counter
atofile($rows,$filename, "|");  // запись массива в файл
/*************************************************************/
$counter=sprintf("%'09s",$counter); /* форматируем значение счетчика как
                                    строку с добавлением ведущих нулей */
/**** Создание изображения счетчика ****************************/
header("Content-type: image/gif");  // HTTP-заголовок о типе данных
$img = imagecreatefromgif("counter.gif");  /* создаем ресурс изображения
                                    из графического файла counter.gif*/
/**** Параметры для внедряемого текста — значения счетчика: ****/
$color = imagecolorallocate($img, 255, 0, 0);   // красный цвет
$font="times.ttf"; $fontsize=12;                // шрифт
$angle=0; $x=20; $y=57;             // угол наклона и координаты
/* Внедряем в изображение отформатированное значение счетчика: */
imagettftext($img, $fontsize,$angle,$x,$y,$color,$font,$counter);
imagegif($img);                     // выводим изображение
/**** Определения функций ***************************/
function find($arr,$fld,$val,$rel="=="){
// см. листинг 5.29
}
function atofile($arr,$filename, $separator="|"){
// см. листинг 5.27
}
function filetoa($filename,$separator="|"){
// см. листинг 5.28
}
?>
```

В данном примере идентифицируется адрес страницы, с которой произошло обращение к сценарию счетчика (`$_SERVER['HTTP_REFERER']`).

Внимание

Для формирования изображения счетчика применяются специальные функции PHP из библиотеки работы с графикой. Эта библиотека, являющаяся расширением PHP (для Windows — файл php_gd2.dll), не устанавливается по умолчанию, но входит в дистрибутив модуля PHP. Как устанавливаются расширения PHP, рассказано в *приложении 2*. За описанием большого количества функций этой библиотеки следует обратиться к справочному руководству по PHP.

На сервере находится небольшой графический файл, представляющий подложку для значения счетчика (в примере — counter.gif). Его можно создать с помощью графического редактора. Из этого файла через функцию `imagecreatefromgif()` создается ресурс соответствующего изображения, над которым можно выполнять различные манипуляции, в том числе внедрить другое изображение или текст. Посредством функции `imagettftext()` в изображение подложки счетчика внедряется его значение, отформатированное функцией `sprintf()`, а с помощью функции `imagegif()` результирующее изображение возвращается браузеру и отображается им.

Примечание

В листинге 5.36 определения функций `find()`, `atofile()` и `filetoa()` вставлены непосредственно, а не путем включения отдельного файла. Попытка использовать включение файла с помощью `include()` приводит к тому, что графическое изображение счетчика не отображается в браузере. Это связано с особенностями применения функции `header()`.

5.4.3. Баннер

Баннер (banner — знамя, символ, газетный заголовок) представляет собой небольшое графическое изображение, размещаемое на Web-странице для рекламы сайтов, товаров и услуг. Это может быть статическое изображение из файлов формата GIF, JPEG, PNG, или анимационное. Анимационные изображения сохраняются в файлах формата GIF89a или SWF (Flash-ролики).

Щелчок на баннере может приводить к переходу на определенную Web-страницу (страницу спонсора, хозяина баннера). Кроме того, показ страницы с баннером и/или щелчок на баннере может приводить к выполнению серверного сценария, который ведет учет этих действий с целью последующего анализа эффективности баннерной рекламы, назначения бонусов владельцам страниц, на которых размещен баннер, и т. п.

Таким образом, баннер можно создать как графическое изображение, заключенное в контейнерный тег, в котором можно указать адрес перехода: `<a>` или `<form>`. В первом случае баннер оказывается графической ссылкой, а во втором играет роль кнопки типа `submit` (**Отправить**). Однако баннер — это не просто графическая ссылка или форма с графической кнопкой. Дело в том, что в HTML-коде баннера нельзя указывать адрес графического файла, поскольку в противном случае владелец сайта мог бы указать его по своему усмотрению. В результате спонсор (хозяин баннера) начислял бы бонусы за показ не своей рекламы. Далее мы рассмотрим, как этого избежать.

Не трудно заметить, что со структурно-функциональной точки зрения, баннер похож на сложный счетчик, рассмотренный в *разд. 5.4.3*. Действительно, баннер, как и счетчик, получает некоторые данные от посетителя страницы и сохраняет их в таблице, расположенной на сервере. В следующем разделе мы увидим, что подобным образом может быть устроена и гостевая книга.

Рассмотрим создание баннера в виде графической ссылки:

```
<a href=banner.php?click=yes><img src= banner.php></a>
```

Здесь адрес ссылки (значение атрибута `href`) и адрес источника графического изображения (значение атрибута `src`) совпадают. Это в данном случае файл banner.php (вообще говоря, можно использовать URL-адрес), который содержит сценарий учета показов страницы с баннером и количества щелчков на нем, а также выводит в браузер графическое изображение. Данная конструкция позволяет отдельно учитывать щелчки на баннере и просто показы страницы с баннером.

Обратите внимание, что в качестве адреса изображения указан не графический файл, а файл со сценарием. Если недобросовестный владелец страницы изменит значение атрибута `src`, то показы баннера не будут учитываться. Однако щелчки на изображении будут обрабатываться сценарием. И это правильно, поскольку сценарий banner.php не только учтет щелчок, но и выполнит переход на страницу хозяина баннера (спонсора). Если владелец страницы изменит только значение атрибута `href`, то щелчки на баннере не будут учитываться, а показы обработаются.

При щелчке на баннере сценарию banner.php будет передано значение `yes` переменной `click`. Эта информация понадобится, чтобы узнать, был ли щелчок на баннере.

В листинге 5.37 приведен пример простого HTML-кода страницы, содержащей баннер, а в листинге 5.38 — сценарий работы с баннером (файл banner.php).

Листинг 5.37. Страница с баннером

```
<html>
  <h1>Моя страница</h1>
  <a href="http://localhost/banner.php?click=yes">
  <img src="http://localhost/banner.php"></a>
</html>
```

Листинг 5.38. Сценарий работы с баннером (banner.php)

```
<?php
  if ($_REQUEST['click']=="yes"){   // был щелчок на баннере
    header("Location:anypage.htm"); /* переход на страницу
                                       хозяина баннера */
  } else {
    header("Content-type:image/gif");      // посылаем заголовок
    $img=imagecreatefromgif("banner.gif"); /* создаем изображение как
                                        ресурс из графического файла */
    imagegif($img);                        // выводим изображение
  }
  include("bcounter.php");                 // счетчик
?>
```

Внимание

Данный сценарий будет работать, если установлена библиотека функций PHP для работы с графикой. Для Windows это — php_gd2.dll. Установка расширений PHP обсуждается в *приложении 2*. Описание функций этой библиотеки можно найти в справочном руководстве по PHP.

В сценарии листинга 5.38 включается файл bcounter.php (разумеется, имя может быть произвольным), в котором содержится сценарий счетчика щелчков на баннере и просто его показа. Для проверки работоспособности приведенного выше кода на первых порах в качестве счетчика можно использовать код листинга 5.33. Напомним, что этот сценарий записывает в таблицу IP-адрес компьютера пользователя и количество его посещений страницы. Однако для нашего баннера такая информация не вполне подходит. Во-первых, важен не IP-адрес посетителя, а адрес страницы с баннером. Во-вторых, наш баннер потенциально может фиксировать как его показы, так и щелчки. Поэтому таблица, в которой сохраняется информация о работе баннера, должна содержать, по крайней мере, три столбца:

- адрес Web-страницы с баннером;
- количество показов (т. е. посещений или загрузок страницы);
- количество щелчков на баннере.

Пример сценария, реализующего счетчик для баннера (включаемый файл bcounter.php), приведен в листинге 5.39.

Листинг 5.39. Счетчик показов и щелчков на баннере (bcounter.php)

```
<?php
$isclick=($_REQUEST['click']=="yes"); // был ли щелчок на баннере
include('tablib.inc');      /* определения функций
                               filetoa(), find() и atofile() */
$filename="bcounter.txt";   // файл с таблицей состояния счетчика
$ref= $_SERVER['HTTP_REFERER'];  // адрес страницы
$rows=filetoa($filename,"|");    // читаем файл с таблицей в массив
if ($rows!=NULL){                // если что-то есть
  $i=find($rows,"0",$ref,"==");  /* поиск строки с данным
                                    адресом страницы */
  if ($i>=0) {                   // если уже есть такая страница
    if ($isclick) {              // был щелчок
      $rows[$i][2]=$rows[$i][2]+1;
    } else {                     // не было щелчка
      $rows[$i][1]=$rows[$i][1]+1;
    }
  } else {                       // иначе добавляем строку
    if ($isclick) {
      $rows[]=array($ref,1,1);
    } else {
      $rows[]=array($ref,1,0);
    }
  }
} else{                          // в файле счетчика ничего нет
  if ($isclick) {
    $rows[]=array($ref, 1, 1);
  } else {
    $rows[]=array($ref, 1, 0);
  }
}
atofile($rows,$filename, "|");   // записываем массив в файл
?>
```

Таким образом, чтобы проверить работу рассмотренного выше баннера, требуется создать:

- страницу с баннером (листинг 5.37);
- сценарий работы с баннером (banner.php, листинг 5.38);

- сценарий счетчика показов и щелчков на баннере (bcounter.php, листинг 5.39). Файл bcounter.txt, содержащий информацию о состоянии счетчика, создается автоматически;
- файл tablib.inc с определениями функций `filetoa()`, `find()` и `atofile()` (листинги 5.27—5.29).

Загрузите в браузер страницу с баннером, перезагрузите ее несколько раз, выполните щелчки на баннере и посмотрите в текстовом редакторе содержимое файла bcounter.txt. Пример этого файла в Блокноте Windows показан на рис. 5.8.

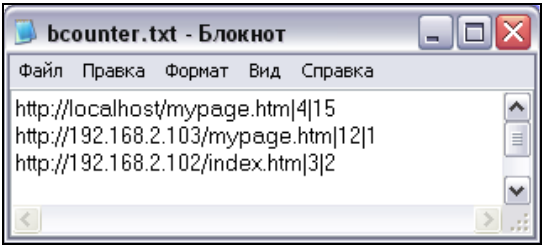


Рис. 5.8. Пример файла с состоянием счетчика показов и щелчков на баннере

5.4.4. Гостевая книга

Гостевая книга предназначена для хранения на сервере сообщений от посетителей вашего сайта. Работа с гостевой книгой предполагает добавление записей посетителями и их коррекцию владельцем книги, который также может отвечать на сообщения посетителей. Так называемые форумы устроены аналогично, хотя и имеют свои особенности. Хранить данные гостевой книги можно в текстовом CSV-файле, а сценарий работы с ним аналогичен сценариям для счетчика посещений и баннера *(см. разд. 5.4.2, 5.4.3)*. Тем не менее, в проекте гостевой книги есть ряд особенностей.

Запись в гостевой книге обычно содержит следующие поля:

- имя посетителя;
- город (место);
- URL-адрес сайта посетителя;
- адрес электронной почты посетителя;
- текст сообщения посетителя;
- ответ хозяина гостевой книги;
- дата и время отправки сообщения посетителем.

Возможны различные алгоритмы работы с гостевой книгой. Вот один из простейших:

1. При обращении к гостевой книге в браузер загружается страница, содержащая уже имеющиеся в книге записи, а также кнопку **Добавить запись**.
2. Щелчок на этой кнопке загружает в браузер другую страницу с формой, содержащей поля ввода данных и кнопку **Добавить запись**.
3. Щелчок на этой кнопке приводит к выполнению сценария, который анализирует принятые данные, введенные в форму. Если результат анализа положителен (по некоторому критерию), то данные записываются в файл хранения гостевой книги, а в противном случае — нет.
4. Завершение работы сценария на предыдущем этапе приводит к переходу на начальную страницу, отображающую все записи гостевой книги. Если новая запись была добавлена, то теперь она отображается.

Для реализации данного алгоритма потребуются три файла (имена, разумеется, могут быть произвольными):

- gbook.php — вывод всех записей гостевой книги из файла mygbook.txt;
- gbadd.php — вывод формы ввода данных посетителя, а также их анализ и запись в файл mygbook.txt;
- mygbook.txt — файл хранения данных гостевой книги (его не надо создавать предварительно).

В листинге 5.40 приведен сценарий, выводящий существующие записи гостевой книги и форму, содержащую единственную кнопку **Добавить запись** типа `submit`, щелчок на которой приведет к переходу на страницу gbadd.php. Обратите внимание, что записи гостевой книги выводятся в порядке, противоположном тому, в котором они расположены в файле хранения. Таким образом, более поздние записи оказываются наверху. Пример выполнения данного сценария показан на рис. 5.9.

Листинг 5.40. Вывод содержимого гостевой книги (gbook.php)

```
<?php
include('tablib.inc');          /* содержит определение filetoa()
                                   (см. листинг 5.28) */
$filename="mygbook.txt";        // файл с данными гостевой книги
$rows=filetoa($filename,"|");   // читаем файл в массив
/* Форма с кнопкой  для перехода к странице добавления записи */
echo "<center><h2>Моя гостевая книга</h2>";
```

```
echo "<form action=gbadd.php method=POST>
        <input type=submit name=addrecord value='Добавить запись'>
      </form><br>";
/* Вывод содержимого гостевой книги */
if ($rows!=NULL){  // если что-то есть
  krsort($rows);   // сортируем массив по убыванию индексов
  $i=0;
  foreach ($rows as $ind=>$r){
    $i++;
    echo "<b>$i. {$r[0]}</b>  {$r[1]} {$r[6]}<br>
          <b>Сайт:</b> <a href={$r[2]}>{$r[2]}</a><b> E-mail:</b>
          <a href={$r[3]}>{$r[3]}</a><br>
          <br><textarea rows=4 cols=60>{$r[4]}</textarea><br>";
    if ($r[5]){
      echo "<b>Ответ:</b><br>
            <textarea rows=4 cols=60>{$r[5]}</textarea>";
    }
    echo "<hr>";  // горизонтальная линия
  }
}
if ($i>0){
  echo "Всего записей: $i";
}else{
  echo "Нет записей";
}
?>
```

Щелчок на кнопке **Добавить запись** (`name='addrecord'`) приведет к переходу на страницу gbadd.php, код которой представлен в листинге 5.41. Эта страница выводит форму с полями и кнопкой типа `submit` (`name='adddok'`) **Добавить запись** (рис. 5.10). Щелчок на этой кнопке передает данные формы этой же странице — gbadd.php. Интерпретатор PHP начинает работать с ней заново. Теперь значение `$_POST['addrecord']` не определено, т. к. среди элементов формы, из которой передавались данные, нет элемента с именем `'addrecord'`. Зато определено и не пусто значение `$_POST['addok']`. Следовательно, форма ввода данных не отображается, а происходит переход к странице gbook.php, отображающей все записи гостевой книги. При этом сначала проверяется, не пусто ли поле ввода сообщения. Если оно пусто, то выполнение сценария завершается, и данные формы не записываются в файл хранения гостевой книги.

В противном случае данные полей формы, а также пустой ответ хозяина книги и текущая дата-время добавляются в массив, который затем записывается в файл mygbook.txt. Обратите внимание, что сообщение вводится пользователем в HTML-элемент `<textarea>` и оно может содержать невидимые символы перевода строки. При записи сообщения в файл мы их заменяем на символ пробела с помощью функции `str_replace()`. Если не делать этого, нарушится структура данных в файле хранения гостевой книги.

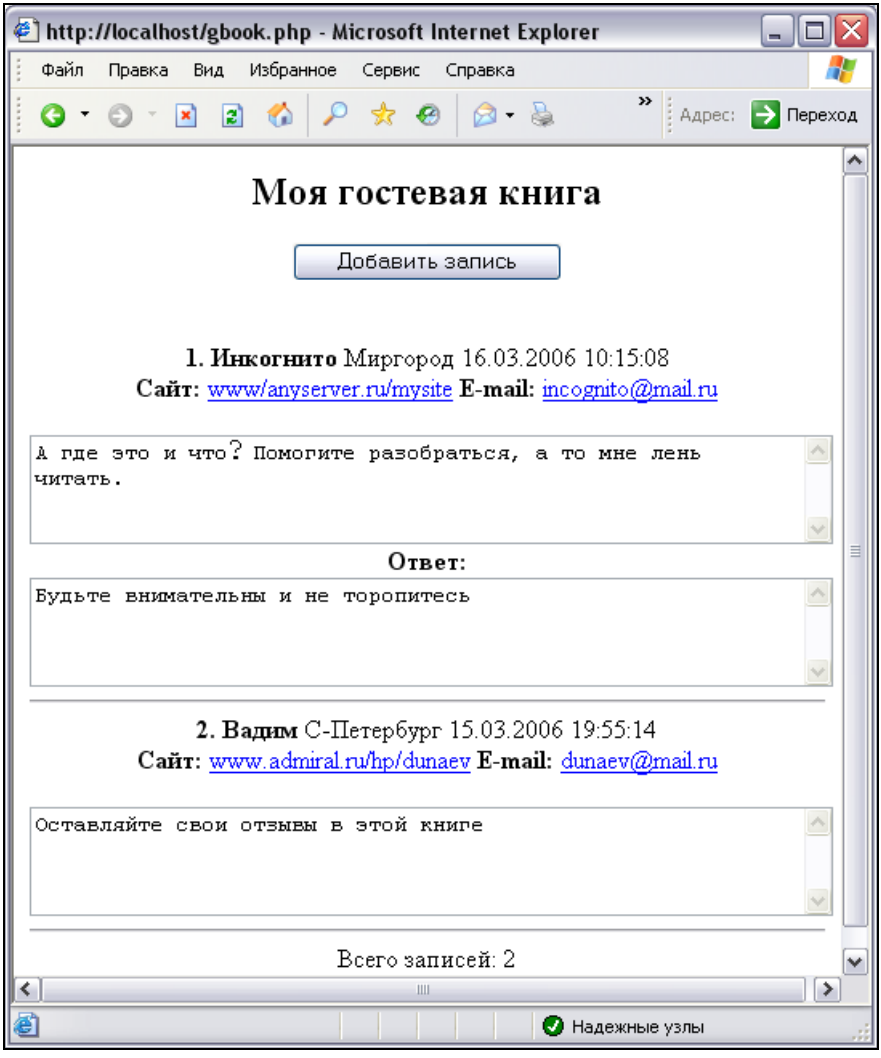


Рис. 5.9. Страница, отображаемая сценарием из листинга 5.40

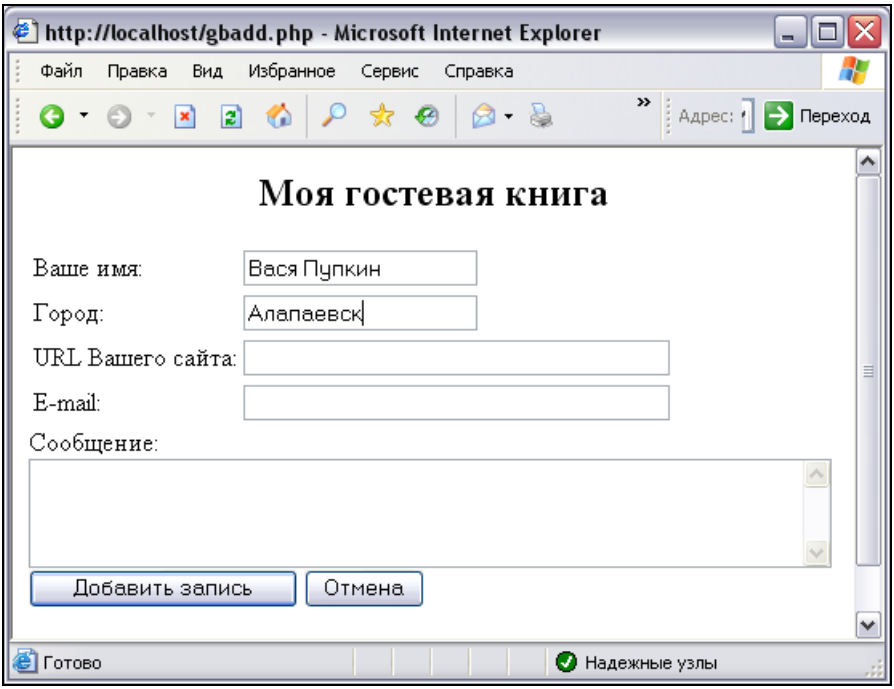


Рис. 5.10. Форма ввода данных посетителем гостевой книги

Листинг 5.41. Отображение формы ввода и добавление данных в гостевую книгу (gbadd.php)

```
<?php
if ($_POST['addrecord']){  // если требуется добавить запись
  /* Форма для ввода данных */
  echo "<html>
          <center><h2>Моя гостевая книга</h2></center>
          <form action=gbadd.php method=POST>
            <table>
              <tr><td>Ваше имя:</td>
                <td><input type=text name=username size=20></td></tr>
              <tr><td>Город:</td>
                <td><input type=text name=city size=20></td></tr>
              <tr><td>URL Вашего сайта:</td>
                <td> <input type=text name=site size=40></td></tr>
              <tr><td>E-mail:</td>
```

```
                <td><input type=text name=email size=40></td></tr>
            </table>
            Сообщение:<br>
            <textarea  name=msg rows=4 cols=60></textarea><br>
            <input type=submit name=addok value='Добавить запись'>
            <input type=reset name=reset value='Отмена'>
          </form><br>
          </html>";
} elseif ($_POST['addok']) {      // если подтвердили ввод в форму
  header("Location:gbook.php");  /* Переход к странице
                                    показа записей гостевой книги */
  if (!$_POST['msg']){exit;};    // если сообщение пусто, то выход
  include('tablib.inc');         /* содержит определения функций
                                    filetoa() (см. листинг 5.28)
                                    и atofile() (см. листинг 5.27) */
  $filename="mygbook.txt";       // файл хранения гостевой книги
  $rows=filetoa($filename,"|");  // читаем файл в массив
  /* Добавляем запись в массив */
  $rows[]=array(trim($_POST['username']),
       trim($_POST['city']),
       trim($_POST['site']),
       trim($_POST['email']),
       str_replace("\r\n"," ",trim($_POST['msg'])),
       '',
       date("d.m.Y H:i:s"));
  atofile($rows,$filename,"|");  // запись массива в файл
}
?>
```

Итак, посетитель может просматривать и добавлять записи в гостевую книгу. Однако на этом проект гостевой книги не заканчивается. Необходимо еще создать защищенную паролем страницу для редактирования и ввода ответов, которой может пользоваться хозяин гостевой книги. Подробно эту задачу мы здесь не будем рассматривать. Ограничимся лишь общими замечаниями.

Страница, на которой пользователь может корректировать записи и вводить ответы, устроена аналогично странице показа всех записей. Ее основная особенность заключается в следующем. Кроме полей, выводимых для обычных посетителей, для каждой записи следует вывести еще и поле для ввода ответов, а также кнопки типа `submit`: **Принять изменения** и **Удалить**. В другом,

лучшем, варианте вместо этих кнопок для каждой записи можно вывести пару флажков с такими же надписями. Тогда понадобится только одна кнопка **Принять изменения** типа `submit`. При щелчке на ней следует перейти к сценарию, который перезаписывает (а не добавляет) или удаляет отмеченные записи гостевой книги.

Гостевая книга может содержать сотни и тысячи записей, и тогда выводить на страницу все записи одновременно оказывается неэкономично. Лучше показывать по 3—5 записей. Для этого на странице следует разместить панель навигации, например, с кнопками **Вперед**, **Назад** и т. д.

Анализ данных, введенных посетителем (в частности, проверку наличия сообщения), лучше выполнять не на стороне сервера, как в рассмотренном выше примере, а на стороне клиента посредством сценария, написанного на JavaScript. Этот сценарий должен вызываться по событию `onsumit` *(см. разд. 3.16)*, но его код может быть сформирован на стороне сервера (`echo "<SCRIPT> ... </SCRIPT>"`).

5.4.5. Форум

По существу форум устроен аналогично гостевой книге. Главная его особенность состоит в том, что посетители могут выбирать тему обсуждения из уже существующих, а также создавать новые темы. Для каждой темы можно создать на сервере отдельный текстовый CSV-файл для хранения данных посетителей. Кроме того, там должен быть еще один файл, в котором хранятся соответствия между названиями тем и именами файлов с информацией по данным темам. Назовем этот файл каталогом.

Главная страница форума содержит, кроме прочего, список всех тем, имеющихся в каталоге. При выборе темы из этого списка серверный сценарий определяет по каталогу соответствующий ей файл и формирует страницу с данными, которая отображается браузером. Посетитель форума может добавить запись, как и в случае гостевой книги.

Как на главной странице форума, так и на странице, отображающей информацию по выбранной теме, посетитель может щелкнуть на кнопке **Создать тему**. В результате должна появиться форма для ввода названия темы. Эти сведения отправляются серверному сценарию, который проверяет, имеется ли в каталоге полученное название темы. Если нет, то название темы и соответствующее ей сгенерированное каким-либо образом имя файла добавляются в каталог. Затем создается новый пустой файл с этим именем и формируется страница без данных, но с кнопкой **Добавить запись**. Что делать далее, вам уже известно. Если полученное от посетителя название темы не является

новым, то в браузере отображается страница с данными по уже существующей теме.

Таков, в общих чертах, алгоритм работы форума. Разумеется, его можно развить в различных направлениях. Попробуйте реализовать этот проект, даже если вам не нужен собственный форум. Работая над ним, вы многому научитесь в плане программирования.

5.5. Работа с базами данных

5.5.1. Общие сведения о базах данных

Общим для таких приложений, как счетчик посещений Web-страниц, баннер, гостевая книга, форум, ведение прайс-листов, учет продаж и т. п., является работа с хранилищем данных. В случае простых приложений, а также при относительно небольшом объеме данных, последние можно хранить в текстовых файлах. Однако если ваш проект предполагает использование несколько таблиц, содержащих тысячи и более записей, выборку нужных сведений по различным критериям, а также более или менее сложный их анализ, то следует подумать о применении баз данных.

База данных представляет собой хранилище данных, представленное одним или несколькими файлами. В настоящее время наиболее широко используются так называемые реляционные базы данных, в которых информация организована в виде таблиц. Таблицы могут быть связаны между собой, образуя сложную систему. Создание баз данных и доступ к ним обеспечивается системой управления базами данных (СУБД).

PHP имеет наборы специальных функций для работы со многими СУБД, такими как Microsoft SQL Server (MS SQL), MySQL, Oracle, PostgreeSQL, mSQL, Informix, Sybase, Ingres и IBM DB2. Кроме того, PHP поддерживает стандарт ODBC (Open Database Connectivity, открытый интерфейс доступа к базам данных), который поддерживают почти все реляционные СУБД. Используя ODBC, можно обращаться к таким СУБД, как MS Access.

СУБД устанавливается независимо от PHP. Чтобы работать с ней посредством PHP-сценария, следует установить соответствующую данной СУБД библиотеку функций. Например, при работе в Windows для СУБД MySQL версии не старше 4.0 требуется библиотека php_mysql.dll, для MySQL версии 4.1 и старше — php_mysqli.dll; для MS SQL Server — php_mssql.dll; для PostgreeSQL — php_pgsql.dll. При установке PHP по умолчанию библиотеки

работы с базами данных не устанавливаются. Установка библиотек (расширений PHP) описана в *приложении 2*.

Библиотеки для различных СУБД содержат разные наборы функций. Если вы собираетесь использовать базу данных в своем приложении, то должны узнать, какие СУБД поддерживаются вашей хостинговой компанией. От этого зависит, какие функции работы с базой данных вы будете использовать в своих сценариях.

Большинство современных СУБД поддерживают язык SQL (Structured Query Language, язык структурированных запросов), посредством которого формируются выражения, называемые запросами к базе данных. Запросы, выполняемые СУБД, позволяют создавать и удалять таблицы, а также добавлять, редактировать, удалять и выбирать данные. Кроме того, они осуществляют и обработку данных. Результатом выполнения SQL-запроса к базе данных является набор записей (временная таблица), удовлетворяющих условию запроса. Если таких записей в базе данных не оказалось, то возвращаемая таблица будет пустой.

Чтобы работать с базой данных, необходимо знать язык SQL. Наиболее часто используемые выражения этого языка имеют довольно простой синтаксис. Для отправки SQL-запросов к базе данных из PHP-сценария используются специальные функции, принимающие в качестве параметра строку, содержащую SQL-выражение.

Для эффективного применения SQL-запросов знание языка SQL еще не достаточно. Кроме этого, необходимо понимать, как устроены и взаимосвязаны таблицы базы данных. Однако изложение основ языка SQL и теории реляционных баз данных выходит за рамки данной книги. В несложных проектах база данных содержит одну или несколько таблиц. Например, для счетчика посещений Web-страницы, баннера, гостевой книги и даже форума вполне можно обойтись одной таблицей.

В случае одной таблицы задача проектирования базы данных сводится к правильному определению столбцов (полей) — их имен, типов и размеров. База данных может состоять из нескольких не связанных между собой таблиц. Этот случай принципиально не отличается от однотабличной базы данных. Если вам необходимо спроектировать систему из нескольких связанных таблиц, а вы новичок в этом деле, то обратитесь к специальной литературе по проектированию баз данных. В этой книге мы рассмотрим создание гостевой книги на основе простой однотабличной базы данных. Главная наша цель при этом состоит в том, чтобы понять, как работать с базой данных в PHP-сценариях.

5.5.2. Основные средства PHP для взаимодействия с базой данных

При взаимодействии сценария с базой данных обычно выполняются следующие действия:

1. Подключение к базе данных.
2. Передача СУБД SQL-запроса.
3. Получение от СУБД данных, соответствующих переданному ранее SQL-запросу.
4. Обработка полученных данных.
5. Отключение от базы данных, если не предполагается повторно выполнить этапы 2—4.

Подключение к базе данных

Подключение к базе данных (БД) осуществляется с помощью специальных функций PHP, которым передаются в качестве параметров следующие данные:

- местоположение (*host*) — для многих серверов БД это доменное имя или IP-адрес компьютера, на котором расположена БД; если она находится на том же компьютере, что и модуль PHP, то достаточно указать localhost. Для MS SQL Server указывается его имя;
- регистрационное имя (*username*) — строка символов, соответствующая имени пользователя, под которым он может получить доступ к БД; сообщается администратором БД;
- пароль (*password*) — строка символов, которая вместе с регистрационным именем служит для обеспечения доступа к БД; сообщается администратором БД;
- имя базы данных. СУБД может поддерживать несколько БД, каждая из которых должна иметь уникальное имя.

При подключении к базам данных под управлением разных СУБД в PHP используются различные функции. Ниже приведены примеры подключения к базе данных для трех популярных СУБД.

- MySQL 4.0:

```
$connect=mysqli_connect(host, username, password);
$db=mysqli_select_db($connect, имя_БД);
```

- MS SQL Server:

```
$connect=mssql_connect(имя_сервера_БД, username, password);
$db=mssql_select_db(имя_БД, $connect);
```

- PostgreeSQL:

```
$connect=pg_connect("host=host user=username password=password
                     dbname=имя_БД");
```

Для подключения к БД MySQL и MS SQL Server требуются две функции. Первая из них выполняет соединение с сервером БД и возвращает идентификатор соединения при успехе и `false` при неудаче. При этом для MySQL в первом параметре этой функции указывается *host* (доменное имя или IP-адрес компьютера), а для MS SQL Server — имя сервера БД, устанавливаемое при регистрации сервера. Вторая функция выбирает конкретную базу данных на сервере, на который указывает идентификатор `$connect`, возвращенный первой функцией. Эта функция возвращает `true` при успешном завершении операции и `false` — в противном случае.

Подключение к БД PostgreeSQL обеспечивается одной функцией, параметр которой содержит строку со всеми необходимыми данными. В этой строке `host=`, `user=`, `password=` и `dbname=` являются ключевыми словами.

Для отключения от базы данных используется функция, которой передается идентификатор соответствующего подключения. Например, `mysql_close($connect)` — для MySQL, `mssql_close($connect)` — для MS SQL Server, `pg_close($connect)` — для PostgreeSQL.

Подробные сведения о функциях работы с базами данных, как рассмотренных в этой книге, так и других, можно найти в справочном руководстве по PHP.

Передача запросов к базе данных

Запрос к базе данных формулируется на языке SQL и в качестве строкового параметра передается специальной функции PHP, которая в свою очередь передает запрос СУБД. Эта функция возвращает указатель на временную таблицу данных, являющуюся результатом выполнения запроса. Для различных СУБД функции передачи запроса похожи, но все же имеют некоторые отличия.

- MySQL 4.0:

```
mysql_query(SQL_запрос, идентификатор_соединения);
```

- MS SQL Server:

```
mssql_query(SQL_запрос, идентификатор_соединения);
```

- PostgreeSQL:

  pg_query(*идентификатор_соединения*, *SQL_запрос*);

Пример:

```
$connect=mssql_connect("MYDBSERVER", "guest"); /* подключение к серверу
                                                  БД */
$db=mssql_select_db("gbook", $connect);      // выбор БД
$strsql="SELECT * FROM gb";                  // строка с SQL-запросом
$selectall=mssql_query($strsql, $connect);   // выполнение SQL-запроса
$mssql_close($connect);                      // отключение от БД
```

Здесь производится подключение к MS SQL Server с именем `MYDBSERVER` для пользователя с регистрационным именем `guest` без пароля. Затем выбирается база данных `gbook` и выполняется SQL-запрос на выборку из таблицы `gb` всех полей (столбцов) и записей (строк). Наконец, соединение с БД разрывается.

Обработка данных в сценарии

Выбор нужных данных из одной или нескольких таблиц выполняется с помощью SQL-запроса, начинающегося с ключевого слова (оператора) `SELECT`. Критерий (условие) выбора данных формулируется в выражении SQL-запроса вслед за ключевым словом `WHERE`. Результатом выполнения такого запроса является временная таблица (возможно, пустая, если нет данных, удовлетворяющих условию запроса). Функция PHP, выполняющая запрос, возвращает указатель на полученную в результате временную таблицу. Этот указатель используется в PHP-сценарии для доступа к данным с помощью специальных функций.

Довольно часто данные из временной таблицы читают в массив, а затем его обрабатывают в сценарии (например, выводят на экран, изменяют значения, производят вычисления и т. п.). Если требуется изменить значения полей таблицы, добавить новую или удалить имеющуюся запись, то формируют строку с соответствующим SQL-запросом, а затем передают ее в качестве параметра функции, выполняющей запрос (например, `mssql_query()`).

Чтение в массив записей временной таблицы, полученной в результате выполнения SQL-запроса, производится функциями, в имени которой находится слово fetch (достать). Например, функция `mssql_fetch_assok()` читает текущую запись таблицы БД MS SQL Server в массив с символьными индексами (ключами). При этом значения индексов совпадают с именами соответствующих полей таблицы. Функция `mssql_fetch_row()` создает аналогичный массив, но с числовыми индексами, значения которых совпадают с номерами полей таблицы.

5.5.3. Гостевая книга

В качестве иллюстрации применения функций работы с базами данных рассмотрим создание гостевой книги. Проект гостевой книги в общих чертах обсуждался в *разд. 5.4.4*. Его особенность состояла лишь в том, что данные сохранялись не в базе данных под управлением СУБД, а в текстовом файле. Теперь мы будем использовать СУБД, которая должна быть предварительно установлена. Здесь описывается использование MS SQL Server 2000. Применение других серверов БД более или менее аналогично.

Создание гостевой книги на основе базы данных состоит из двух основных этапов:

- создание БД для хранения сообщений посетителей. Для гостевой книги достаточно одной таблицы в этой БД;
- разработка серверных сценариев, которые должны осуществлять подключение к БД, выводить ее содержимое для просмотра, предоставлять форму для ввода новых записей, а также добавлять эти записи в БД.

Администрирование MS SQL Server, создание базы данных, ее таблиц, а также заполнение таблиц данными производится с помощью специальной утилиты Enterprise Manager, которая имеет удобный пользовательский интерфейс.

Создание базы данных

Для создания базы данных необходимо запустить Enterprise Manager (рис. 5.11). В древовидном списке в левой части окна этой утилиты раскройте группу узлов с именем сервера БД: **Microsoft SQL Servers | SQL Server Group |** ***Имя_сервера*** **(Windows NT)**. В рассматриваемом примере сервер БД называется **VADIM**. Щелкните правой кнопкой мыши на узле **Databases** и в раскрывшемся контекстном меню выберите **New Database** (Новая база данных). В открывшемся диалоговом окне введите имя БД, например **Гостевая книга**, и щелкните на кнопке **OK**. В результате это имя появится в древовидном списке в группе **Databases**.

Теперь необходимо в БД **Гостевая книга** создать таблицу, в которой будут храниться данные. Для этого следует щелкнуть правой кнопкой мыши на имени БД в древовидном списке и в контекстном меню выбрать **Создать**, а затем — **Table**. В результате откроется диалоговое окно **New Table in '*имя БД*' on '*Имя сервера БД*'** (рис. 5.12), в котором можно определить поля (столбцы) таблицы, т. е. задать их имена, типы, размеры и другие характеристики. Помимо полей, которые предназначены для хранения данных, зададим поле, значения которого присваиваются автоматически и являются уникальными идентификаторами записей. Это — поле с именем `recid`, а его характеристики показаны на рис. 5.12.

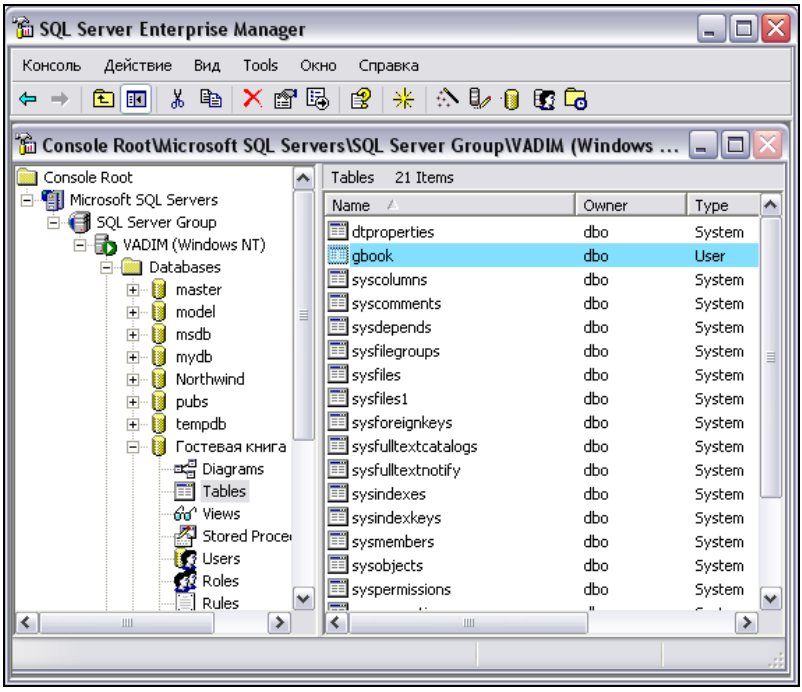


Рис. 5.11. Окно **SQL Server Enterprise Manager**

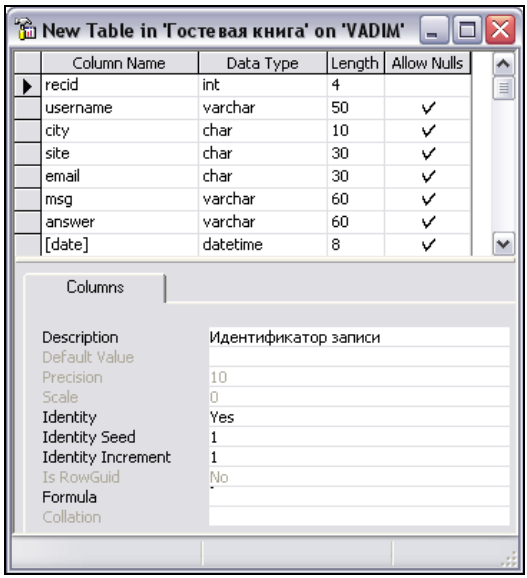


Рис. 5.12. Окно для определения полей таблицы

После закрытия окна **New Table...** предлагается сохранить таблицу под требуемым именем (например, `gbook`). Вручную вводить данные в таблицу не нужно, поскольку это будут делать посетители гостевой книги.

Определение регистрационного имени и пароля пользователя

Теперь следует определить пользователя (регистрационное имя и пароль), а также его права по работе с таблицей. Сначала определим пользователя. Для этого в древовидном списке окна **SQL Server Enterprise Manager** щелкните правой кнопкой мыши на узле **Users** (Пользователи) и в контекстном меню выберите **New Database User** (Новый пользователь БД). В результате откроется диалоговое окно **Database User Properties — New User** (Свойства пользователя БД — новый пользователь). В раскрывающемся списке **Login name** (Регистрационное имя) выберите **<new>**. Откроется окно **SQL Server Login Properties** (Свойства регистрации), в котором в поле **Name** введите какое-нибудь значение, например, **Гость**; установите переключатель **SQL Server Authentication** (Идентификация сервером), а в поле **Password** введите значение пароля или оставьте его пустым, если пароль не нужен. Для посетителей гостевой книги пароль можно не вводить. Щелкните на кнопке **OK**, чтобы сохранить настройки и вернуться к окну **Database User Properties — New User**. В этом окне в раскрывающемся списке **Login name** выберите имя, которое было создано ранее (например, **Гость**), и щелкните на кнопке **OK**. В результате имя пользователя появится в списке пользователей базы данных **Гостевая книга**.

Аналогичным образом определяются регистрационное имя и пароль владельца гостевой книги. В данном случае пароль необходим.

Определение прав пользователя

Далее необходимо определить права созданного пользователя БД. Посетителю гостевой книги не стоит предоставлять все возможные права по манипуляции базой данных.

В окне **SQL Server Enterprise Manager** найдите в списке пользователей имя **Гость**, щелкните на нем правой кнопкой мыши и выберите **Все задачи | Manage Permissions** (Управление разрешениями). В раскрывшемся окне **Database User Properties — Гостевая книга** (рис. 5.13) в строке, значение поля **Object** которой равно **gbook** (имя таблицы), установите флажки в полях **SELECT** и **INSERT**. Тем самым пользователю предоставляется право выполнять указанные SQL-операторы, обеспечивающие выборку и добавление данных соответственно. Это — типичные права посетителя гостевой книги.

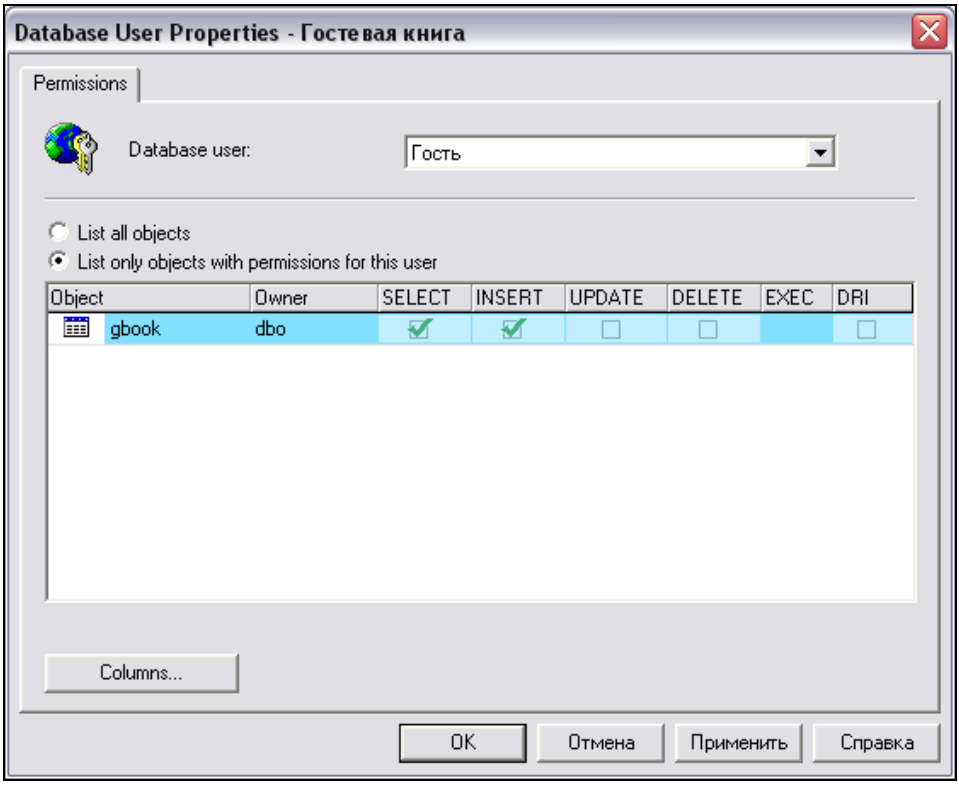


Рис. 5.13. Окно установки прав пользователя по работе с данными

Аналогичным образом определяются права владельца гостевой книги. Кроме прав гостя (выборка и добавление данных), он должен иметь возможность изменять (**UPDATE**) и удалять (**DELETE**) записи.

Сценарии для взаимодействия с посетителем

Для взаимодействия гостевой книги с посетителем потребуются два сценария. Первый сценарий формирует страницу, содержащую все записи из базы данных гостевой книги. Кроме того, на этой странице находится кнопка **Добавить запись**, щелчок на которой приводит к переходу ко второму сценарию, предоставляющему форму для ввода данных и выполняющему запись их в БД. Страницы, созданные этими сценариями, выглядят в окне браузера так же, как на рис. 5.9 и 5.10. Пусть эти сценарии сохранены в файлах gbookdb.php и gbadd_db.php.

В листинге 5.42 представлен код сценария из файла gbookdb.php.

Листинг 5.42. Вывод содержимого гостевой книги (gbookdb.php)

```
<?php
/* Форма с кнопкой  для перехода к странице добавления записи */
echo "<center><h2>Моя гостевая книга</h2>";
echo "<form action=gbadd_db.php method=POST>
        <input type=submit name=addrecord value='Добавить запись'>
      </form><br>";
/**** Подключение к БД ***********************************************/
$servername="VADIM";         // имя сервера БД
$dbname="[Гостевая книга]";  // имя БД
$user='Гость';               // login пользователя
$connect=mssql_connect($servername,$user);  // соединение с сервером
$db=mssql_select_db($dbname, $connect);     // выбор БД
/**** Выбор всех записей *********************************************/
$strsql="SELECT * FROM gbook ORDER BY date DESC";
$rez=mssql_query($strsql,$connect);   // выполнение запроса
/**** Вывод содержимого БД *******************************************/
while ($row=mssql_fetch_assoc($rez)){ // цикл по всем записям таблицы
  $i++;
  $t=$row['date'];
  echo "<b>$i.  {$row['username']}</b>  {$row['city']}  $t<br>
        <b>Сайт:</b> <a href={$row['site']}>{$row['site']}</a>
        <b> E-mail:</b> <a href={$row['email']}>{$row['email']}</a><br>
        <br><textarea rows=4 cols=60>{$row['msg']}</textarea><br>";
  if ($row['answer']) {
    echo "<b>Ответ:</b><br>
          <textarea rows=4 cols=60>{$row['answer']}</textarea>";
  }
  echo "<hr><br>";
}
mssql_close($connect); // отключение от БД
?>
```

Обратите внимание, что если имя базы данных содержит пробелы, дефисы и некоторые другие специфические символы, то его следует заключить в квадратные скобки. Например, `$dbname="[Гостевая книга]";`.

SQL-запрос в данном сценарии возвращает все записи и все поля таблицы `gbook`, упорядоченные по убыванию значений поля `date` (дата и время). Таким образом, новые записи располагаются в таблице выше старых.

В листинге 5.43 представлен код сценария из файла gbadd_db.php.

Листинг 5.43. Отображение формы ввода и добавление данных в гостевую книгу (gbadd_db.php)

```
<?php
if ($_POST['addrecord']){  // если требуется добавить запись
  /* Вывод формы для ввода сообщения */
  echo "<html>
          <center><h2>Моя гостевая книга</h2></center>
          <form action=gbadd_db.php method=POST>
            <table>
              <tr><td>Ваше имя:</td>
                <td><input type=text name=username size=20></td></tr>
              <tr><td>Город:</td>
                <td><input type=text name=city size=20></td></tr>
              <tr><td>URL Вашего сайта:</td>
                <td> <input type=text name=site size=40></td></tr>
              <tr><td>E-mail:</td>
                <td><input type=text name=email size=40></td></tr>
            </table>
            Сообщение:<br>
            <textarea  name=msg rows=4 cols=60></textarea><br>
            <input type=submit name=addok value='Добавить запись'>
            <input type=reset name=reset value='Отмена'>
          </form><br>
        </html>";
} elseif ($_POST['addok']) {      // если подтвердили ввод в форму
  header("Location:gbookdb.php"); /* Переход к странице показа
                                     гостевой книги */
  if (!$_POST['msg']){exit;};     // если сообщение пусто, то выход
  $t=date("Y-m-d H:i:s");         // текущие дата и время
  /**** Подключение к БД ************************************/
  $servername="VADIM";            // имя сервера БД
  $dbname="[Гостевая книга]";     // имя БД
```

```
  $user='Гость';                    // login пользователя
  $connect=mssql_connect($servername,$user);  // соединение с сервером
  $db=mssql_select_db($dbname, $connect);     // выбор БД
  $strsql="INSERT INTO gbook
            (username, city, site, email, msg, date)
          VALUES (\"{$_POST['username']}\", \"{$_POST['city']}\",
                  \"{$_POST['site']}\",\"{$_POST['email']}\",
                  \"{$_POST['msg']}\",
          CAST('".$t."' AS datetime))";
  $rez=mssql_query($strsql,$connect);  // выполнение запроса
  mssql_close($connect);               // отключение от БД
}
?>
```

SQL-запрос в данном сценарии содержит SQL-функцию `CAST()` для преобразования типов данных. В нашем случае необходимо привести строковое значение даты-времени `$t` к типу `datetime` поля `date` таблицы БД.

Определение параметров подключения к БД можно разместить в отдельном файле и включить его в оба рассмотренных сценария с помощью функции `include()`.

В данном примере мы не предусмотрели обработку возможных ошибок подключения и выполнения запроса к БД. В реальном проекте это следует сделать.

Сценарии для владельца гостевой книги

Для владельца гостевой книги необходимо создать отдельную, защищенную паролем страницу, а в сценариях ее обслуживания — установить другие параметры соединения с БД (регистрационное имя и пароль). Владелец книги обладает дополнительными правами на изменение и удаление записей.

Страница для владельца книги внешне может быть аналогична странице для посетителя, но есть и отличия. На мой взгляд проще всего каждую выводимую запись гостевой книги заключить в теги формы с кнопками **Принять изменения** и **Удалить** типа `submit`. Кроме того, в скрытое поле формы поместить значение поля БД `recid` (идентификатор записи). Тогда в сценарии обработки щелчков на этих кнопках полученное значение скрытого поля можно использовать в условии SQL-запроса к БД. Например, пусть кнопка **Удалить** имеет имя (значение атрибута `name`) `delete`, а скрытое поле со зна-

чением поля БД `recid` — `recid` (т. е. такое же, что и поле БД). Тогда для удаления записи может быть применен следующий фрагмент PHP-кода:

```
if ($_POST['delete']){
  $strsql="DELETE FROM gbookdb WHERE recid={$_POST['recid']}";
  mssql_query($strsql,$connect);
}
```

Предположим, кнопка **Принять изменения** имеет имя (значение атрибута `name`) `update`, а поле формы для ввода ответа — `answer` (т. е. такое же, что и поле БД). Допустим, вы хотите сохранить только введенный вами ответ. Тогда можно воспользоваться следующим фрагментом PHP-кода:

```
if ($_POST['update']){
  $strsql="UPDATE gbookdb SET answer=\"{$_POST['recid']}\"
           WHERE recid={$_POST['recid']}";
  mssql_query($strsql,$connect);
}
```

Предлагаю вам самостоятельно довести проект гостевой книги "до ума".

Приложения

Приложение 1. Основные объекты браузера и документа

Приложение 2. Установка модуля PHP

Приложение 3. Установка Web-сервера IIS в Windows 2000/XP

Приложение 1

Основные объекты браузера и документа

Здесь мы рассмотрим основные объекты браузера Internet Explorer и загруженного в него документа. Эти объекты имеют иерархическую структуру, показанную на рис. П1.1. Однако следует иметь в виду, что для других браузеров эта структура может быть несколько иной.

П1.1. Объект *window*

Объект `window` (окно) содержит коллекцию `frames` всех фреймов, заданных в HTML-документе с помощью контейнерного тега `<FRAMESET>`. Объект `window` имеет свойства, методы, события, а также дочерние объекты. Приведем их полные перечни и рассмотрим, с разной степенью подробности, только наиболее важные с практической точки зрения.

П1.1.1. Свойства объекта *window*

Список свойств объекта `window` таков:

- `parent` — возвращает родительское окно для текущего окна;
- `self` — возвращает ссылку на текущее окно;
- `top` — возвращает ссылку на главное окно;
- `name` — название окна;
- `opener` — окно, создаваемое текущим;
- `closed` — сообщает, если окно закрыто;
- `status` — текст, отображаемый в строке состояния браузера;
- `defaultStatus` — текст по умолчанию строки состояния браузера;
- `returnValue` — позволяет определить возвращенное значение для события или диалогового окна;

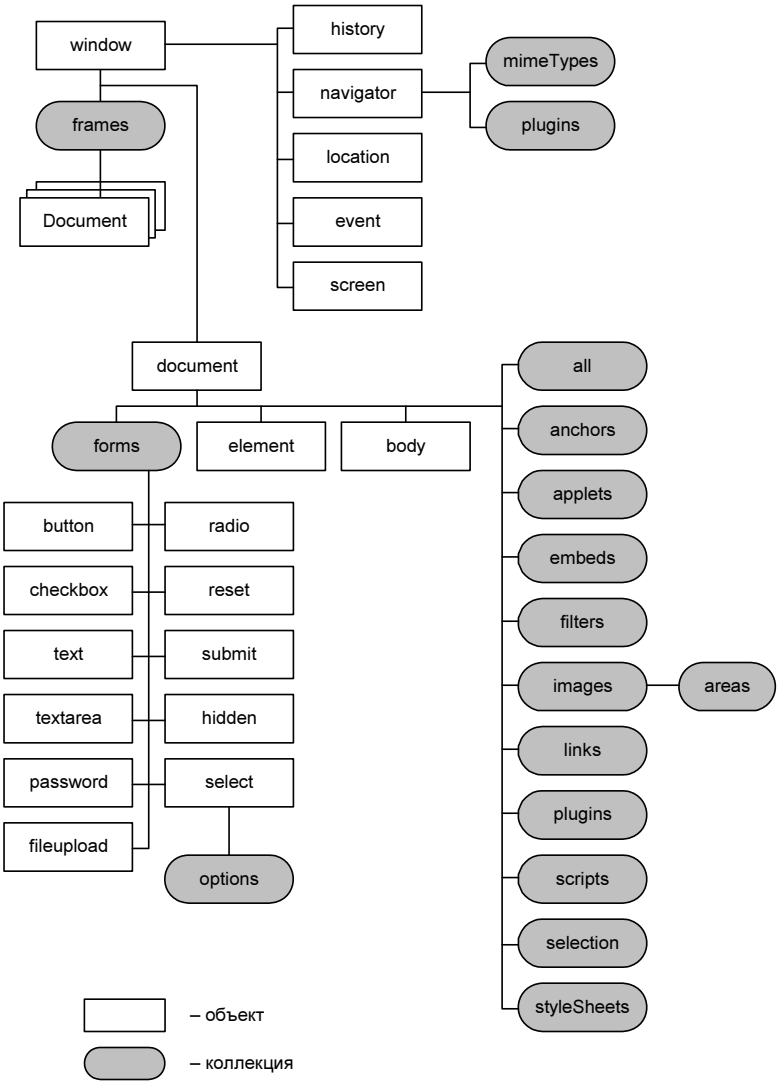


Рис. П1.1. Иерархия объектов

- `client` — ссылка, которая возвращает объект навигатора браузеру;
- `document` — ссылка только для чтения на объект окна `document`;
- `event` — ссылка только для чтения на глобальный объект `event`;
- `history` — ссылка только для чтения на объект окна `history`;
- `location` — ссылка только для чтения на объект окна `location`;

- `navigator` — ссылка только для чтения на объект окна `navigator`;
- `screen` — ссылка только для чтения на глобальный объект `screen`.

Свойство `parent` позволяет обратиться к объекту, находящемуся в иерархии на одну ступень выше, например, к окну, содержащему коллекцию фреймов, в которой находится наш фрейм. Для перемещения на две ступени вверх мы должны использовать `parent.parent` и т. д.:

```
parent.window.frames(0)
parent.parent.window.frames(0)
```

Чтобы обратиться к самому главному окну, т. е. к окну браузера, следует использовать свойство `top`. Однако `top` не может обращаться к главному фрейму вашей системы разбиения на фреймы.

Свойство `name` соответствует имени фрейма, которое мы задаем в теге `<FRAMESET>`.

Свойство `status` полезно использовать для вывода сообщений во время работы сценария, например, при отладке:

```
window.status="Сейчас работает сценарий"
```

П1.1.2. Методы объекта *window*

Перечислим методы объекта `window`:

- `open()` — открывает новое окно браузера;
- `close()` — закрывает текущее окно браузера;
- `showHelp()` — показывает окно подсказки как диалоговое;
- `showModalDialog()` — показывает новое окно как диалоговое (модальное);
- `alert()` — показывает окно предупреждения с сообщением и кнопкой **OK**;
- `prompt()` — показывает окно приглашения с сообщением, текстовым полем и кнопками **OK** и **Отмена** (Cancel);
- `confirm()`— показывает окно подтверждения с сообщением и кнопками **OK** и **Отмена** (Cancel);
- `navigate()` — загружает другую страницу с указанным адресом;
- `blur()` — убирает фокус с текущей страницы; соответствующее событие — `onblur`;
- `focus ()` — устанавливает страницу в фокус; соответствующее событие — `onfocus`;
- `scroll()` — разворачивает окно на заданные ширину и высоту;

- `setInterval()` — указывает процедуре выполняться периодически через заданное количество миллисекунд;
- `setTimeout()` — запускает программу через заданное количество миллисекунд после загрузки страницы;
- `clearInterval()` — обнуляет таймер, заданный методом `setInterval()`;
- `clearTimeout()` — обнуляет таймер, заданный методом `setTimeout()`;
- `execScript()` — выполняет код сценария; по умолчанию JScript.

П1.1.3. События объекта *window*

Далее перечислены события объекта `window`:

- `onblur` — выход окна из фокуса;
- `onfocus` — окно становится активным;
- `onhelp` — нажатие пользователем клавиши <F1>;
- `onresize` — изменение пользователем размеров окна;
- `onscroll` — прокрутка окна пользователем;
- `onerror` — ошибка при передаче;
- `onbeforeunload` — возникает перед выгрузкой страницы, что позволяет сохранить данные;
- `onload` — страница полностью загружена;
- `onunload` — происходит непосредственно перед выгрузкой страницы.

Три из перечисленных выше событий происходят в результате действий пользователя. Если открыто несколько окон браузера, пользователь может переключаться между ними, переводя фокус с одного окна на другое. Эти действия инициируют события `onblur` и `onfocus`. Заметим, что эти же события можно вызвать программным способом, используя методы `blur` и `focus`. Если возникает ошибка при загрузке страницы или ее элемента, то инициируется событие `onerror`. Мы можем использовать это событие в программе, чтобы, например, попытаться еще раз загрузить страницу или как-то изменить дальнейшие действия. Вот пример:

```
<SCRIPT>
  function window.onerror(){
    alert("Произошла ошибка! Попробуйте еще раз.")
  }
</SCRIPT>
```

Событие `onload` происходит, когда страница полностью загружена в окно; событие `onbeforunload` — перед тем как страница покинет окно; событие `onunload` — когда страница выгружена, перед загрузкой новой страницы или перед закрытием браузера. Например:

```
<SCRIPT>
  function window.onunload(){
    alert("Страница выгружается!")
  }
</SCRIPT
```

Объект `window` имеет несколько дочерних объектов, которые доступны с его помощью: `document`, `history`, `navigator`, `location`, `event` и `screen`.

П1.2. Объект *document*

Объект `document` является центральным в иерархической объектной модели и представляет всю информацию об HTML-документе с помощью коллекций и свойств. Он также предоставляет множество методов и событий для работы с документами. Поскольку мы уже рассматривали некоторые основные приемы обращения с этим объектом, то здесь ограничимся лишь справочными сведениями (табл. П1.1).

Таблица П1.1. *Свойства объекта* `document`

Свойство	**Атрибут**	**Назначение**
`activeElement`		Идентифицирует активный элемент
`alinkColor`	`ALINK`	Цвет активных ссылок на странице
`bgColor`	`BGCOLOR`	Цвет фона элемента
`body`		Ссылка только для чтения на неявный основной объект документа, определенный в теге `<BODY>`
`cookie`		Строка cookie-записи. Присвоение нового значение этому свойству приводит к записи cookie на диск после закрытия браузера
`domain`		Устанавливает или возвращает домен документа для его защиты или идентификации
`bgColor`	`TEXT`	Устанавливает цвет текста переднего плана
`lastModified`		Дата последнего изменения страницы, доступна как строка

Таблица П1.1 *(окончание)*

Свойство	Атрибут	Назначение
linkColor	LINK	Цвет еще не посещенных гиперссылок на странице
location		Полный URL документа
parentWindow		Возвращает родительское окно для документа
readyState		Определяет текущее состояние загружаемого объекта
referer		URL страницы, которая вызвала текущую
selection		Ссылка только для чтения на дочерний для document объект selection
title	TITLE	Определяет справочную информацию для элемента, используемую при загрузке или во всплывающей подсказке
url	URL	URL-адрес документа клиента или в теге <META>
vlinkColor	VLINK	Цвет посещенных ссылок на странице

П1.2.1. Коллекции объекта *document*

Перечислим коллекции объекта document:

- all — коллекция всех тегов и элементов в основной части документа;
- anchors — коллекция всех "якорей" (закладок) в документе;
- applets — коллекция всех объектов в документе, включая встроенные элементы управления, графические элементы, апплеты, внедренные и другие объекты;
- embeds — коллекция всех внедренных объектов в документе;
- forms — коллекция всех форм на странице;
- frames — коллекция всех фреймов, определенных в теге <FRAMESET>;
- images — коллекция всех графических элементов (изображений) на странице;
- links — коллекция всех ссылок и блоков <AREA> на странице;
- plugins — еще одно название для коллекции внедренных объектов документа;

- `scripts` — коллекция всех разделов `<SCRIPT>` на странице;
- `styleSheets` — коллекция всех конкретных свойств стиля, определенных в документе.

П1.2.2. Методы объекта *document*

Приведем список методов объекта `document`:

- `clear` — очищает выделенный участок;
- `close` — закрывает текущее окно браузера;
- `createElement` — создает экземпляр элемента для выделенного тега;
- `elementFromPoint` — возвращает элемент с заданными координатами;
- `execCommand` — выполняет команду (операцию) над выделением или областью;
- `open` — открывает документ как поток для обработки результатов применения методов `write` и `writeln`;
- `queryCommandEnabled` — сообщает, доступна ли данная команда;
- `queryCommandIndeterm` — сообщает, если данная команда имеет неопределенный статус;
- `queryCommandState` — возвращает текущее состояние команды;
- `queryCommandSupported` — сообщает, поддерживается ли данная команда;
- `queryCommandText` — возвращает строку, с которой работает команда;
- `queryCommandValue` — возвращает значение команды, определенное для документа или объекта `TextRange`;
- `write` — записывает текст и код HTML в документ, находящийся в указанном окне;
- `writeln` — записывает текст и код HTML, заканчивающийся возвратом каретки.

П1.2.3. События объекта *document*

Перечислим события объекта `document`:

- `onafterupdate` — возникает при окончании передачи данных;
- `onbeforeupdate` — возникает перед выгрузкой страницы;
- `onclick` — происходит при щелчке левой кнопкой мыши;
- `ondblclick` — происходит при двойном щелчке левой кнопкой мыши;

- `ondragstart` — происходит, когда пользователь начинает перетаскивание;
- `onerror` — ошибка при передаче;
- `onhelp` — нажатие пользователем клавиши <F1>;
- `onkeydown` — возникает при нажатии клавиши;
- `onkeypress` — возникает при нажатии клавиши и продолжается при удерживании клавиши в нажатом состоянии;
- `onkeyup` — возникает, когда пользователь отпускает клавишу;
- `onload` — возникает при полной загрузке документа;
- `onmousedown` — происходит при нажатии кнопки мыши;
- `omousemove` — происходит при перемещении указателя мыши;
- `onmouseout` — происходит, когда указатель мыши выходит за границы элемента;
- `onmouseover` — происходит, когда указатель мыши входит на элемент;
- `onmouseup` — происходит, когда пользователь отпускает кнопку мыши;
- `onreadystatechange` — возникает при изменении свойства `readystate`;
- `onselectstart` — происходит, когда пользователь в первый раз запускает выделенную часть документа.

П1.3. Объект *location*

Объект `location` (расположение) содержит информацию об URL-адресе (и его компонентах) текущей страницы, а также методы, позволяющие обновлять страницы.

П1.3.1. Свойства объекта *location*

Далее приведен список свойств объекта `location`:

- `href` — полный URL-адрес в виде строки;
- `hash` — строка, следующая в URL за символом `#`, с помощью которого указывается, к какому якорю внутренней ссылки следует переместиться при загрузке документа;
- `host` — часть URL "*хост*:*порт*"; значение порта содержится лишь тогда, когда оно явно было указано в URL-адресе;
- `hostname` — часть URL "*хост*";
- `pathname` — путь к объекту или файлу, указанный после третьего слэша в URL-адресе;

- `port` — номер порта URL;
- `protocol` — начальная часть, определяющая протокол, за которой следует двоеточие, например, `"http:"`;
- `search` — строка запроса или данные URL после знака `?`.

Например, если вы загрузили страницу с адресом **http://www.cityline.spb.ru**, то значением `location.href` будет эта строка.

Присваивая свойству `href` новое значение, мы можем изменять показываемую в браузере страницу, например:

```
window.location.href="http://www.rambler.ru"
```

П1.3.2. Методы объекта *location*

Перечислим методы объекта `location`:

- `assign()` — загружает другую страницу; этот метод эквивалентен изменению свойства `window.location.href`;
- `reload()` — обновляет текущую страницу;
- `replace()` — загружает страницу с указанным в параметре URL-адресом и заменяет URL-адрес текущей страницы (эквивалентен `location.href`).

Некоторые Web-сайты могут содержать страницы, которые не предназначены для посещения пользователем и занесения их в список посещенных страниц. Например, перемещение по сайту может привести пользователя на некоторые промежуточные страницы, которые ему больше не понадобятся. При этом желательно, чтобы пользователь не мог вернуться к ним снова с помощью кнопки **Назад** (Back). В этом случае применяется метод `location.replace(`*`URL-адрес`*`)` для перехода к указанной странице без занесения его в список документов, переход к которым осуществляется щелчком на этой кнопке.

В следующем примере показано, как обеспечить автоматический переход к той или иной странице в зависимости от браузера посетителя.

```
if (navigator.appName=="Microsoft Internet Explorer") loca-
tion.replace("page1.htm")
else location.replace("page2.htm");
```

П1.4. Объект *history*

Объект `history` (история) содержит информацию об адресах страниц, которые браузер отобразил во время текущего сеанса. Вы можете передвигаться

по этому списку с помощью сценария и загружать соответствующие страницы. Объект `history` имеет только одно свойство и три метода.

П1.4.1. Свойство объекта *history*

`length` — количество элементов в списке посещенных страниц.

Это свойство используется в сценарии для контроля выхода за пределы списка.

П1.4.2. Методы объекта *history*

Методы таковы:

- `back()` — загружает предыдущую страницу из списка;
- `forward()` — загружает следующую страницу из списка;
- `go()` — загружает страницу с относительным номером `n` из списка (от 0 по `history.length` – 1) или с указанным URL-адресом.

В следующем примере загружается первая страница, а затем — пятая страница, причем предварительно проверяется, существует ли она в списке уже посещенных страниц:

```
window.history.go(1)
if (window.history.length>4)
      window.history.go(5)
```

Очевидно, метод `history.go(-1)` эквивалентен `history.back()`, а метод `window.history.go(1)` эквивалентен методу `history.forward()`.

П1.5. Объект *navigator*

Объект `navigator` (навигатор) содержит информацию о производителе браузера, его версии и возможностях.

П1.5.1. Свойства объекта *navigator*

Далее перечислены свойства объекта `navigator`:

- `appCodeName` — название кода браузера; например, "Mozilla";
- `appName` — название браузера; например, "Microsoft Internet Explorer";
- `appVersion` — версия браузера; например, `"4.0 (compatible; MSIE 6.0; Windows 98; Win 9x 4.90)"`;

- `cookieEnabled` — определяет возможность использования в браузере cookies со стороны клиента; является логическим значением;
- `userAgent` — название браузера, посылаемое с помощью HTTP-протокола; например, `"Mozilla/4.0 (compatible; MSIE 6.0; Windows 98; Win 9x 4.90)"`.

П1.5.2. Коллекции объекта *navigator*

Перечислим коллекции объекта `navigator`:

- `mimeTypes` — коллекция всех типов документов и файлов, поддерживаемых браузером;
- `plugins` — коллекция всех внедряемых объектов на странице.

П1.5.3. Методы объекта *navigator*

Методы объекта `navigator` таковы:

- `taintEnabled` — возвращает значение `false`; метод включен для совместимости с Netscape Navigator;
- `javaEnabled` — сообщает, возможен ли в данном браузере запуск кода сценария на языке JavaScript; является логическим значением.

Иногда требуется установить, каков браузер пользователя, чтобы выполнить переход к следующему HTML-документу. Например, мой сайт с учебными примерами (**www.admiral.ru/~dunaev**) изобилует элементами, которые правильно отображаются только в Internet Explorer. Чтобы у пользователя не возникало недоразумений, я проверяю, каким браузером он пользуется. В зависимости от результата этой проверки происходит переход на соответствующую страницу. Это можно сделать с помощью следующего сценария:

```
if (navigator.appName=="Microsoft Internet Explorer")
  location.replace("mypage1.htm")
else location.replace("mypage2.htm")
```

П1.6. Объект *event*

Объект `event` (событие) позволяет получить информацию о каком-либо событии, происходящем в браузере. Эта информация содержится в следующих свойствах:

- `altKey` — возвращает состояние клавиши <Alt>, когда происходит событие;

- `button` — определяется кнопка мыши, вызывающая событие;
- `cancelBubble` — устанавливается для запрета прохождения заданного события вверх по объектной иерархии;
- `clientX` — возвращает координату *x* элемента, исключая обрамление, отступы, полосы прокрутки и т. д.;
- `clientY` — возвращает координату *y* элемента, исключая обрамление, отступы, полосы прокрутки и т. д.;
- `CtrlKey` — состояние клавиши <Ctrl> при появлении события;
- `fromElement` — возвращает элемент, с которого ушел курсор мыши, для событий `onmouseover` и `onmouseout`;
- `keyCode` — код ASCII нажатой клавиши; есть возможность изменять значение, передаваемое объекту;
- `offsetX` — возвращает координату *x* указателя мыши в пикселах относительно содержащего его элемента при возникновении события;
- `offsetY` — возвращает координату *y* указателя мыши в пикселах относительно содержащего его элемента при возникновении события;
- `reason` — указывает, что перемещение данных прошло успешно или из-за чего оно прекратилось;
- `returnValue` — определяет возвращаемое значение для события;
- `screenX` — возвращает горизонтальную координату указателя мыши относительно экрана, когда происходит событие;
- `screenY` — возвращает вертикальную координату указателя мыши относительно экрана, когда происходит событие;
- `shiftKey` — определяет состояние клавиши <Shift> при возникновении события;
- `srcElement` — возвращает элемент, с которого началось прохождение события;
- `srcFilter` — возвращает фильтр, создавший событие `onfilterchange`;
- `toElement` — возвращает элемент, на который наводится курсор мыши, при появлении события `onmouseover` или `onmouseout`;
- `type` — возвращает название события как строку, без приставки "on";
- `x` — возвращает координату *x* указателя мыши соответственно либо к позиционированному родительскому элементу, либо к окну;

- `y` — возвращает координату *y* указателя мыши соответственно либо к позиционированному родительскому элементу, либо к окну.

Свойства объекта `event` устанавливаются в момент прохождения события и большинство из них доступны только для чтения (их нельзя изменить). Однако есть два изменяемых свойства: `keyCode` и `returnValue`. Мы уже рассматривали применение объекта `event`.

П1.7. Объект *screen*

Объект `screen` (экран) содержит информацию о возможностях экрана пользователя и может применяться, например, для определения размеров создаваемых окон, и выбора разрешения передаваемой графики (нет смысла загружать 32-битное изображение, если монитор и видеокарта пользователя поддерживают только 256 цветов). Объект `screen` имеет следующие свойства:

- `width` — возвращает ширину экрана пользователя (в пикселах);
- `height` — возвращает высоту экрана пользователя (в пикселах);
- `bufferDepth` — определяет, используется ли буфер для хранения "второго экрана";
- `colorDepth` — возвращает информацию, позволяющую решить, как использовать цвета на экране; количество битов на пиксел устройства или видеобуфера пользователя;
- `updateInterval` — возвращает или задает интервал времени между обновлениями экрана пользователя.

П1.8. Объект *TextRange*

Объект `TextRange` (текстовая область) отображает разделы потока текста, формирующего HTML-документ. Может использоваться для управления текстом внутри страницы, а также для организации поиска в тексте.

П1.8.1. Свойства объекта *TextRange*

Свойства объекта `TextRange` таковы:

- `htmlText` — возвращает содержимое `TextRange` как текст и код HTML;
- `text` — простой текст, находящийся внутри элемента `TextRange` или тега `<OPTION>`.

П1.8.2. Методы объекта *TextRange*

Перечислим методы объекта `TextRange`:

- `collapse` — стягивает текстовую область в точку в начале или конце текущей области;
- `compareEndPoints` — сравнивает две текстовые области и возвращает значение, показывающее результат;
- `duplicate` — возвращает копию области `TextRange`;
- `execCommand` — выполняет команду (операцию) над выделением или областью;
- `expand` — расширяет текстовую область, добавляя туда новый знак, слово, предложение, или указывает, какие неполные блоки полностью в ней содержатся;
- `findText` — определяет текстовую область, содержащую только искомый текст;
- `getBookmark` — возвращает значение, позволяющее в дальнейшем идентифицировать данную позицию в документе;
- `inRange` — определяет, находится ли заданная текстовая область внутри текущей;
- `isEqual` — определяет, равны ли заданная и текущая текстовые области;
- `move` — изменяет начальную и конечную точки текстовой области для включения в нее различного текста;
- `moveEnd` — заставляет текстовую область сжаться или расшириться до заданной конечной точки;
- `moveStart` — заставляет текстовую область сжаться или расшириться до заданной начальной точки;
- `moveToBookmark` — передвигает границы текстовой области для включения другой, определенной ранее с помощью `getBookmark`;
- `moveToElementText` — передвигает границы текстовой области для включения текста в заданном элементе;
- `moveToPoint` — передвигает границы текстовой области и сжимает ее вокруг выбранной точки;
- `parentElement` — возвращает элемент, родительский по отношению ко всему, что входит в текстовую область;
- `pasteHTML` — вставляет текст и/или код HTML в текущую текстовую область;

- `queryCommandEnabled` — сообщает, доступна ли данная команда;
- `queryCommandIndeterm` — сообщает, если данная команда имеет неопределенный статус;
- `queryCommandState` — возвращает текущее состояние команды;
- `queryCommandSupported` — сообщает, поддерживается ли данная команда;
- `queryCommandText` — возвращает строку, с которой работает команда;
- `queryCommandValue` — возвращает значение команды, определенное для документа или объекта `TextRange`;
- `scrollIntoiew` — переносит текущую текстовую область в видимую часть окна браузера;
- `select` — делает активный подсвеченный участок выделения на странице равным текущей текстовой области;
- `setEndPoint` — переносит начальную или конечную точки текущей текстовой области в начало или конец заданной области.

Приложение 2

Установка модуля PHP

Модуль PHP можно установить в системах UNIX/Linux, Mac и Windows. Здесь мы рассмотрим установку PHP в Windows NT/2000/XP. Хотя модуль PHP можно использовать и в Windows 98/ME, это не рекомендуется делать.

Существуют два варианта модуля PHP, каждому из которых соответствует отдельный исполняемый файл:

- PHP CGI — для поддержки Web-сайтов (сценариев, выполняемых Web-сервером);
- PHP CLI — для создания независимых сценариев, работающих вне связи с Web-сервером.

Возможна установка сразу двух вариантов.

П2.1. Где взять модуль PHP?

Файлы модуля PHP находятся на сайте **www.php.net**. Зайдите на этот сайт и перейдите в раздел **Downloads** (Загрузки), а затем — в раздел **Windows Binaries**. В этом разделе щелкните на одной из двух ссылок, соответствующих последней версии PHP:

- **PHP *версия* zip package** — полный пакет PHP (CGI и CLI);
- **PHP *версия* installer** — программа автоматической установки PHP CGI.

Программа автоматической установки PHP CGI имеет графический интерфейс, позволяющий выбирать параметры установки. Здесь она не будет описана. Рекомендую установить полный пакет, но тогда модуль PHP придется устанавливать вручную. Это сделать несложно, а приобретенный опыт пригодится в дальнейшем.

К моменту завершения работы над книгой была доступной версия 5.2.0. Для выбора полного пакета PHP этой версии щелкните на ссылке **PHP 5.2.0 zip package.** В результате будет выведен список зеркальных сайтов. Для России выберите **ru.php.net**. Щелчок на этой ссылке приведет к открытию диалогового окна, в котором следует:

1. Выбрать режим сохранения файла.
2. Указать папку на вашем компьютере, где требуется сохранить zip-файл (архив исходных файлов PHP).
3. Щелкнуть на кнопке **Save** (Сохранить). Начнется перекачка файла в указанную папку. По окончании загрузки в этой папке появится файл с именем, например, php-5.2.0-win32.zip. Цифры 5.2.0 указывают номер версии PHP.

П2.2. Установка модуля PHP

П2.2.1. Размещение модуля PHP на диске компьютера

Итак, мы рассматриваем ручную установку полного пакета PHP. Теперь необходимо распаковать загруженный архивный файл в папку, в которой будет размещаться модуль PHP вместе с другими его файлами. Разархивируйте загруженный zip-файл, например, php-5.2.0-win32.zip в папку, в которой вы хотите установить модуль PHP, например, C:\php. Не рекомендуется устанавливать модуль PHP в папку, имя которой содержит пробелы, например, Programs Files. Папка, в которую вы разархивировали файлы PHP, называется корневой папкой PHP. Если вы разархивировали файлы в папку C:\php, то именно эта папка и будет корневой.

В корневой папке PHP (например, C:\php) появятся несколько папок (dev, ext, extras, PEAR), а также какие-то файлы. Модулю PHP CLI соответствует исполняемый файл php.exe, а модулю PHP CGI — php-cgi.exe.

Итак, модуль PHP с сопутствующими ему файлами размещен на диске компьютера. Теперь его необходимо настроить, т. е. установить параметры.

П2.2.2. Настройка модуля PHP

Параметры настройки модуля PHP находятся в текстовом конфигурационном файле php.ini.

В системе Windows в зависимости от ее версии этот файл должен находиться в папке:

- WINDOWS — в системах Windows 98/ME/XP;
- WINNT — в системах Windows NT/2000.

При использовании программы автоматической установки файл php.ini создается самостоятельно. При ручной установке необходимо исходный конфигурационный файл php.ini-dist из корневой папки модуля PHP скопировать под именем php.ini в нужную папку, в зависимости от версии Windows.

Если вы используете Web-сервер Internet Information Services (IIS), то необходимо изменить значение одного из параметров в файле php.ini. Для этого откройте файл php.ini в текстовом редакторе (например, Блокноте) и найдите в нем строку:

```
;cgi.force_redirect = 1
```

В начале этой строки удалите точку с запятой (оператор комментария) и замените значение 1 на 0.

В конфигурационном файле php.ini содержится много подобных строк, называемых *директивами* или *параметрами модуля PHP*. В дальнейшем, скорее всего, вам придется изменять значения директив.

П2.2.3. Установка расширений PHP

Интерпретатор PHP состоит из ядра (core), поддерживающего основную функциональность языка, и расширений (extensions). Некоторые расширения устанавливаются по умолчанию при установке PHP. Другие входят в дистрибутив, но их необходимо установить отдельно. Например, расширение, содержащее функции работы с графическими изображениями, СУБД, PDF-документами автоматически не устанавливаются. Чтобы узнать, какие расширения установлены, достаточно вызвать функцию `phpinfo()`. Эта функция выводит сначала параметры ядра PHP, а затем всех установленных расширений.

Вот некоторые из наиболее популярных расширений, которые устанавливаются вручную:

- расширения для работы с базами данных (СУБД) без использования интерфейса ODBC. Расширение для поддержки ODBC включено по умолчанию, а для каждой используемой СУБД требуется установить отдельное расширение (библиотеку функций).

Ниже приведен список из трех таких расширений:

- MySQL — php_mysql.dll;
- MS SQL Server — php_mssql.dll;
- PostgeeSQL — php_pgsql.dll;

❒ библиотека GD2 функций для работы с графическими изображениями — php_gd2.dll.

Файлы расширений находятся в папке ext, расположенной в корневой папке модуля PHP, например, C:\php\ext. Чтобы установить расширение, выполните следующие действия:

1. Скопируйте dll-файл расширения из папки ext в корневую папку PHP, например, C:\php.
2. Внесите изменения в конфигурационный файл php.ini. А именно найдите в нем строку вида:

   ```
   ;extension=файл_расширения
   ```

   Например, если вы устанавливаете расширение для работы с СУБД MS SQL Server, то требуемая строка в php.ini имеет вид:

   ```
   ;extension=php_mssql.dll
   ```

   В начале найденной строки удалите точку с запятой и сохраните изменения.

П2.3. Настройка Web-сервера IIS

Чтобы сценарии, написанные на языке PHP, могли выполняться Web-сервером, последний необходимо настроить на использование модуля PHP. Для этого выполните следующие действия:

1. Откройте консоль управления IIS (**Панель управления | Администрирование | Internet Information Services**).
2. Откройте окно свойств параметров своего Web-узла. Для этого в окне консоли IIS щелкните правой кнопкой мыши на имени Web-узла (это может быть **Веб-узел по умолчанию**) и в контекстном меню выберите команду **Свойства**.
3. Перейдите на вкладку **Домашний каталог** и щелкните на кнопке **Настройка**. Перейдите на вкладку **Сопоставления** и щелкните на кнопке **Добавить**.

4. В поле **Исполняемый файл** введите путь к интерпретатору PHP, например, C:\php\php-cgi.exe. В поле **Расширение** введите **.php**. Установите флажок **Обработчик сценариев**. Аналогичным образом можно добавить другие расширения файлов, которые должны ассоциироваться с интерпретатором PHP.
5. Щелкните на кнопке **OK**.

Приложение 3

Установка Web-сервера IIS в Windows 2000/XP

В Windows 98 SE программным обеспечением Web-сервера, поставляемым вместе с операционной системой, является Personal Web Server (PWS, персональный Web-сервер), а в Windows 2000/XP Professional — Internet Information Services (IIS, информационные службы Интернета). PWS в Windows 2000/XP Professional не поддерживается. При установке Windows 2000/XP Professional компонент IIS автоматически не устанавливается, и поэтому для его инсталляции нужно выполнить следующие действия:

1. В Панели управления выберите утилиту **Установка и удаление программ**.
2. Щелкните на кнопке **Добавление компонентов Windows**. Откроется окно **Мастер компонентов Windows**.
3. Следуя появляющимся на экране инструкциям, установите IIS (при необходимости удалите или добавьте компоненты). Дополнительно вы можете установить почтовый и FTP-сервер.

Чтобы проверить IIS, установленный на вашем локальном компьютере, запустите браузер и в поле адреса введите **localhost**. В Windows XP Professional откроются два окна браузера со справочной информацией. Здесь можно найти достаточно много сведений по IIS.

Для настройки сервера (с целью, например, создания дополнительных виртуальных каталогов, установки разрешений или для перезапуска сервера в случае неполадок) выполните команду **Пуск | Панель управления | Администрирование | Internet Information Services**. В результате откроется диалоговое окно **Internet Information Services**, с помощью которого можно произвести необходимые настройки.

Папка, используемая для размещения Web-публикаций, называется также *домашним каталогом*. При установке IIS создается домашний каталог \Inetpub\wwwroot, который используется по умолчанию. Все ваши файлы, размещенные в домашнем каталоге, могут быть вызваны из браузера по адресу: **http://*имя_сервера.имя_файла***. Здесь ***имя_сервера*** — доменное, сетевое имя или цифровой IP-адрес компьютера, выполняющего роль сервера. Если сервер установлен на вашем локальном компьютере, то вызов файла можно выполнить с помощью следующего URL: **http://localhost.*имя_файла***. Если сервер расположен на другом компьютере локальной сети и имеет имя, например, myserver, то введите **http://myserver/myfile.php**.

Windows 2000/XP Professional с IIS позволяет размещать на компьютере единственный Web-узел и единственный FTP-узел. Размещение нескольких Web- или FTP-узлов на одном компьютере возможно в версии Windows Server.

Предметный указатель

A

ActiveX 37
Anchor 29

C

Cascading Style Sheets (CSS) 91
Content 6
Cookie 304, 473

F

FTP 31

G

Gopher 31

H

HTML-документ 5
HTTP 31

I

IIS. *См.* Inernet Information Service
Internet Information Services 551

P

Personal Web Server 551

R

RGB 18

S

Session 475
SQL 516

U

Uniform Resource Locator (URL) 4
URL 23, 30
UseNet 31

W

Web design 3
Web development 3
Web-палитра 19
Web-сайт:
- динамический 5
- одностраничный 3
- ресурс 6
- содержимое 6
- статический 5

Web-узел 3
Word Wide Web 31

А

Альтернативный текст 34
Анимация 354
Апплет 42
Атрибут 18
  align 21
  background 18
  behavior 67
  bgcolor 18
  bgproperties 18
  bottommargin 18
  class 22, 95
  color 64
  direction 67
  event 255
  face 64
  for 255
  href 14, 15
  id 22
  language 16
  leftmargin 18
  link 18
  loop 67
  media 94
  noshade 22
  rel 15, 93
  rightmargin 18
  scroll 18
  scrollamount 68
  scrolldelay 68
  size 21, 64
  style 22, 92
  text 18
  title 22
  topmargin 18
  type 69, 93
  vlink, 18
  width 21
  атрибут-событие 22
  тега 7

Б

База данных 515
Баннер 504
Бегущая строка 67

В

Ввод/вывод данных 132
Выравнивание 35
Выражение:
  с операторами 164
  с функциями 177
  элементарное 145, 150, 152, 154, 177, 389, 392, 393

Г

Гиперссылка 6, 22
Гипертекст 6
Гостевая книга 508, 520
Графическая карта ссылок 25
Графический фильтр 110

Д

Деструктор 451
Документ гипертекстовый, 22

З

Заголовок 62
  выравнивание 63
Загрузка изображения
  предварительная 299
Закладка 29
Зарезервированные ключевые слова 143

И

Изображение,
перемещение 311
Имя цвета 18
Исключение 452

К

Календарь 217
Каскадная таблица стилей 91, 274
Класс 443
родительский 444
стиля 95
Класс PHP 444
private 449
protected 449
public 450
базовый 444
наследник 444
ограничение доступа 449
Клиент 4
Кнопка 83
Комментарий:
CSS 93
JavaScript 17, 146
HTML 8
Константы PHP 387
математические 412
создание 387
Конструктор 446
объекта 229

М

Массив:
двумерный 423
одномерный 423
Массив JavaScript 190
двумерный 192
индекс 190
копирование 193
многомерный 192
одномерный 192
Массив PHP 420
автоглобальный *См.* суперглобальный
запись элементов в переменные 430
копирование 425
перемещение 428
регулярный,421
создание 420
сортировка 425
суперглобальный 434
Масштабирование изображения 34
Меню 318
двухуровневое 321, 323
одноуровневое 320
Метод:
alert() 132
confirm() 133
prompt() 134
write() 134
writeln() 135
передачи:
get 77
post 77
обработки строк 180
форматирования строк 185

Н

Навигационная панель 58

О

Область:
активная 25
горячая 25
Объект 443
cssRule 274
document 535
event 258, 260, 541

history 539
Image 299
location 538
navigator 540
popup 295
recordset 345
screen 543
style 274
styleSheet 274
TextRange 352, 543
window 281, 531
Объект JavaScript:
Array 190, 191
Boolean 223
Date 209
document 134
Function 224
Math 207
Number 204
String 178
window 132
пользовательский 228
связанный 232
Объект PHP 444
клонирование 451
создание 448
удаление 451
Окно:
всплывающее 295
модальное 133, 283
немодальное 283
Операнд 145, 389
Оператор JavaScript 145
?: 158
break 161
continue 162
delete 239
do-while 163
for 159
if 153
in 240
instanceof 241
return 171
switch 155
typeof 141
while 162
арифметический 146
декремента 147
инкремента 147
комментария 146
логический (булевый) 152
логическое сложение (ИЛИ) 152
логическое умножение (И) 152
объектный 239
отрицания (НЕ) 152
побитовый 238
присваивания 143, 144, 149
сравнения 150
управления 153, 159
условного перехода *См.* if, switch, ?:
цикла *См.* for, while, do-while
Оператор PHP 389
echo 372, 380
foreach 428
if 396
new 448
switch 397
арифметический 389
декремента 390
инкремента 390
комментария, 389
конкатенации 402
логический (булевый) 393
логическое сложение (ИЛИ) 394
логическое умножение (И) 394
отрицания (НЕ) 394
побитовый 395
приведения к типу 376
присваивания 378, 391
сравнения 392
строковый 391
управления 396
Основной шрифт 64
Отступ 67

П

Пароль 81
невосстанавливаемый 306

Переключатель 80
Переменная JavaScript 142
  время жизни 145
  глобальная 145, 173
  имя 142
  инициализация 144
  локальная 145, 173
  область действия 145
Переменная PHP 377
  глобальная 385, 434
  имя 377
  локальная 385
  область действия 385
  предопределенная 434
  статическая 385
Периодический запуск функции JavaScript 302
Поиск в тексте 352
Поле ввода данных 80
Поля вокруг изображения 35
Принцип короткой обработки 153, 395
Присвоение:
  по значению 378
  по ссылке 379
Псевдокласс стиля 95

Р

Радиокнопка 80
Рамка вокруг изображения 36
Раскрывающийся список 319
  древовидный 331
Расширения ячеек 50
Рисование 358

С

Сеанс 475
Сегментированная графика 25
Селектор 92
  Контекстный 96
Сервер 4
Сессия *См.* сеанс
Система управления базами данных (СУБД) 515
Скрипт 16
Слой 109
Событие 251
  всплывание 264
  захват 264
  прохождение 263
Список:
  иерархический 72, 73, 86
  маркированный 69
  нумерованный 71
  определений 72
  раскрывающийся 84
Ссылка 22
  внутренняя 23, 29
  графическая 24, 36
  относительная 14, 15
  текстовая 23
Статус завершения 371
Стилевой параметр 91
  @font-face 99
  background-attachment 101
  background-color 100
  background-image 101
  background-position 101
  background-repeat 101
  border-color 104
  border-style 104
  border-top-color, border-right-color, border-left-color, border-bottom-color 104
  border-top-style, border-right-style, border-left-style, border-bottom-style 104
  border-top-width, border-right-width, border-left-width, border-bottom-width 103
  border-width 103
  clear 107
  color 100

display 108
float 108
font 99
font-family 98
font-size 98
font-style 99
font-variant 99
font-weight 98
height 102
left 109
letter-spacing 105
line-height 105
list-style 106
list-style-image 106
list-style-position 106
list-style-type 106
margin 103
margin-top, margin-right, margin-left, margin-bottom 103
overflow 108
padding 103
padding-top, padding-right, padding-left, padding-bottom 103
position 109
text-align 106
text-decoration 106
text-indent 107
text-transform 107
top 109
vertical-align 107
visibility 108
width 102
z-index 109
Стили:
логические 62
физические 63
Строка пустая 137
Сценарий 16
клиентский 122, 129
серверный 126
Счетчик посещений:
простой 492
распространяемый 501
сложный 499

Т

Таблица 46, 335
выравнивание 52
выравнивание содержимого ячеек 53
добавление строки 338
отступы в ячейках 50
перемещение по строкам 345
размер 51
рамка 48
расстояние между ячейками 50
расширение ячеек 50
сортировка строк 348
удаление строки 338
фильтрация данных 349
фон 54
цвет 54
Тег:
<a> 95
<b> 63

 35
 91
<i> 63
<label> 90
<option> 85
<pre> 407
<s> 63
<style> 92
<tbody> 46
<tfoot> 46
<thead> 46
<tt> 63
<u>, 63
Тег HTML 6
<!DOCTYPE> 9
<a> 23
<address> 74
<area> 25
<base> 14
<basefont> 64, 66
<big> 66
<blockquote> 67
<body> 17, 36

 20, 21, 53

<button> 83
<caption> 46
 20
<cite> 75
<code> 74
<col> 46, 54
<colgroup> 46, 54
<dd> 72
 19
<dl> 72
<dt> 72
 75
<embed> 38, 41
<fieldset> 89
<font> 64
<form> 76
<FORM> 361
<frame> 57
<frameset> 55
<h1>, <h2>,..., <h6> 62
<head> 10
<hr> 20, 21
<html> 6
<iframe> 59, 293, 315
<img> 26, 33, 40, 45
<input> 80
<kbd> 74
<legend> 89
 69
<link> 15, 93
<map> 25
<marquee> 67
<meta> 10
<nobr> 20, 21
<noframe> 57
<object> 37, 42, 339
 71
<optgroup> 86
<option> 319
<p> 20
<param> 37, 44, 339
<plaintext> 75
<pre> 69
<samp> 74
<script> 16
<select> 84, 319
<small> 66
 19
<style> 16
<STYLE> 274
<sub> 65
<sup> 65
<table> 46
<td> 46
<textarea> 75, 87, 312
<th> 46
<tr> 46
 69
<var> 75
<wbr> 20, 21
<xmp> 75
закрывающий 6
контейнерный 6
открывающий 6
Текстовая область 87
перемещение 312
Текстура 36
Тип данных JavaScript
свободный 135, 374
Тип данных в PHP
примитивный 374
Трансформация графического объекта 116

У

Установочный HTML-файл 55, 285

Ф

Файл:
CSV 339, 489
GIF 354
php.ini 371
закрытие 486
открытие 484

расширение:
avi 40
css 92
htm 6
js 17, 123
map 27
php 126
ra 39
ram 39
swf 37
удаление 486
чтение 487, 488, 489
Флажок 80
Фоновая графика 36
Форма 76, 361
Форум 514
Фрейм 55, 280
обыкновенный 55
плавающий 55, 59, 294
перемещение 315
Функция 436
Функция JavaScript 166
eval() 168, 355
escape() 170
isNaN 168
parseFloat() 168
parseInt() 167
typeOf() 170
unescape() 170
вставки и замены подстрок 187
встроенная 167
обработки числовых массивов 199
пользовательская 171
приведения к типу 138
работы с cookie 308
рекурсивное определение 176
удаления ведущих и заключительных пробелов 188
Функция PHP:
arsort() 427
asort() 426
copy() 469
count()422
date() 417
define() 387
defined() 388
die() 371
empty() 386
eval() 455, 497
exit() 371, 453
fclose() 486
feof() 487
fgetcsv() 489
fgetss() 488
file() 488
file_exists() 486
file_get_contents() 487
fopen() 484
fread() 488
header() 470
imagecreatefromgif() 504
imagegif() 504
imagettftext() 504
include() 371, 385, 465
is_dir() 486
isset() 386, 388, 462
krsort() 427
ksort() 427
mktime() 417
move_uploaded_file() 469
print_r() 383, 424
printf() 407
range() 421
rsort() 427
session_destroy() 477
session_id() 476
session_start() 475
setcookie() 473
sizeof() 422
sort() 425
sprintf() 407
strip_tags() 489
time() 417
unlink() 486
unset() 379, 423, 451
var_dump() 382, 424
встроенная 443
даты-времени 417
математическая 411
переменная 442

перемещения указателя массива 428
пользовательская 436
преобразования:
массива в строку 431
строк 402
строки в массив 431
чисел 414
проверки:
существования 386
типов 386
работы с базой данных 516
форматирования:
строк 407
чисел 415

Ц

Цвет 18
шрифта 64

Ш

Шрифт,
индекс 65

Э

Эквиваленты символов 53, 73
Элемент управления ActiveX:
Flash-проигрыватель 37
STD или TDS 339
TreeView 331
Эффект перспективы 312

Я

Язык HTML 6
Якорь 29